IT University of Copenhagen

Ph.D. Dissertation

Flexible Process Notations for Cross-organizational Case Management Systems

Author:
Tijs Slaats

Supervisor:
Dr. Thomas T. Hildebrandt

Co-supervisor:
Dr. Marco Carbone

Evaluation Committee:

Dr. Wil van der Aalst	Eindhoven University of Technology, The Netherlands
Dr. Hagen Völzer	IBM Research - Zurich, Switzerland
Dr. Andrzej Wasowski	IT University of Copenhagen, Denmark

January 2015

Abstract

In recent times western economies have become increasingly focussed on knowledge work. Knowledge work processes depend heavily on the expert knowledge of workers and therefore tend to require more flexibility then the processes seen in traditional production work. Over-constrained processes cause frustration and inefficiency because they do not allow workers to use their expert experience to make the best judgements on how to solve the unique challenges they are faced with. However some structuring of their work is still required to ensure that laws and business rules are being followed.

IT Systems for process control have a large role to play in structuring and organizing such processes, however most of these systems have been developed with a focus on production work and fail to support the more flexible processes required by knowledge workers. The problem arises at the core of these systems: the notations in which the processes are defined. Traditional process notations are flow-based: control of the process flows from one activity to the next. This paradigm inherently encourages modellers to think in terms of strict orderings instead of supporting flexibility. Flow-based models that do try to capture a large degree of flexibility tend to turn into so-called "spaghetti models", because all possible paths through the process need to be modelled explicitly. Over the last decade new, more flexible, process notations have been researched by using a constraint-based paradigm, where one directly models the rules of a process.

Dynamic Condition Response (DCR) Graphs, which have been developed at the process and systems models group at IT University of Copenhagen (ITU), are one such notation. They stand apart from other constraint-based notations by having a small set of 4 basic constraints, yet offering the full formal expressiveness of both regular and ω-regular languages. They also offer an operational semantics based on transformations of markings, which means that the notation can represent a process at both design- and runtime, facilitating easy reasoning about the execution of the process and techniques for runtime adaptation.

This dissertation reports on the results of the Technologies for Flexible Cross-organizational Case Management Systems (FLExCMS) research project which was started in cooperation between ITU and the company Exformatics A/S. The goals of the project were to strengthen the industrial adoption of constraint-based notations and techniques by further developing DCR Graphs to be industrially applicable, with a particular focus on guaranteeing safety for cross-organizational processes. We will show how DCR Graphs have been extended with new dimensions such as time, data and hierarchy and we will report on the development of techniques for the safe distribution and run-time adaptation of cross-organizational processes based on DCR Graphs. We brought this research into practice at Exformatics by developing tools for modelling and executing DCR Graphs and applying these tools within customer projects, we report on a number of case studies based on these projects. Finally we report on a new angle of research within the Business Process Management field tentatively called Hybrid BPM Technologies, which aims to combine the advantages of both the flow- and constraint-based paradigms.

Acknowledgements

Some people appear to be under the impression that pursuing a PhD is a rather lonely affair, but this has not been my personal experience. Many people have to a greater or lesser extend contributed to the work I will present in this dissertation and without them I would not have been where I am today.

I'd like to start by thanking my supervisor Thomas Hildebrandt, without whom I might never have started out on my path as researcher: Thomas believed in my ability to work in academia long before I had much faith in myself and has been a driving factor in helping me apply for funding. Throughout my project he has been an excellent mentor, pushing my limits on all fronts.

Morten Marquard, as the founder and proprietor of Exformatics, has personally played a large part in getting my PhD project funded. In addition he has been a great company supervisor during the project and had a large hand in successfully applying the research in practice.

Raghava Rao Mukkamala has been my most prominent co-author and has always been a pleasure to work and attend conferences with. I hope we will get to work and travel together again in the future.

Soren Debois comes in as a close second, while we only started cooperating in the last year of my PhD our work together has been extremely rewarding and he made a great addition to Thomas in helping me fully understand some of the more foundational topics we worked on, such as process algebra and session types. In addition, his sharp wit always makes for entertaining discussions.

Michael Westergaard has been an excellent host during my stay abroad at the Architecture of Information Systems group of Eindhoven University of Technology. Working together with Michael was an interesting experience and I was sad to learn that he moved to industry.

During my stay in Eindhoven I met with Hajo Reijers and we discovered a common interested in declarative and hybrid process technologies. I've strongly enjoyed working together with Hajo and hope that we will continue our cooperation in the near future.

Additional thanks go out to Fabrizio Maria Maggi, Marco Carbone, Nobuko Yoshida, Christian Stahl and Francesco Zanitti, who have also all been great to work with.

I would also like to thank Wil van der Aalst, Hagen Völzer and Andrzej Wasowski for joining the assessment committee for this dissertation and take the time to come to Copenhagen for my PhD defence.

Last, but certainly not least, my girlfriend Merete has had the patience of a saint while I worked on my PhD and I consider it a small miracle that I have not been forbidden from further pursuing a career in academia.

Finally I would like to thank all the people that were not mentioned by name, but have been great colleagues regardless during my time at the IT University, Exformatics and my stay abroad in Eindhoven.

Contents

1 Introduction

Over the last few decades the focus of western economies has steadily shifted from production work to so-called knowledge work [10, 5]. Production work tends to be highly structured, take for example a car manufacturing factory: the work is typically done along an assembly line and while there may be a certain amount of variability (the colour of the cars, materials used in the interior, additional features, etc), these variations in the production process tend to be well-defined. Processes in knowledge industries on the other hand tend to be much less rigid: for example legal cases may often share similarities but are rarely exactly the same and the lawyers handling such cases typically have the freedom to make their own decisions in how to handle the various challenges each case presents. Knowledge workers therefore require a new level of flexibility in their work processes [44, 62, 36]: they have strong insights into what the best solution to a problem is and are not helped by being constrained to a standard solution aimed at a generic version of the problem which does not take into account the unique challenges for the specific case they are working on. At the same time some structuring of their work is still required: there may be (legal or business) rules that need to be followed in all cases and these can constrain the ways they are allowed to approach their work.

Within the field of computer science we can employ formal methods to ensure that the software systems developed for process support strictly follow the rules specified for them. This is achieved by modelling both the behaviour of these systems and the rules as mathematical constructs and then using verification techniques such as state-space exploration and syntax checking to ensure that they follow the rules and do not contain any errors (bugs). If a sound mapping exists from the notations used to describe the processes and rules to such mathematical constructs then verification of the models can be automated without the risk of human translation error. Additionally, if the models of the process are directly executable by the workflow system then the running processes can be guaranteed to be correct (assuming that no mistakes were made in the specification of the rules).

Most state-of-the-art process and workflow systems have been developed with a focus on production work and fail to support the more flexible work processes that knowledge workers require. The problem arises at the core of these systems: the notations in which processes and workflows are defined. Traditional workflow notations are grounded in the concept of flow: control of the process progresses from one activity to the next. For example in the case of the car manufacturing factory, the main process will describe in discrete steps how the components of a car are put together along the assembly line. Depending on the features selected for the specific car some of the steps may be skipped, or a choice between different optional components may be made, but the overall process will always follow the same ordering in which the components are put together.

One such flow-based workflow notation is the *Business Process Model and Notation (BPMN)* [39, 61] which is considered the industry standard within the field of Business Process Management [64, 54] and being maintained by the Object Management Group. We will exemplify what a BPMN model looks like by using the process of writing this dissertation as a running example. We recognize that this is mostly a toy example, but found it useful for illustrating these concepts succinctly. For more realistic examples taken from industry we refer to the papers within this dissertation, in particular 4.2, 4.4 and 6.2.

Many of the main concepts underlying BPMN are shared with other notations such as flowcharts and UML activity diagrams [12] and therefore most of the observations made in the following paragraphs can be generalized to these other flow-based notations.

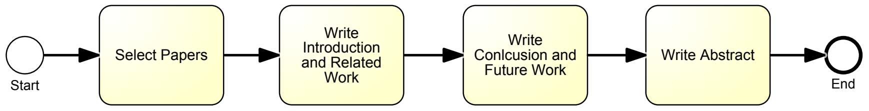

Start
Select Papers
Write Introduction and Related Work
Write Conlcusion and Future Work
Write Abstract
End

Figure 1: BPMN Diagram of Example 1

Example 1. Writing a paper-based dissertation starts by 1) selecting the papers to be used 2) writing the introduction and related work 3) writing the conclusion and future work and finally 4) writing the abstract. This process is drawn using BPMN in Figure 1. The circle at the start is called the *start event* and represents the start of the process, the four rectangles represent the activities listed above and the final bolded circle, called the *end event* represents the end of the process. The arrows are called *sequence flow* and describe the flow between the start of the process, the four activities and the end of the process. To understand the semantics of a BPMN model it can be useful to imagine a token moving through the diagram indicating our current position in the process. The token starts at the start event and by starting the process is moved on to the sequence flow between the start event and the Select Papers activity. When we start the Select Papers activity, the token is moved into its box, indicating that it is currently being performed and when we finish the activity the token is moved onto the outgoing sequence flow, enabling the Write Introduction and Related Work activity to be started. This continues until the token reaches the end event, which ends the process.

A strict ordering as described in Example 1 is not very realistic, in particular this author has a habit of working on different sections intermixedly, as for example ideas for the conclusion may arise while writing the introduction and it can be beneficial to write these down immediately. BPMN and other flow-based languages support having activities occur at the same time with a concept known as *parallelism* which we will demonstrate in the following example.

Example 2. In Figure 2 we have changed Example 1 to allow the activities Write Introduction and Related Work, Write Conclusion and Future Work and Write

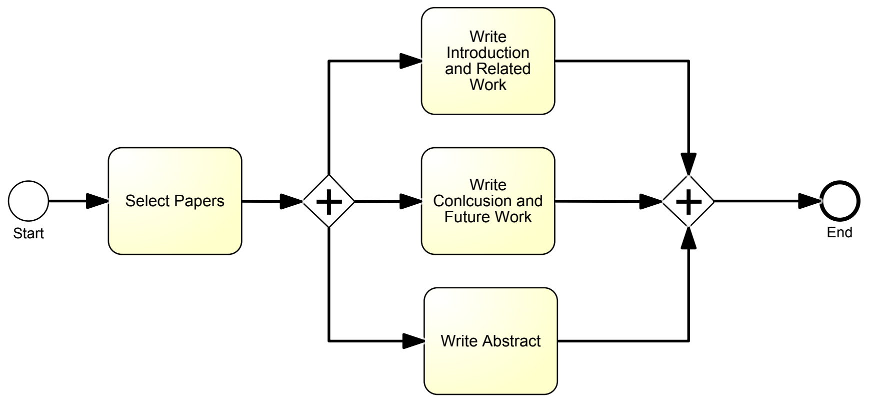

Start
Select Papers
Write Introduction and Related Work
Write Conlcusion and Future Work
Write Abstract
End

Figure 2: BPMN Diagram of Example 2

Abstract to occur at the same time. To do so we use the *parallel split gateway*, drawn as a diamond with a plus sign inside which, using the token-based semantics introduced in the previous example, takes a token on the incoming sequence flow and places an outgoing token on every outgoing sequence flow, enabling all three activities at the same time. We can now move any of the three created tokens as we see fit, doing these activities in any order (or even truly concurrent by doing them at the same time). All the activities lead into a *parallel join gateway*, which takes a token on each incoming sequence flow and places a token on the single outgoing sequence flow, thereby guaranteeing that the process can not end before all activities have completed. Note that the split and join gateways use the same symbol but can be distinguished by the fact that the split gateway has one incoming and multiple outgoing sequence flows, whereas the join gateway has multiple incoming and only one outgoing sequence flow.

While the preceding example gives a good abstract view of the process, it makes a somewhat unrealistic assumption that we know exactly when we are done writing each section of the dissertation. If one was to directly implement the process in an IT system then each activity could occur only once and when it was closed, it could not be reopened. This would force users to be very careful about closing an activity, ie, if the author of this dissertation first finished writing the introduction and closed that activity, then did more work on the conclusion and realised some possible improvements on the introduction, the process as given above would not allow him to make these changes. A real user of such a system would likely learn this limitation and always choose to leave activities open until all can be closed together, but this would be a work-around for a flaw in the process description and instead it is preferable to extend the process model to allow for repetition of these activities. This can be accomplished in flow-based languages by using *decision points* and *loops* as we will demonstrate in the following example.

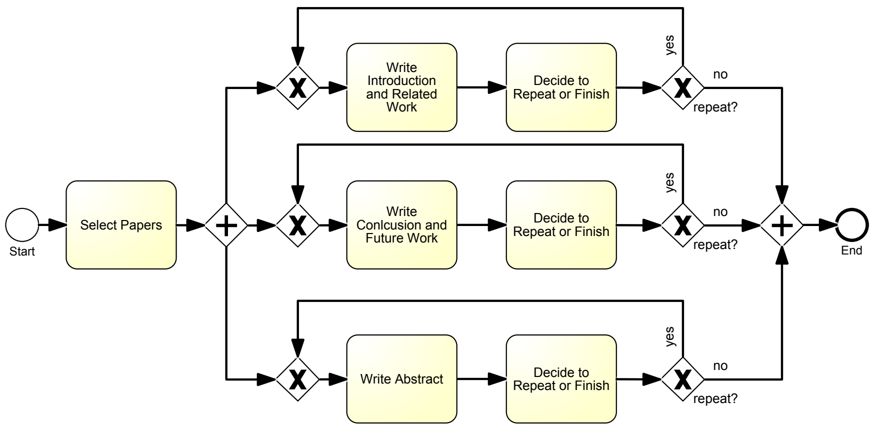

Start
Select Papers
Write Introduction and Related Work
Decide to Repeat or Finish
yes
no
repeat?
Write Conlcusion and Future Work
Decide to Repeat or Finish
yes
no
repeat?
Write Abstract
Decide to Repeat or Finish
yes
no
repeat?
End

Figure 3: BPMN Diagram of Example 3

Example 3. In Figure 3 We extend Example 2 and allow the activities Write Introduction and Related Work, Write Conclusion and Future Work and Write Abstract to be repeated by using the *exclusive (XOR) gateway* construct. Exclusive gateways are similar to parallel gateways in that they denote a point where the control splits into different paths and a join where different control paths merge back together, however, whereas in the case of the parallel gateway control continues on each path, in the case of the exclusive gate way control continues only on one of the possible paths. Which path is taken is based on a condition attached to the gateway. Within BPMN these conditions must be data-based, so if one wants to model a split based on a user decisions then making that decision must be modelled explicitly as an activity before the gateway. In the case of Figure 3 this means for example that when we end the activity Write Introduction and Related Work, we first have an additional activity where we decide if we want to work more on the introduction or finish the dissertation. Typically one will leave this activity open as long as one has not decided to either repeat or finish. After the new activity the token moves into an exclusive gateway with the condition "repeat?", it is assumed that the data necessary to evaluate the condition is provided by the preceding activities. If we chose "yes" the token moves back to exclusive join gateway before the Write Introduction and Related Work activity and we can repeat it, if we choose "no", the token moves on to the parallel join gateway and once all tokens arrive here we can end the process.

As we illustrated in Example 1-3, flow-based notations such as BPMN are not particularly well-suited for describing more flexible processes. The first problem we encountered was that because these notations inherently require one to think in terms of a strict ordering enforced by a notion of flow, a process will often be modelled more rigidly then is required in reality. Secondly, when we do remember to make a flexible model, as in Example 3, we can see that the required number of visual elements in the model starts to grow disproportional to the simple process we are trying to describe. This happens because all the possible paths need to be modelled explicitly. While the example process is relatively small and still understandable, for larger flexible processes this often means that the model turns into a so-called *spaghetti model* and becomes unreadable.

Over the last decade there have been a number of research initiatives to develop notations to support such flexible processes[40, 47, 17, 38, 20], usually by grounding the work in a constraint- instead of flow-based paradigm. In a constraint-based notation one does not model flow, but instead directly describes the rules of the process such as "we need to select papers before we can perform other activities" and "every activity needs to be performed at least once". It is then up to the workflow system to determine all possible actions for the user and allow him the freedom to make his own decisions as long as they fall within the rules (optionally suggesting best-practice solutions, but not enforcing them). Within academia constraint-based notations are commonly called *declarative* notations, whereas flow-based notations are often referred to as *imperative* or *procedural* notations, a practice stemming from an original comparison to declarative (functional) and imperative (procedural) programming languages. Within the papers of this dissertation we primarily use the terms declarative and imperative to refer to respectively constraint-based and flow-based notations.

1.1 Dynamic Condition Response (DCR) Graphs

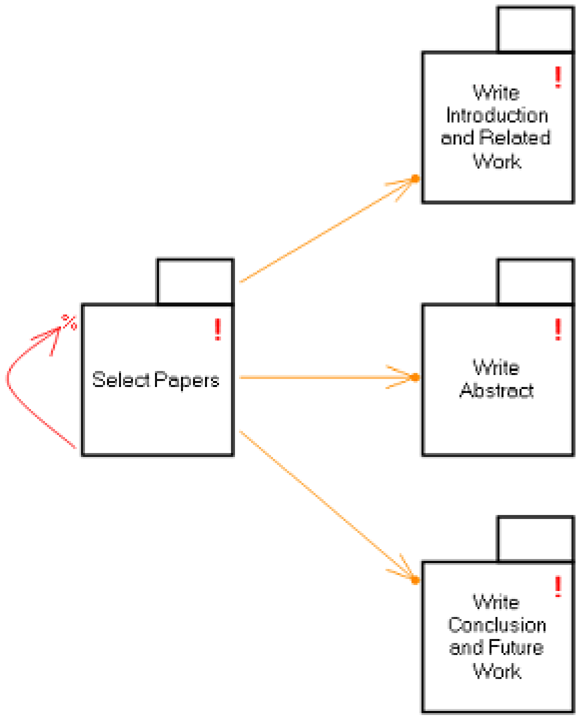
One such declarative notation called *Dynamic Condition Response (DCR) Graphs* [17, 35] was developed at the IT University of Copenhagen by Thomas T. Hildebrandt and Raghava Rao Mukkamala as a part of the TrustCare project[1] and the latter's dissertation [34]. DCR Graphs are a generalization of Event Structures [37, 65, 66, 67, 59] and consist of events (which can be used to model both activities and BPMN events such as timers), a marking over the events and four possible relations between these events. When not constrained (through relations or the marking) events can be executed at any time and any number of times.

The marking of a DCR Graph keeps track of 1) which events have been executed (at least once), 2) which events are still required to be done (often referred to as being pending) and 3) which events are currently included. Events can only be executed if they are included and are also only considered relevant to the process while they are included, this in particular means that while an event is excluded (not included), we can disregard if it is required. This ties into the accepting condition of a DCR Graph which determines when exactly a

[1]Trustworthy Pervasive Healthcare Processes (TrustCare) Research Project - `http://www.trustcare.dk/`

process is allowed to end: we can finish executing a DCR Graph whenever there are no events that are both included and required. There is no default initial marking for a DCR Graph, this means that it is for example possible to have certain events be *initially required*, in essence modelling a to-do list at the start of the process.

The relations describe how events constrain each other and effect the marking: the *condition* relation states than one event can not be executed before another event is executed (unless the latter event is excluded), the *response* relation states that doing one event makes another event required (i.e. doing one activity adds another activity to the list of things that need to be done before we can end our process) and finally the *dynamic exclusion* and *dynamic inclusion* relations respectively remove and add events back into the workflow. We will now show how the process from Example 3 can be modelled as a DCR Graph.



!
Write Introduction and Related Work
!
Select Papers
%
!
Write Abstract
!
Write Conclusion and Future Work

Figure 4: DCR Graph of the process in Example 3

Example 4. The DCR Graph in Figure 4 represents the same process as in Figure 3. Similar to BPMN, the boxes represent events(activities). The ear of the box is reserved for adding roles to the activities, but these are not used in the current example. The relations are drawn as arrows between the events. The red arrow with a percentage sign at the end denotes the exclude relation, in this case stating that the Select Papers activity excludes itself, meaning that when we can execute this activity only once because afterwards it removes itself from the workflow. Note that this rule was implicit in the case of the BPMN model because there is no loop around the activity. The orange arrows with a dot at the end denote the condition relation, in this case Select Papers is a condition for Write Introduction and Related Work, Write Conclusion and Future Work and Write Abstract meaning that these activities can not be performed before the papers for the dissertation have been selected. Finally the exclamation marks on the events denote that they are required in the initial marking, thereby ensuring that they all need to be done at least once before the process can end.

Figure 3 and Figure 4 demonstrate how describing a flexible process can be done more succinctly using DCR Graphs then BPMN. Perhaps counter intuitively, further restricting the behaviour of a flexible process by adding additional rules can make it even more cumbersome to model it in a flow-based notation, while it is fairly straight-forward to do so in a constraint-based notation. We will demonstrate this in the following example.

Example 5. We want to extend the previous example with an additional rule, namely that after making changes to the introduction or conclusion, one always needs to update the abstract to make sure that it accurately represents any new or changed content. In Figure 5 we extended the BPMN model with this new rule, as one can see we require quite a large amount of new elements in the model, most importantly many of the activities are now represented by two separate instances. This is because the first block ensures that each activity is done at least once (this is not ensured by the flower pattern in the second block), while the second part ensures that the abstract is always updated after making changes to the introduction and conclusion. Note that in the first block we also added the possibility to skip writing the abstract because the process model needs to support an execution where one chooses to only work on the abstract once at the end of writing the dissertation. One may argue that only having the first block, followed by only the Write Abstract activity would represent the same language as the given diagram, but note that this would force to user to choose a ”final” update to the abstract, whereas in the model given here the user is still able of making further edits to the introduction and conclusion after each update to the abstract, which is important because the user may realise while updating the abstract that he would like to make further changes to one of the other sections and if he had indicated that this was the ”final” version of the abstract he would not be allowed to do so.

In Figure 6 we extend the DCR graph with the new rule. We add two response relations, drawn as a blue arrow with a dot at its source, which state that 1) whenever Write Introduction and Related Work is done it makes Write Abstract required and 2) whenever Write Conclusion and Future Work is done it makes Write Abstract required. This way we ensure that if either of the two other activities is performed the process can not be completed unless we do Write Abstract afterwards.

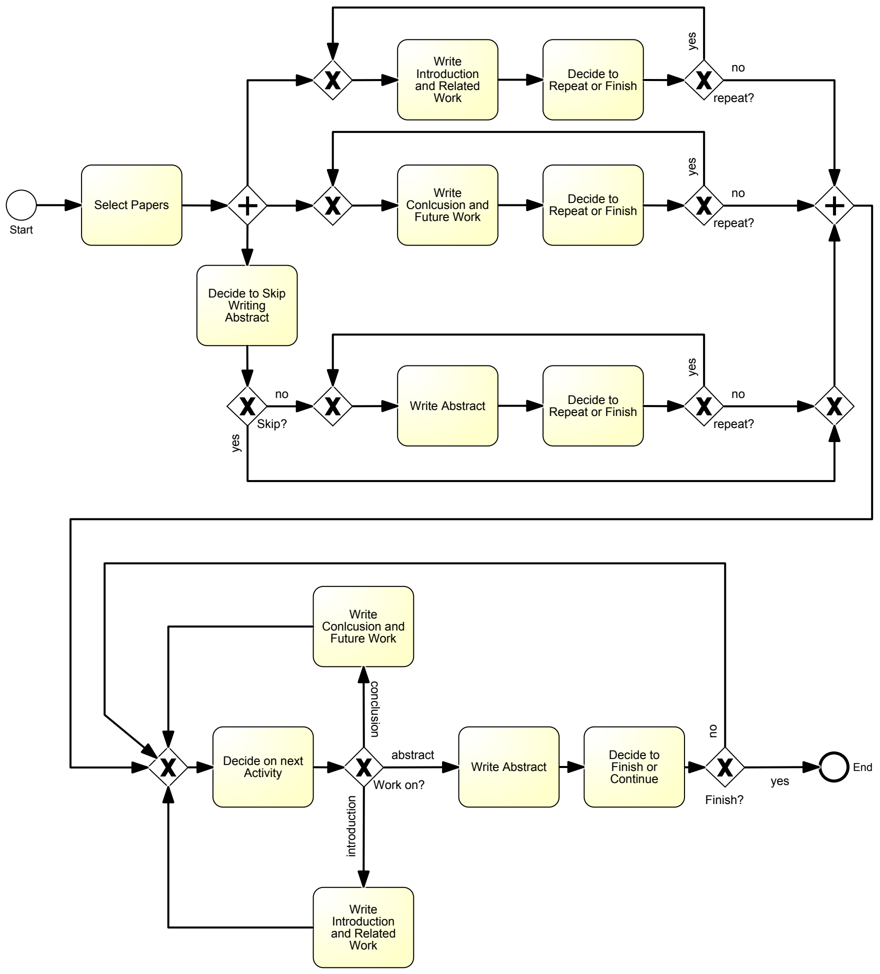

Start
Select Papers
Write Introduction and Related Work
Decide to Repeat or Finish
yes
no
repeat?
Write Conlcusion and Future Work
Decide to Repeat or Finish
yes
no
repeat?
Decide to Skip Writing Abstract
no
Skip?
yes
Write Abstract
Decide to Repeat or Finish
yes
no
repeat?
Write Conlcusion and Future Work
conclusion
Decide on next Activity
abstract
Work on?
Write Abstract
Decide to Finish or Continue
no
yes
Finish?
End
introduction
Write Introduction and Related Work

Figure 5: BPMN Diagram of the process in Example 5

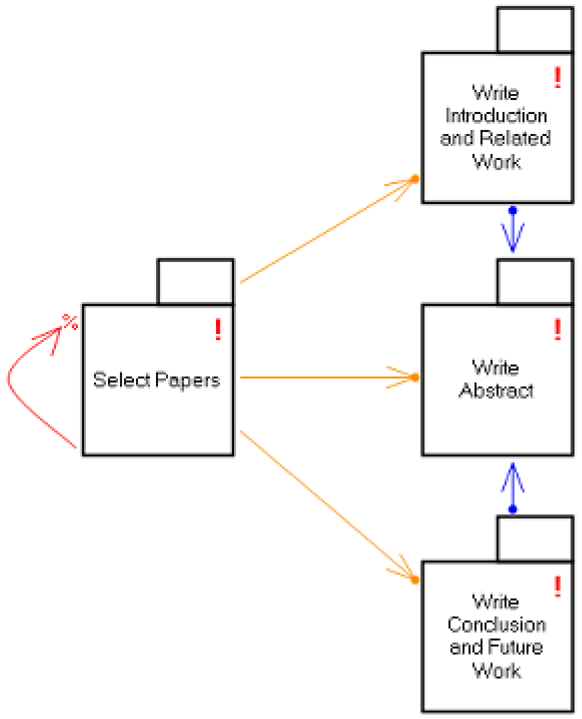

Write Introduction and Related Work
!
Select Papers
!
%
Write Abstract
!
Write Conclusion and Future Work
!

Figure 6: DCR Graph of the process in Example 5

This final example highlights a third advantage of declarative notations, namely that the rules of the process are much more directly embedded in the notation then in the case of flow-based notations. Looking at Figure 5 it is not apparent where exactly in the diagram it is stated that "after making changes to the introduction or conclusion, one always needs to update the abstract", however in Figure 6 the two response relations exactly denote: "after making changes to the introduction one always needs to update the abstract" and "after making changes to the conclusion one always needs to update the abstract". If a modeller who did not originally design the model is asked to remove the new rule again then this will be straightforward in the case of the DCR Graph: one simply removes the two response relations; but in the case of the BPMN diagram this is not as straightforward, for example there is a realistic risk that the modeller removes the second block but forgets to remove the possibility to skip writing the abstract in the first step, resulting in an error in the model where writing the abstract can be skipped altogether.

The semantics of DCR Graphs are usually formalized as step-wise transformations of the marking of the graph, where each step represents the execution of an event. In [35] it has been shown how one can encode DCR Graphs as Büchi automata, thereby providing a straightforward mechanism for verifying them for deadlock, livelock and non-emptiness of their language. In [34] the author expanded on this work by proposing a new verification technique employing the SPIN model checker. In addition it has been shown [34] that the formal expressive power of DCR Graphs is exactly that of the union of regular and omega-regular languages.

1.2 The Technologies for Flexible Cross-organizational Case Management Systems Research Project

While a number of declarative notations and techniques have been developed within academia, this research has not really been adopted by industry yet. Different reasons for this have been suggested [14, 69], in particular the paradigm change from flow- to constraint-based notations not only requires modellers to learn new notations, but also change the way they think about modelling processes. Instead of describing possible process flows as they are used to, they need to learn to specify the constraints and rules of their processes. Secondly declarative notations and techniques are still relatively immature when compared to imperative notations and techniques, for example research on extensions such as hierarchy, time and data is abundant in the imperative world but rather sparse in the declarative community.

In 2011 we began to cooperate with Exformatics A/S, a Danish software developer providing Electronic Case Management (ECM) [58, 46] Systems to knowledge-intensive businesses and organizations in the public and private sector. Exformatics shared our vision that flow-based notations did not offer adequate flexibility for modern ECM systems and together we carried out a case study with one of their customers where we used DCR Graphs to model their processes (reported on in paper 4.2). Exformatics was pleased with the results of the case study and interested in developing tools based on DCR Graphs while continuing the research cooperation in the form of an industrial PhD project. We therefore initiated the *Technologies for Flexible Cross-organizational Case Management Systems (FLExCMS)* project with the overall goal of strengthening and encouraging industrial adaptation of declarative process notations and techniques.

The research goals of the project focussed on further developing the DCR Graphs notation to be industrially applicable, with a particular focus on guaranteeing safety for cross-organizational processes [25, 8, 15, 63]. By guaranteeing safety we both mean ensuring that processes follow the rules, but also that the rules do not give rise to classical problems such as *deadlock* (getting stuck in a process, i.e. one is able to make bad decisions after which it is impossible to move on) or *livelock* (a situation in which one is still able to move in the process, but the available steps do not provide progress towards its goals). By focussing on cross-organizational processes we brought a new dimension to the work that has not been explored for flexible processes notations before and for which the issue of safety is of particular relevance: problems such as deadlock and livelock are prevalent in distributed settings where for example it can happen that two parties are unable to progress in their work because each party is waiting for input from the other (deadlock), or two parties continually "pass the ball" to each other, satisfying their own local rules that require them to pass on their work, but never satisfying the global goal of the process (livelock).

The industrial goals of the project were to develop tools, based on the extended notation and new techniques and methods, for modelling and executing flexible processes and to carry out a number of case studies with customers of Exformatics where we applied these tools and methodologies. These case studies in turn providing feedback for the research activities.

In addition to the original project goals we investigated two more areas of interest. First of all we researched techniques for the safe runtime adaptation of flexible processes. Runtime adaptation of processes is particularly relevant for long-running processes, for example the handling of mortgages by a credit institution, where rules and regulations can change during the lifetime of a process. In such cases running processes need to be adapted to the changes in the regulations without introducing errors to the process. Different approaches can be used to ensure that adaptations are made safely, one can check for each process individually if the adapted process contains the possibility for errors such as deadlock and livelock, but one can also check formally that certain classes of adaptations can always be made without introducing unforeseen issues.

Finally we participated in starting up a new research area within the BPM field which we tentatively call Hybrid BPM Technologies. This new research area is founded on the observation that in reality few domains contain exclusively structured or flexible processes and in many cases even a single process can contain both distinctly structured and flexible parts. For example in hospitals the processes handling patient care may contain both very rigid treatment plans (because they deal with potentially dangerous medicine) and much more flexible treatment plans which leaves a lot of the decision making to the doctor so that he may fine-tune the treatment to specific patients. There has been some work on the overlap and combination of the flow- and constraint-based paradigms [45, 56] in the past, but only more recently (paper 8.1) has it been proposed that investigating such hybrid approaches is a worthwhile field of research on its own with many open research questions.

Based on the original research goals and the new angles of work that arose during the project, this dissertation addresses four primary hypotheses:

1. The declarative DCR Graph model can be extended to cover all the required aspects of flexible workflow, in particular hierarchy, deadlines (time) and data.

2. We can support safe cross-organizational flexible process models by developing distribution techniques for the declarative DCR Graph notation.

3. We can support safe runtime adaptation of flexible process models by developing adaptation techniques for the declarative DCR Graph notation.

4. Creating a bridge between the traditional imperative and new declarative notations will allow for a) more straightforward adoption of the latter and 2) more concise and understandable hybrid models that combine the strengths of the two paradigms.

1.3 Related Work

There is a significant overlap between this dissertation and the work presented by Raghava Rao Mukkamala in his dissertation [34] resulting from our projects partially running concurrently and a large degree of cooperation as we both worked on similar topics. However, whereas Mukkamala's main contribution was to introduce DCR Graphs as a formal model for Declarative processes, this dissertation's main contribution is to extend the notation and techniques to make them industrially applicable. In particular completely new contributions in this dissertation (which will be introduced in more detail further on) are the work on live session types, hierarchical DCR Graphs and hybrid notations and techniques.

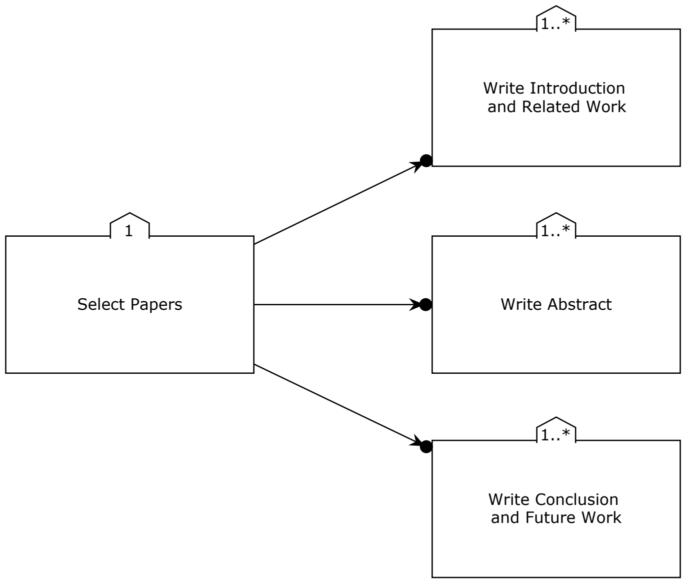

1..*
Write Introduction and Related Work
1
Select Papers
1..*
Write Abstract
1..*
Write Conclusion and Future Work

Figure 7: Declare model of the process in Example 3

Declare

Declare [55, 40, 50] was the first large-scale effort within the BPM community to create a declarative notation for business processes. In essence Declare is a template language: it consists of a fairly large number of predefined constraints between activities which are formalized as Linear Temporal Logic(LTL) formulae. The semantics of a declare model is defined as the conjunction of the formulae of all the constraints in the model. More recent papers have proposed formalizing Declare in other languages such as coloured automata [31] and SCIFF [33, 32]. Declare has gained traction in the academic BPM community and is a commonly used notation when investigating declarative techniques [70], for example common in process mining [29], however to our knowledge there is very little documented usage of Declare in industry. There are a number of factors that set DCR Graphs apart from declare: first of all formal expressiveness results show that DCR Graphs, using only the four basic relations, can express exactly the union of regular and ω-regular languages. To our knowledge there are no published results on the formal expressiveness of Declare. It is sometimes argued that one can always just add new constraint templates and therefore the expressiveness of Declare is that of the underlying language chosen, however this assumes that it is possible to add an infinite number of constraints and requires that the modellers are able of modelling in terms of the underlying language and not just Declare. Secondly the runtime semantics of DCR Graphs are given in terms of DCR Graphs (i.e. as transformations of the marking), this benefits reasoning about DCR Graphs at runtime, allowing for reusing the design-time graphical notation to also visualize run-time models and easing the development of run-time adaptation techniques. The third difference is that because declare models are defined as the conjunction of their constraints, adding additional constraints to a declare model always limits the possible behaviours. This is not the case for DCR Graphs, where relations can affect each other in ways where adding an additional relation to the model actually allows new behaviours, which may be unexpected for inexperienced modellers and meant that we had to develop a theory of refinement for DCR Graphs as discussed in paper 6.4.

Figure 7 shows how example 3 can be modelled using Declare. It looks somewhat similar to the DCR Graph, the main difference being that Declare has cardinality constraints that can be used to state that Select Papers should happen exactly once (denoted by the 1 over the activity) and that the other activties should happen at least once (denoted by the 1..$*$ over the activity). The arrow that looks like the condition relation from DCR Graphs has a similar semantics in Declare, but is named the *precedence* constraint.

Petri nets

Petri nets [41] are a graph-based formal model particularly well-suited to describing distributed and concurrent systems [24] which have become a popular choice within academia for modelling workflows [52, 57, 11]. A Petri net is a bipartite graph that consists of *transitions* and *places*, connected by directed *arcs*. Places represent the presence of resources (which can be for example data, documents and human resources, but also simply the fact that some activity has been performed previously), whereas transitions represent activities and events. The resources themselves are represented as *tokens* which occupy a place when they are present. To *fire* (execute) a transition at least one token needs to be

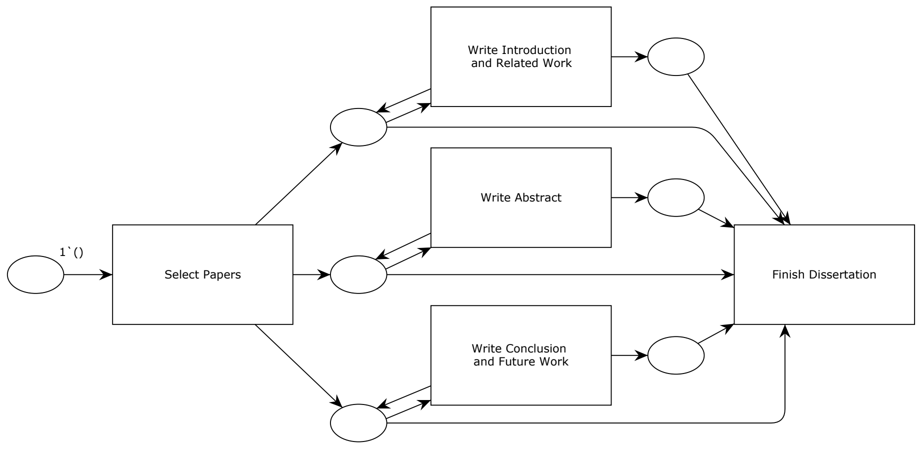

Write Introduction and Related Work
Write Abstract
1`()
Select Papers
Finish Dissertation
Write Conclusion and Future Work

Figure 8: Petri net of the process in Example 3

present on each place connected to it by an incoming arc. When the transition fires these tokens are *consumed* and removed from their places, in their place a new token is created in each place that is connected through an outgoing arc with the transition. There are many variations of Petri nets, some based on limiting the original notation to make certain decision problems easier to analyse [13] such as *WorkFlow nets* (WF-nets) [52, 1], which are aimed specifically at modelling business processes, while others provide extensions to make them more expressive such as *Coloured Petri nets* (CPN) [23, 22, 43] which add types and values to places and tokens and guards to arcs and transitions, allowing for the modelling of traditional data structures and value-based constraints.

Figure 8 shows how example 3 can be modelled as a Petri net. On the left is an initial place with a single token in it (denoted by the 1‘()), executing Select Papers removes this token and places a token in each of the following three places. We can now no longer execute Select Papers because there is no token in the initial place, but we can execute all the other activities. Each of the other three activities will take a token from their preceding place and put it back after executing, thereby ensuring that they can be executed again any number of times. They will also place a token in an additional place to their right, which counts the number of executions for each activity. To signal when we are done with the dissertation we added a Finish Dissertation activity. It will only be enabled when all activities have been done at least once and when firing will remove the three tokens that enable the writing activities, thereby disabling all activities and ending the process. One could do without the Finish Dissertation activity if one defines an acceptance criteria for the Petri net where any marking in which each of the three counting places on the outer right has at least one token in it is accepting.

Process Mining

Process mining [53] is an emerging topic within the BPM field with the aim "to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's (information) systems" [49]. As such it can be seen as a specialization of data mining and machine learning and indeed many process mining approaches make use of existing data mining and machine learning techniques, such as association rule mining [28] and inductive logic programming[4, 26]. Supported by strong interest from industry, where process mining techniques can be used to get insights into the way companies actually do their work in practice and thereby improve process efficiency, the topic has received increased attention in the business process management community, leading to the creation of an special task force on the subject[2]. One particular area of interest in the process mining community is process discovery [48, 2], which concerns itself with developing algorithms that based on event logs attempt to build a models of the underlying processes that could have generated the logs. In the last few years many different process discovery techniques have been proposed both for flow-based models, in particular Petri nets [51, 27] and declarative models [28, 4, 26, 9, 30].

Session Types

Session types [18, 68, 60] are a behavioural typing system, meaning that their focus is not on typing data but instead the behaviour of processes. The goal is therefore not to guarantee type-compatibility of variables and expressions, but instead behavioural compatibility of interacting π-calculi processes. The session type specifies the protocol that process interactions should adhere to, usually referred to as *session fidelity*. Session types have the additional benefit that well-typed processes are guaranteed to be deadlock free, which provides a means of verifying safety of processes by syntax checking instead of state-space exploration. Session types have seen many extensions, for example to multi-party session types [19] which allow the specification of protocols with more then two parties. It has also been proposed that session types can be used to provide a behavioural typing system for BPMN processes which involve communication between distributed actors modelled as choreographies [7].

Guard-Stage-Milestone model

The *Guard-Stage-Milestone* (GSM) model [21] developed at IBM Research is based on the earlier work on artifact-centric business processes [3]. A GSM model is divided into *stages* which contain the atomic tasks of the model. A stage can have *guards* which need to be satisfied before the stage becomes active and its tasks can be executed. A stage can also contain *milestones* which can be seen as the acceptance criteria of the stage, through performing the tasks of the stage the milestones can become satisfied, which can in turn be part of satisfying

[2]IEEE CIS Task Force on Process Mining - `http://www.win.tue.nl/ieeetfpm/`

the guards of other stages. While GSM has a distinct declarative flavour it is mainly a data-centric model. The GSM model has had a strong influence on the development of the Case Management Model And Notation (CMMN) [38], which is a new standard notation developed by the Object Management Group aimed in particular at case management.

2 Structure of the Dissertation

The body of this dissertation consists of the papers that resulted from the FlexCMS project, separated into the following five areas of interest:

Hierarchical Dynamic Condition Response Graphs

In section 4 we present four publications that deal with adding different forms of hierarchy to DCR Graphs. The papers focus on validating the first part of hypothesis 1, by showing that indeed DCR Graphs can be successfully extended with a notion of hierarchy.

In *Nested Dynamic Condition Response Graphs* we introduce an extension to DCR Graphs that allows for the nesting of events. Nesting in this case is mainly cosmetic: only atomic events are executable, while higher-level events are used to group sets of events together and relations to and from super-events apply to all their atomic sub-events. This can be used to make process diagrams more succinct, for example if a process contains two phases and doing all the activities in the first phase is a prerequisite for the activities in the second phase, then we can use two super-events to represent the phases and draw a single condition relation between them (instead of $n \times m$ condition relations for each pairing of events between the first and second phase). Another main contribution of the paper is the milestone relation which inherently captures the accepting condition of DCR Graphs, namely that for one event to be executable another event needs to either not be required or be excluded. Using this relation together with nesting means that one can disable a phase as long as all the required tasks in a previous phase have not been performed yet.

In the second paper, *Designing a Cross-organizational Case Management System using Dynamic Condition Response Graphs*, we report on a case study performed together with Exformatics and Landsorganisationen i Danmark (LO), an umbrella organization for trade unions in Denmark. During the case study we used nested DCR Graphs to model the main process underlying the cross-organizational case management system aimed at handling conflicts between employees and their employers that Exformatics was developing for LO.

In *Hierarchical Declarative Modelling with Refinement and Sub-processes* we introduce Hierarchical Dynamic Condition Response (Hi-DCR) Graphs, which add a notion of dynamically spawned sub-processes to DCR Graphs. A sub-process is in itself a Hi-DCR Graph that is spawned as a side-effect to executing an event. In the sub-process we distinguish between interface and local events: interface events are merged with equally named events in the parent process, whereas local events are always copied into the main process with a new unique name, allowing one to define multi-instance sub-processes. The spawning of a sub-process is formally defined through the composition of the main graph with the sub-process graph and the paper also identifies a class of compositions that guarantee language-refinement, which is investigated further in paper 7.3.

The final paper *A Case for Declarative Process Modelling: Agile Development of a Grant Application System* reports on the development of a case management system for a Danish funding agency by Exformatics. The processes for handling funding requests were modelled as DCR Graphs and these models were used directly as executable code within the implementation of the case management system for workflow enactment. The paper has been included in this section because the project demonstrated a clear need for support for multi-instance sub-processes and (an abstracted version of) one of the processes identified during the case study has been used in the previous paper as a running example.

Dynamic Condition Response Graphs with Time and Data

In section 5 we present two publications that deal with adding time and data dimensions to DCR Graphs. The papers primarily focus on validating the second part of hypothesis 1, by showing how DCR Graphs can be successfully extended with a notion of time and data.

The first paper, *Contracts for Cross-organizational Workflows as Timed Dynamic Condition Response Graphs*, is a journal publication in which we extend DCR Graphs to support discrete time deadlines. We also provide a technique for verifying safety and liveness properties through a mapping to finite state transition systems. We also prove that the general technique for safe distribution of DCR Graphs provided in Paper 6.1 can be extended to timed DCR Graphs. We exemplify the use of timed DCR Graphs and the distribution technique in practice on a timed extension of the process from paper 4.2.

In the second paper, *Exformatics Declarative Case Management Workflows as DCR Graphs*, we give an informal presentation of an extension to DCR Graphs with data, show how DCR Graphs are used by Exformatics to model workflows through a case study of an invoice workflow and give an overview of the tools that have been developed by Exformatics to support working with DCR Graphs. We report on the use of these tools both in commercial projects and in a bachelor level course at ITU.

Safe Cross-Organizational Flexible Processes

In section 6 we present four publications that deal with different techniques aimed at supporting cross-organizational processes. The papers focus on validating hypothesis 2 by developing distribution techniques for DCR Graphs and laying the foundations for bridging the work on DCR Graphs to the highly relevant work on session types.

In *Safe Distribution of Declarative Processes* we present a technique that, given a declarative global description of a process and a separation of the activities of the process over collaborating actors, projects a local declarative process for each actor that only describes the parts of the process that are relevant for them. We show how these local processes can be safely executed in a synchronous distributed setting by only communicating the execution of activities or events, all the while adhering to the original global specification. We exemplify the distribution technique on the process identified in the case study that we reported on in paper 4.2.

In *Declarative Modelling and Safe Distribution of Healthcare Workflows* we present a case study based on an oncology treatment process from a Danish hospital that applies the techniques developed in the previous paper. We also extend the techniques from the previous paper to cover nesting and the milestone relation as defined in paper 4.1.

In the third publication, *RSVP: Live Sessions with Responses* we propose extending session types to allow for specifying liveness properties. Concretely we do so by introducing sessions with responses, where branching and selection labels can be annotated with the expected responses given in conjunctive normal form (without negation). A process is considered live when whenever a label is selected, one of the labels in each of the disjunctive responses is eventually selected, similar to the response semantics of DCR Graphs. The typing system for guaranteeing that well-typed processes are live in this sense is given informally.

The fourth paper *Type Checking Liveness for Collaborative Processes with Bounded and Unbounded Recursion* continues the work initiated in the previous paper. Unlike the other paper we define responses as a single set of labels, however we give a formally proven sound typing system guaranteeing request-response liveness of well-typed processes and prove that the extended types are strictly more expressive than standard session types. We apply the new types to a process calculus similar to collaborative BPMN processes and exemplify the use of the calculus and type system on a concrete example.

Safe Runtime Adaptation of Flexible Processes

In section 7 we present three publications that deal with different techniques aimed at runtime adaptation of flexible processes. The papers mainly focus on validating hypothesis 3 by developing several approaches that can be used for the run-time adaptation of DCR Graphs.

In *Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs* we discuss techniques for safe run-time adaptation of DCR Graphs. We introduce three basic operations that can be used for adapting DCR Graphs: compose, change and discard. Composition allows one to add new events and relations to a DCR Graph by composing it with a process fragment defined as a second DCR Graph that contains exactly the events and relations one wants to add. The change operator allows one to change the names of events and labels. The discard operation allows one to remove events and relations from the Graph. The composition and discard operations can also be used to modify the run-time marking of a DCR Graph. To ensure safety of adapted DCR Graphs we show how they can be verified for deadlock freedom and liveness after adaptation.

In *Modular Context-Sensitive and Aspect-Oriented Processes with Dynamic Condition Response Graphs* we show how DCR Graphs, supported by the techniques introduced in the previous paper, can be used as a formal foundation for the modular design and implementation of context-sensitive and aspect-oriented processes. We exemplify the approach with a context-sensitive authorization process.

The final paper in this section *Towards a Foundation for Modular Run-time Adaptable Process-Aware Information Systems?* is the only draft paper in this dissertation. It is an extension of the work started in paper 4.3, but here we instead focus on the refinement of processes through composition by introducing the DCR Process Language, a calculus-based representation of DCR Graphs providing easier reasoning about the composition of processes. We show how run-time adaptation can be achieved through composition and when such adaptations can be considered safe refinements. The paper also contains formal proofs for the expressiveness results of both regular DCR Graphs and Hi-DCR Graphs.

Hybrid BPM Technologies

In section 8 we present three publications that investigate hybrid process notations and techniques. The papers lay the foundations for validating hypothesis 4 by 1) documenting an industrial interest in hybrid process notations, 2) proposing such a notation and 3) initiating work on developing relevant technologies that use hybrid notations.

In *Declarative Modeling - An Academic Dream or the Future for BPM?* we present a study performed at a Dutch developer of ECM software on what practitioners think of declarative modelling. We conclude that the practitioners involved in the study are receptive to the idea of a hybrid approach combining imperative and declarative techniques, rather than making a full shift from the imperative to the declarative paradigm. Moreover, we report on requirements, use cases, limitations, and tool support of such a hybrid approach. Based on the gained insight, we propose a research agenda for the development of this novel modelling approach.

In *Mixing Paradigms for More Comprehensible Models* we propose an approach to modelling workflows that combines colored Petri nets with the declarative languages Declare and DCR Graphs. The combined approach makes it possible to use both imperative and declarative constructs in a single language and allows one to model data in both Declare and DCR Graphs. We provide considerations necessary for enactment and describe how the approach has been implemented in CPN Tools 4.

In *The Automated Discovery of Hybrid Processes* we introduce an algorithm for mining hybrid process models. The approach first identifies structured and flexible parts of the log and then mines these parts separately using respectively a declarative or flow-based miner. This results in a number of declarative and/or flow-based sub-processes which are then combined in a root process that is itself modelled either declarative or flow-based. The approach can be used iteratively to generate a multi-level sub-processes hierarchy. In our implementation we use Declare as a declarative notation and Petri nets as a flow-based notation, but the approach does not prescribe the use of specific miners and other notations could be used. We evaluated the approach on the BPI 2012 challenge log.

3 Conclusion

Overall we believe that we brought the FlexCMS project to a very successful conclusion. The two initial hypotheses of the project were validated by 1) developing notions of hierarchy, time and data for DCR Graphs and 2) developing techniques for the safe distribution of such declarative models. The third hypothesis, which arose during the project, was successfully validated by developing two distinct approaches for the run-time adaptation of DCR Graphs. Finally we laid a clear foundation for further investigating the usability and feasibility of hybrid process notations. Further experimental research is needed to verify the final hypothesis, but this is not surprising as it arose closer to the end of the project and was not a part of the initial goals. More precisely we produced valuable research results in the following areas:

We investigated two different methods for adding hierarchy to DCR Graphs, the first being Nested DCR Graphs, an extension aimed at representing complex diagrams with fewer visual elements and thereby improving understandability of the models. As a part of this work we also introduced the additional milestone relation, which has proven to be useful for modelling acceptance criteria in processes. While it can be proven that the relation does not make the notation formally more expressive, it is also not readily apparent how the relation can be modelled as a pattern using only the other basic relations. The second method, Hierarchical DCR (Hi-DCR) Graphs, offers a semantic extension that adds support for multiple instance sub-processes to DCR Graphs which was seen as an important feature by the industrial partner to model the processes that they were encountering in practice. We believe that these two extensions together greatly improve the usability of DCR Graphs.

We also developed extensions for time and data, respectively supporting the modelling of deadlines and delays/durations and the modelling of global variables that can be used as guards on both events and relations, satisfying two more needs from industry for modelling real-life processes.

To support cross-organizational processes we researched a technique for safely projecting a DCR Graph over sets of events. The projection technique ensures that the projected graphs can be executed on separate platforms as long as communication between them is performed synchronously. A strong benefit of this approach is that given a global view of how the entire process should work one can generate local views for the participating parties that only show the activities that directly relate to their work, allowing for a distribution of concerns where the collaborators in a process do not necessarily need to be aware of the details of the other parties role in the process. We also extended this technique to work on Timed DCR Graphs. In addition we worked on extending the theory of session types with the ability of checking for liveness constraints, which enables verification of processes for safety and liveness properties based on syntax checking instead of state space exploration. Making the mapping between DCR Graphs and session types is left for future work, once this has been completed the approach will also allow for DCR Graphs to be combined with processes specified in other notations as long as their shared interfaces are defined using session types.

We developed two methods for run-time adaptation. The first method defines a number of operators that can be used for making both incremental and decremental changes to DCR Graphs and provides techniques for checking the adapted graphs for deadlock and livelock. The second method allows only incremental updates to a DCR Graph through composition, but also identifies a safe subset of adaptations that guarantee refinement of the original model. Note that the terms incremental and decremental here are used in terms of the elements (events, relations, etc.) of the DCR Graph and not the behaviour of the DCR Graph. In fact, incremental updates to a DCR Graph can both decrease and increase the possible behaviours: one can add a new event, enabling an activity that was not present before, or one can add an unsatisfiable condition to an existing event, disabling an activity that was enabled before.

We also reported on a number of case studies: we showed how nested DCR Graphs have been used to model the case management system of a Danish umbrella organization of trade unions, how the projection technique can be used to safely distribute the responsibilities of different actors in the oncology workflow of a Danish hospital and how DCR Graphs were used by Exformatics to design and develop a case management system to support the grant application process of a Danish foundation.

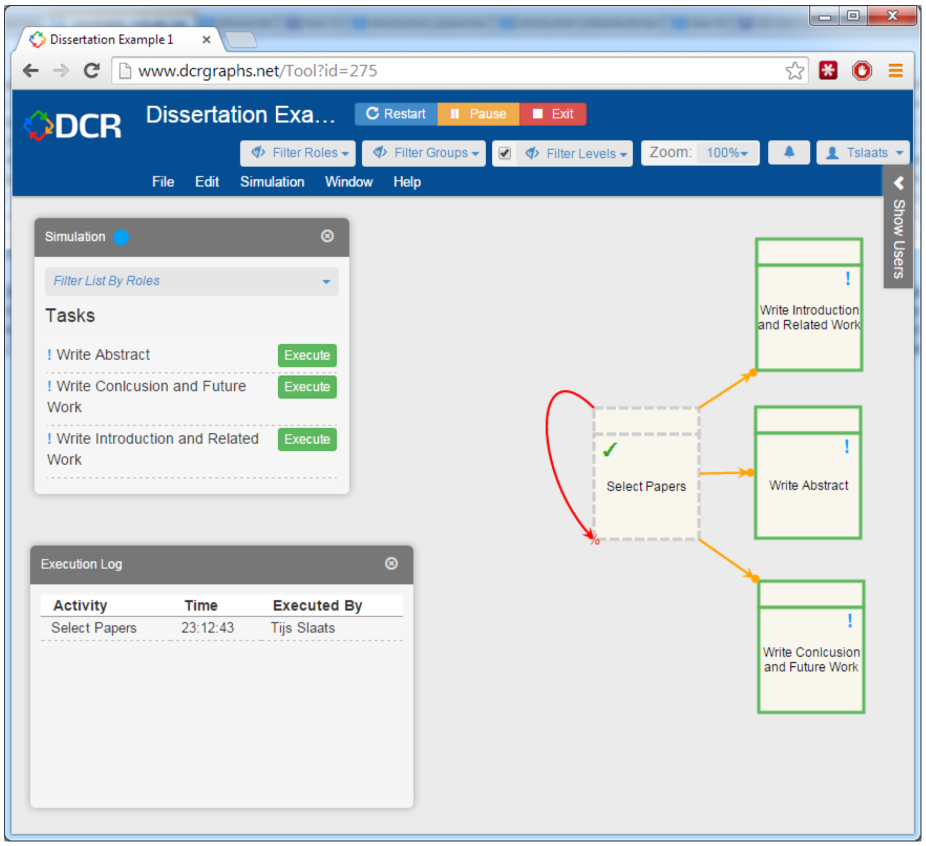
Finally we participated in opening up a new research area in the BPM field focussing on Hybrid Process Technologies, which make use of both imperative and declarative notations and techniques. In particular we set out a research agenda for the new field, proposed how specific imperative and declarative notations could be combined and investigated process discovery of hybrid processes.

We feel that these research results have made a significant contribution to the field of declarative process modelling. To our knowledge we were the first to investigate techniques for supporting cross-organizational declarative processes and by extending DCR with time, data and hierarchy we enabled declarative notations to be used for modelling more complex real-life processes. Being able of supporting run-time adaptation has also shown to be a vital property for the industrial partner, who regularly encounters situations where running processes need to be adapted to the changing needs of customers. The work on hybrid process notations and techniques arose during the project and was not planned in advance but appears to be well received by the community with several other research groups showing interest in such notations and the interplay between declarative and imperative notations in general [6, 16, 42].

This dissertation has been primarily focussed on presenting the academic results of the authors PhD project, however in the next subsection we will shortly touch upon the industrial results of the project as well. Afterwards we will discuss opportunities for future work.

3.1 Industrial Results

Paper 5.2 reports on the tool development that was taking place at Exformatics around the mid-way point of the project, which included a webservice-based



Tasks

! Write Abstract — Execute

! Write Conlcusion and Future Work — Execute

! Write Introduction and Related Work — Execute

Activity	Time	Executed By
Select Papers	23:12:43	Tijs Slaats

Figure 9: Screenshot of the Web-based DCR Graphs Editor

engine for executing DCR Graphs and a Windows-based graphical modelling tool. Since then we have focussed on developing a new web-based modelling tool that is available on all platforms, shown in Figure 9. Whereas the original tools were developed by the author and the modelling tool in particular was mostly a prototype, the new tool has been developed mainly by the development team at Exformatics. This has facilitated the dissemination of knowledge generated during the project within the company and provided the resources to quickly develop a tool of production quality. At the BPM 2014 conference we organized a tutorial on process modelling with DCR Graphs where we used this online modelling tool. The tutorial consisted of 3 parts: first we gave a 20 minutes presentation explaining the basic principles behind DCR Graphs and showed how to use the tool to model a basic process. Afterwards we gave the audience a number of assignments to solve, the first few assignments concerned basic usage of the tool, whereas the last assignment had the audience model a process by themselves. Finally we asked the audience to fill out a questionnaire on the perceived usability of both DCR Graphs themselves and the tool. We were pleasantly surprised by the reactions given in the questionnaire, overall both the tool and DCR Graphs themselves scored significantly above average in regards to understandability and usability. In all fairness the tutorial was done with an academic audience, which may not be a fair representation of the actual users of DCR Graphs. The near future we plan to do a number of similar workshops with regular business users and publish the results.

During the project the attitude towards DCR Graphs within Exformatics changed, originally the intention was mainly to use DCR Graphs as a workflow language within their proprietary Electronic Case Management System, but during the project and especially after the development of the online modelling tool the company is now considering also offering more consultancy-oriented services, where DCR Graphs are used to model and analyse customers processes and where the tools are offered as a product of their own without needing to be embedded within the ECM system. We believe that this development shows that there is a strong confidence within Exformatics that DCR Graphs are a useful tool for modelling and developing flexible processes.

3.2 Future Work

During the project we worked on formally mapping DCR Graphs to other notations such as Petri nets, Declare and the Guard-Stage-Milestone model. Because of other angles taking priority none of these initiatives were finalized and published, but we still believe they hold merit and it would be useful to continue the work as time allows. In the case of mapping to Petri nets, the work would also tie in nicely with the work on Hybrid Process Technologies, as being able of formally mapping between DCR Graphs and Petri nets would facilitate using them together in a hybrid approach.

The long-term goal of adding liveness constraints to session types is to be able of combining the work on session types and DCR Graphs and thereby be able of specifying the interfaces of declarative processes as behavioural types. Introducing a calculus representation of DCR Graphs is another step in this direction, but a true mapping that would allow session types to be used together with DCR Graphs still needs to be researched.

Paper 8.1 made a first step towards empirically testing the perceived understandability and usability of DCR Graphs by actual practitioners. At BPM 2013 we followed up on this initial effort by organising a hands-on tutorial session on DCR Graphs, including a questionnaire to be filled out by the participants aimed at gauging the understandability of the notation. We believe this work has a lot of merit and it would be good to continue investigating the usability of DCR Graphs (and declarative notations in general) in the future, both as a way to improve upon the notations by identifying their weak points and build trust in their actual applicability.

One area of future work that is of particular interest to the author is that of hybrid BPM technologies as we believe there are many open research opportunities in this field. In terms of notations it would be interesting to look at combining declarative notations with BPMN. Because it is the flow-based notation most widely adopted by industry a combined approach that supports it would have the best chance of reaching practitioners, with the additional benefit of making practitioners more aware of declarative approaches.

We also believe that it would be worthwhile to investigate existing notations and evaluate to what extend they already fall within the hybrid paradigm. Petri nets are a good example of a notation that we believe has both flow- and constraint-based aspects to it: while the tokens moving around in a Petri net have a strong sense of flow to them, arcs blocking transitions from firing have a strong resemblance to constraints. Of particular note is that transitions that do not have any incoming arcs are allowed to fire at any time and any number of times, similar to how unconstrained activities can fire at any time in DCR Graphs and Declare, something that is often considered a main property of declarative notations. As we noted in paper 8.2 this property makes Petri nets particularly well-suited to combining with the more traditional declarative notations.

In terms of hybrid process mining the approach we developed in paper 8.3 still has ample room for improvements, in particular it is not very good at distinguishing parallelism from flexibility and only allows activities to occur in a single sub-process.

We are also interested in other hybrid approaches, for example the possibility of using flow-based and declarative notations together, but not necessarily intermixed within a single model, for specifying different properties of a process. For example one could use declarative notations for the initial requirements specifications of a system and then derive (guided by user input) one or more flow-based models that implement the desired processes from these rules, which could be used to implement the actual running processes. In this case it would be up to the modeller that derives the flow-based models from the constraint-based specification to decide how much flexibility should be maintained in the actual implementation. By logging the derivation steps taken it would also be possible to relate the flow-based implementation back to the constraint-based specification that gave rise to it.

List of Publications

Journal publications

1. *Contracts for Cross-organizational Workflows as Timed Dynamic Condition Response Graphs*, Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats and Francesco Zanitti, In Journal of Logic and Algebraic Programming Special issue on Contract Oriented Software.

Conference publications

1. *Hierarchical Declarative Modelling with Refinement and Sub-processes*, Soren Debois, Thomas Hildebrandt and Tijs Slaats, 12th International Conference on Business Process Management (BPM 2014).
2. *The Automated Discovery of Hybrid Processes*, Fabrizio Maria Maggi, Tijs Slaats and Hajo A. Reijers, 12th International Conference on Business Process Management (BPM 2014).
3. *A Case for Declarative Process Modelling: Agile Development of a Grant Application System*, Sren Debois, Thomas Hildebrandt, Morten Marquard and Tijs Slaats, 3rd International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (AdaptiveCM 2014).
4. *Type Checking Liveness for Collaborative Processes with Bounded and Unbounded Recursion*, Soren Debois, Thomas Hildebrandt, Tijs Slaats and Nobuko Yoshida, Accepted for 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE 2014).
5. *Dynamic Condition Response Graphs for Trustworthy Adaptive Case Management*, Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats and Morten Marquard, In Proceedings of International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (AdaptiveCM 2013).
6. *Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs*, Raghava Rao Mukkamala, Thomas Hildebrandt and Tijs Slaats, In Proceedings of The Enterprise Computing Conference (EDOC 2013).
7. *Exformatics Declarative Case Management Workflows as DCR Graphs*, Thomas Hildebrandt, Morten Marquard, Raghava Rao Mukkamala and Tijs Slaats, In Proceedings of International Conference on Business Process Management (BPM 2013).
8. *Declarative ModelingAn Academic Dream or the Future for BPM?*, Hajo A. Reijers, Tijs Slaats and Christian Stahl, In Proceedings of International Conference on Business Process Management (BPM 2013).

9. *Mixing Paradigms for More Comprehensible Models*, Michael Westergaard and Tijs Slaats, In Proceedings of International Conference on Business Process Management (BPM 2013).

10. *Rsvp: Live sessions with responses*, Thomas Hildebrandt, Marco Carbone, and Tijs Slaats, In Proceedings of 1st International Workshop on Behavioural Types (BEAT13).

11. *Modular Context-Sensitive and Aspect-Oriented Processes with Dynamic Condition Response Graphs*, Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats and Francesco Zanitti, In Proceedings of Foundations of Aspect-Oriented Languages workshop (FOAL 2013).

12. *Safe Distribution of Declarative Processes*, Thomas Hildebrandt, Raghava Rao Mukkamala and Tijs Slaats, In Proceedings of 9th International Conference on Software Engineering and Formal Methods (SEFM'11).

13. *Designing a Cross-organizational Case Management System using Dynamic Condition Response Graphs*, Thomas Hildebrandt, Raghava Rao Mukkamala and Tijs Slaats, In Proceedings of Fifteenth IEEE International EDOC Conference (EDOC'11).

14. *Declarative Modelling and Safe Distribution of Healthcare Workflows*, Thomas Hildebrandt, Raghava Rao Mukkamala and Tijs Slaats. In Proceedings of 1st International Symposium on Foundations of Health Information Engineering and Systems (FHIES'11).

15. *Nested Dynamic Condition Response Graphs*, Thomas Hildebrandt, Raghava Rao Mukkamala and Tijs Slaats, In Proceedings of 4th International Conference on Fundamentals of Software Engineering (FSEN'11).

References

[1] Wil M. P. van der Aalst. Verification of workflow nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, ICATPN '97, pages 407–426, London, UK, UK, 1997. Springer-Verlag.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '98, pages 469–483, London, UK, UK, 1998. Springer-Verlag.

[3] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *In preparation*, pages 288–304, 2007.

[4] Federico Chesani, E. Lamma, Paola Mello, M. Montali, F. Riguzzi, and S. Storari. Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-Aware Information Systems*, 5460:278–295, 2009.

[5] Thomas H Davenport, Sirkka L Jarvenpaa, and Michael C Beers. Improving knowledge work processes. *Sloan management review*, 1996.

[6] Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. Multi-paradigm process mining: Retrieving better models by combining rules and sequences. In Robert Meersman, Herv Panetto, Tharam Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzzocrea, and Timos Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, volume 8841 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin Heidelberg, 2014.

[7] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, pages 194–213, 2012.

[8] Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. Amoeba: A methodology for modeling and evolving cross-organizational business processes. *ACM Trans. Softw. Eng. Methodol.*, 19(2):6:1–6:45, October 2009.

[9] C. Di Ciccio and M. Mecella. Mining constraints for artful processes. In *BIS*, 2012.

[10] Peter F. Drucker. *Management Challenges for the 21st Century.* HarperBusiness, 2001.

[11] Rik Eshuis and Juliane Dehnert. Reactive petri nets for workflow modeling. In *Applications and Theory of Petri Nets 2003*, pages 296–315. Springer, 2003.

[12] Rik Eshuis and Roel Wieringa. A formal semantics for uml activity diagrams-formalising workflow models. 2001.

[13] J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - a Survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.

[14] Dirk Fahland, Daniel Lbke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 353–366. Springer Berlin Heidelberg, 2009.

[15] Paul Grefen, Karl Aberer, Heiko Ludwig, and Yigal Hoffner. Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *Bulletin of the Technical Committee on Data Engineering*, 24(1):52–57, 2001.

[16] Vivian C.T. Hermans. A hybrid process modelingapproach. Master's thesis, Eindhoven University of Technology, the Netherlands, 2014.

[17] Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Post-proceedings of PLACES 2010*, 2010.

[18] Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, pages 122–138, 1998.

[19] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284, 2008.

[20] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piwadee Noi Sukaviriya, and Roman Vaculin. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proc. of DEBS '11*, pages 51–62, New York, NY, USA, 2011. ACM.

[21] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of WS-FM'10*, pages 1–24, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] K. Jensen and L.M. Kristensen. *Coloured Petri Nets – Modelling and Validation of Concurrent Systems.* Springer, 2009.

[23] Kurt Jensen. *Coloured petri nets*. Springer, 1987.

[24] Ekkart Kindler, Wolfgang Reisig, Hagen Vlzer, and Rolf Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Computing*, 9, 1996.

[25] Marjanca Koetsier, Paul Grefen, and Jochem Vonk. Contracts for cross-organizational workflow management. In Kurt Bauknecht, SanjayKumar Madria, and Gnther Pernul, editors, *Electronic Commerce and Web Technologies*, volume 1875 of *Lecture Notes in Computer Science*, pages 110–121. Springer Berlin Heidelberg, 2000.

[26] E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *Inductive Logic Programming*, volume 4894. 2008.

[27] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In Jos Manuel Colom and Jrg Desel, editors, *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2013.

[28] F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative models from event Logs. In *CAiSE*, pages 270–285, 2012.

[29] F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2011.

[30] F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-guided discovery of declarative process models. In *CIDM*, pages 192–199, 2011.

[31] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Business Process Management (BPM) 2011*, volume 6896 of *Lecture Notes in Computer Science*, pages 32–147, 2011.

[32] Marco Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[33] Marco Montali, Maja Pesic, Wil MP van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative specification and verification of service choreographiess. *ACM Transactions on the Web (TWEB)*, 4(1):3, 2010.

[34] Raghava Rao Mukkamala. *A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, March 2012.

[35] R.R. Mukkamala and T.T. Hildebrandt. From dynamic condition response structures to büchi automata. In *Theoretical Aspects of Software Engineering (TASE), 2010 4th IEEE International Symposium on*, pages 187 –190, aug. 2010.

[36] N. A. Mulyar, M. H. Schonenberg, Mans, and van der Aalst. Towards a Taxonomy of Process Flexibility (Extended Version). 2007.

[37] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In Gilles Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 266–284. Springer Berlin / Heidelberg, 1979. 10.1007/BFb0022474.

[38] Object Management Group. Case Management Model and Notation, version 1.0. Webpage, may 2014. `http://www.omg.org/spec/CMMN/1.0/PDF`.

[39] Object Management Group BPMN Technical Committee. Business Process Model and Notation, version 2.0. Webpage, january 2011. `http://www.omg.org/spec/BPMN/2.0/PDF`.

[40] M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, pages 287–. IEEE Computer Society, Washington, DC, USA, 2007.

[41] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Universitt Hamburg, 1962.

[42] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. From declarative processes to imperative models. In *Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), Milan, Italy, November 19-21, 2014.*, pages 162–173, 2014.

[43] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. Cpn tools for editing, simulating, and analysing coloured petri nets. In *Applications and Theory of Petri Nets 2003*, pages 450–462. Springer, 2003.

[44] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies.* Springer, Berlin-Heidelberg, 2012.

[45] Shazia Sadiq, Wasim Sadiq, and Maria Orlowska. Pockets of flexibility in workflow specification. In Hideko S.Kunii, Sushil Jajodia, and Arne Slvberg, editors, *Conceptual Modeling ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 513–526. Springer Berlin Heidelberg, 2001.

[46] Keith D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press, 2010.

[47] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[48] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, November 2003.

[49] Wil van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian Gunther, Antonella Guzzo, Paul Harmon, Arthur ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Perez, Ricardo Seguel Perez, Marcos Sepulveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Wynn. Process mining manifesto. In F Daniel, K Barkaoui, and S Dustdar, editors, *Lecture Notes in Business Information Processing*, volume 99, pages 169–194. Springer, 2012.

[50] Wil van der Aalst, Maja Pesic, Helen Schonenberg, Michael Westergaard, and Fabrizio M. Maggi. Declare. Webpage, 2010. `http://www.win.tue.nl/declare/`.

[51] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, September 2004.

[52] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[53] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[54] Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[55] Wil M.P van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In M. Bravetti, M. Nunez, and Gianluigi Zavattaro, editors, *Proceedings of Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *LNCS*, pages 1–23. Springer Verlag, 2006.

[56] W.M.P. van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg. Flexibility as a service. In Lei Chen, Chengfei Liu, Qing Liu, and Ke Deng, editors, *Database Systems for Advanced Applications*, volume 5667 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2009.

[57] Wil van van der Aalst and Christian Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach.* The MIT Press, 2011.

[58] W Vanderaalst, M Weske, and D Grunbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005.

[59] Daniele Varacca, Hagen Vlzer, and Glynn Winskel. Probabilistic event structures and domains. *Theoretical Computer Science*, 358(23):173 – 199, 2006. Concurrency Theory (CONCUR 2004) 15th International Conference on Concurrency Theory 2004.

[60] Vasco Vasconcelos. Fundamentals of session types. *I&C*, 217:52–70, 2012.

[61] Hagen Vlzer. An overview of bpmn 2.0 and its potential use. In Jan Mendling, Matthias Weidlich, and Mathias Weske, editors, *Business Process Modeling Notation*, volume 67 of *Lecture Notes in Business Information Processing*, pages 14–15. Springer Berlin Heidelberg, 2010.

[62] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, September 2008.

[63] Hans Weigand and Willem-Jan van den Heuvel. Cross-organizational workflow integration using contracts. *Decis. Support Syst.*, 33(3):247–265, July 2002.

[64] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[65] Glynn Winskel. *Events in Computation.* PhD thesis, Edinburgh University, 1980.

[66] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz (Eds.) Rozenberg, editors, *Advances in Petri Nets*, volume Vol. 255 of *Lecture Notes in Computer Science,*, pages 325–392. Springer, 1986.

[67] Glynn Winskel. Events, causality and symmetry. *The Computer Journal*, 54(1):42–57, 2011.

[68] Nobuko Yoshida and Vasco Thudichum Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *ENTCS*, 171(4):73–93, 2007.

[69] Stefan Zugal, Pnina Soffer, Cornelia Haisjackl, Jakob Pinggera, Manfred Reichert, and Barbara Weber. Investigating expressiveness and understandability of hierarchy in declarative business process models. *Software & Systems Modeling*, June 2014.

[70] Stefan Zugal, Pnina Soffer, Jakob Pinggera, and Barbara Weber. Expressiveness and understandability considerations of hierarchy in declarative business process models. In *BMMDS/EMMSAD*, volume 113 of *Lecture Notes in Business Information Processing*, pages 167–181. Springer, 2012.

4 Hierarchical Dynamic Condition Response Graphs

Nested Dynamic Condition Response Graphs

Thomas Hildebrandt[1], Raghava Rao Mukkamala[1], and Tijs Slaats[1]*

IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark,
{hilde, rao, tslaats}@itu.dk,

Abstract. We present an extension of the recently introduced declarative process model *Dynamic Condition Response Graphs* (DCR Graphs) to allow *nested subgraphs* and a new *milestone relation* between events. The extension was developed during a case study carried out jointly with our industrial partner Exformatics, a danish provider of case and workflow management systems. We formalize the semantics by giving first a map from Nested to (flat) DCR Graphs with milestones, and then extending the previously given mapping from DCR Graphs to Büchi-automata to include the milestone relation.

1 Introduction

Declarative process models have been suggested by several research groups [1–5,15,16,18,19] as a good approach to describe case management and other non-rigid business and workflow processes where it is generally allowed to redo or skip activities, and even dynamically adapt the set of activities and constraints. The rationale is that if a strict sequencing is the exception, then the implicit specification of control flow in declarative models is more appropriate than notations based on explicit control flows such as the (typical use of) Business Process Model and Notation (BPMN) 2.0 [13].

A drawback of the declarative approaches in general, however, is that the implicit definition of the state and control flow makes it more complex to perceive the state and execute the process. To find out what are the next possible activities it is necessary to evaluate a set of constraints defined relatively to the history of the execution.

This motivates finding an expressive declarative process language that allows for a simple run-time scheduling which is easily visualized for the case worker. As a candidate for such a language we recently introduced in [7, 11] a declarative process model called Dynamic Condition Response Graphs (DCR Graphs). The model is a generalization of the classic event structure model for concurrency [20] and is inspired by the Process Matrix model [10,12] developed by one of our industrial partners Resultmaker, a Danish provider of workflow and case-management systems.

The core DCR Graphs model consists of a set of events and four binary relations between the events: The *dynamic inclusion* and *dynamic exclusion* relations, and the *condition* and *response* relations. The dynamic inclusion and exclusion relations generalize the usual symmetric conflict relation of event structures by splitting it in two

* This research is supported by the Danish Research Agency through a Knowledge Voucher granted to Exformatics (grant #10-087067, www.exformatics.com), the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu) and the Computer Supported Mobile Adaptive Business Processes project (grant #274-06-0415, www.cosmobiz.dk).

asymmetric relations: If an event A excludes an event B, written $A \rightarrow\!\% \ B$, then B can not happen until after the occurrence of an event C that includes event B, which is written $C \rightarrow\!+ \ B$. Similarly, the condition and response relations generalize the usual causal order relation of event structures by splitting it in two relations: If an event B has event A as condition, written $A \rightarrow\!\bullet \ B$, then event A must either be currently excluded or have happened for B to happen. Dually, if an event A has event B as response, written $A \bullet\!\!\rightarrow B$, then event B must eventually happen or always eventually be excluded after an occurrence of event A. To express that events are executed by actors with different roles the core model is extended with roles assigned to the events.

In [7] we show that the run-time state of DCR Graphs can be represented as a marking consisting of three sets of events, recording respectively the *executed events*, the *currently included events*, and the *pending response events*, i.e. events that must eventually happen or be excluded. From the marking, it is easy to evaluate if an event can happen (by checking if all its conditions are either executed or excluded) and to verify if the graph is in a completed state (by checking if the set of included pending responses is empty). It is also easy to update the state when executing an event by adding it to the set of executed events, remove the event from the pending response set and add new response events according to the response relation, and include/exclude events in the set of currently included events according to the include/exclude relations. In [7, 11] we express the acceptance condition for infinite runs (no pending response is continuously included without being executed) by giving a map to a Büchi automaton.

In the present paper we describe how to extend the model to allow for *nested sub-graphs* as is standard in most state-of-the art modelling notations. The work was carried out during a case study, in which we are applied *Nested DCR Graphs* in the design phase of the development of a distributed, inter-organizational case management system carried out by our industrial partner, Exformatics, a company that specializes in solutions for knowledge sharing, workflows and document handling.

Fig. 1 shows the graphical notation for nested DCR graphs and illustrates the use of nested sub-graphs in a sub part of the model arising from our case study. The Arrange meeting event represents the arrangement of a meeting be-

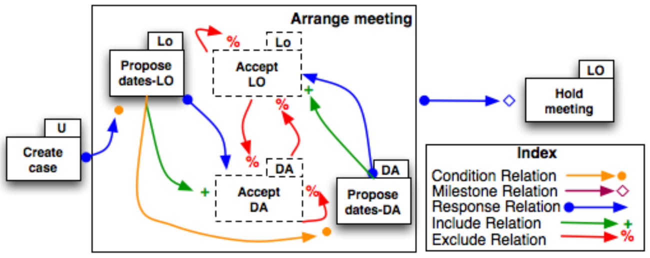


Fig. 1. Nested DCR Graphs with Arrange meeting sub-graph

tween two of the organizations (DA and LO) using the distributed case management system being developed. It has been refined to a sub-graph including four sub events for proposing and accepting dates for the meeting. The dashed boxes indicate that the events Accept DA and Accept LO for accepting meeting dates are initially excluded. Described briefly, when the organization (U) creates a case, it triggers as a response the event Propose dates-LO, representing LO proposing dates for a meeting. This event triggers as a response and includes the event Accept DA, representing DA accepting the dates. But it also enables that DA can propose other dates, represented by the event Propose Dates-DA. Now, this event triggers as a response and includes the event Accept LO, representing LO accepting the dates. Again, LO may do this, or again propose dates. The proposal of dates may continue forever, and as long as no one accepts there will be a pending response on at least one of the accept events. As soon as one of the accept events happen, they will both be excluded, and there will be none of the included events in the sub-graph having pending responses. This corresponds to the accepting condition for finite runs of DCR graphs [7], and thus intuitively reflects that the sub-graph is in a completed state. Now, we want to express that the event Hold meeting can only be executed when this is the case. To do this, we introduced a new core relation between events called the *milestone* relation. If an event A is a milestone for an event B, written $A \rightarrow\!\diamond\, B$, then B can not happen if A is included and required to be executed again (i.e. as a response). The new milestone relation allow us to define nesting as simply a tree structure on events that can be flattened to (flat) DCR Graphs by keeping all atomic events (i.e. events with no sub-events) and letting them inherit the relations defined for their super-events. In particular, the flattening does not introduce new events (in fact it removes all super events) and at most introduce an order of n^2 new relations. Thus, we need not define a new operational semantics for nested DCR Graphs, instead we can make the much simpler extension of the semantics for (flat) DCR Graphs to consider the new milestone relation. It is worth noting that while the milestone relation makes it very direct to express completion of subgraphs, we conjecture that it does not add expressiveness to DCR Graphs.

Related work: Our approach is closely related to the work on ConDec [18, 19]. The crucial difference is that we allow nesting and a few core constraints making it possible to describe the state of a process as a simple marking. ConDec does not address nesting (nor dynamic inclusion/exclusion), but allows one to specify any relation expressible within Linear-time Temporal Logic (LTL). This offers much flexibility with respect to specifying execution constraints. In particular the condition and response relations are standard verification patterns and also considered in [18, 19] (the condition relation is called precedence), and we have used the same graphical notation. However, the execution of a process expressed as LTL (which typically involves a translation to a Büchi-automaton) is more complex and the run-time state is difficult to relate to the original ConDec specification. Moreover, we conjecture that DCR Graphs are as expressive as Büchi-automata, and thus more expressive than LTL. Finally, Nested DCR Graphs relates to the independent (so far unpublished) work on the declarative *Guard-Stage-Milestone* model by Hull, presented in invited talks at WS-FM 2010 and CASCON 2010.

Structure of paper: In Sec. 2 we define Nested DCR Graphs formally, motivated by the case study, and define the mapping to flat DCR Graphs with milestones. In Sec. 3 we then define the lts semantics and the mapping from flat DCR Graphs with milestones to Büchi-automata. The two maps together define the semantics of Nested DCR Graphs. Due to space limitations we refer to [8] and the full version [9] for a detailed description of the case study and tool support. We conclude in Sec. 4 and give pointers to future work.

2 Nested DCR Graphs and Milestones

We now give the formal definition of the Nested DCR Graph model described informally above, which extends the model in our previous work [7] with nesting and the new milestone relation $\rightarrow\!\diamond$ between events.

Definition 1. *A* Nested Distributed dynamic condition response graph with milestones *is a tuple* $(\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$*, where*

(i) E *is the set of* events
(ii) $\rhd : \mathsf{E} \rightharpoonup \mathsf{E}$ *is a partial function mapping an event to its super-event (if defined), and we also write* $e \rhd e'$ *if* $e' = \rhd^k(e)$ *for* $0 < k$*, referred to as the* nesting relation
(iii) $\mathsf{M} = (E, R, I) \subseteq \mathsf{atoms}(\mathsf{E}) \times \mathsf{atoms}(\mathsf{E}) \times \mathsf{atoms}(\mathsf{E})$ *is the* marking*, containing sets of currently executed events (E), currently pending responses (R), and currently included events (I).*
(iv) $\rightarrow\!\bullet \subseteq \mathsf{E} \times \mathsf{E}$ *is the* condition *relation*
(v) $\bullet\!\rightarrow \subseteq \mathsf{E} \times \mathsf{E}$ *is the* response *relation*
(vi) $\rightarrow\!\diamond \subseteq \mathsf{E} \times \mathsf{E}$ *is the* milestone *relation*
(vii) $\pm : \mathsf{E} \times \mathsf{E} \rightharpoonup \{+, \%\}$ *is a partial function defining the* dynamic inclusion and exclusion *relations by* $e \rightarrow\!+\, e'$ *if* $\pm(e, e') = +$ *and* $e \rightarrow\!\%\, e'$ *if* $\pm(e, e') = \%$
(viii) Act *is the set of* actions
(ix) $l : \mathsf{E} \rightarrow \mathsf{Act}$ *is a* labeling *function mapping events to actions.*
(x) R *is a set of* roles*,*
(xi) P *is a set of* principals *(e.g. persons or processors) and*
(xii) $\mathsf{as} \subseteq (\mathsf{P} \cup \mathsf{Act}) \times \mathsf{R}$ *is the* role assignment *relation to principals and actions.*

where $\mathsf{atoms}(\mathsf{E}) = \{e \mid \forall e' \in \mathsf{E}.\ \rhd(e') \neq e\}$ *is the set of* atomic *events.*

We require that the nesting relation $\rhd \subset E \times E$ *is acyclic and that there are no infinite sequence of events* $e_1 \rhd e_2 \rhd \ldots$*. We will write* $e \trianglerighteq e'$ *if* $e \rhd e'$ *or* $e = e'$*, and* $e \trianglelefteq e'$ *if* $e' \rhd e$ *or* $e = e'$*. We require that the nesting relation is consistent with respect to dynamic inclusion/exclusion in the following sense: If* $e \rhd e'$ *or* $e' \rhd e$ *then* $\pm(e, e'') = +$ *implies* $\pm(e', e'') \neq \%$ *and* $\pm(e, e'') = \%$ *implies* $\pm(e', e'') \neq +$.

The new elements are the nesting relation $\rhd \subset E \times E$ and the milestone relation $\rightarrow\!\diamond \subseteq \mathsf{E} \times \mathsf{E}$. The consistency between the nesting relation and the dynamic inclusion/exclusion is to ensure that when we map a nested DCR Graph to the corresponding flat DCR Graph as defined in Def. 2 below, no atomic event both includes and excludes another event. That is, if an event e includes (excludes) another event e'', then any of its super or sub events e' can not exclude (include) the event e''.

The new elements conservatively extend the DCR Graphs defined in [7] in the sense that given a Nested dynamic condition response graph as defined in Def. 1, the tuple $(\mathsf{atoms}(\mathsf{E}), \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$ is a (Distributed) dynamic condition response graph as defined in [7]. In particular, the semantics will be identical if both the $\rhd$ map and the milestone relation are empty.

A nested distributed dynamic condition response graph can be mapped to a flat distributed dynamic condition response graph with at most the same number of events. Essentially, all relations are extended to sub events, and then only the atomic events are preserved. The labelling function is extended by labelling an atomic event e by the sequence of labels labelling the chain of super events starting by the event itself: $e \rhd e_1 \ldots e_k \not\rhd$. The role assignment is extended to sequences of actions by taking the union of roles assigned to the actions.

Definition 2. *For a Nested DCR Graph* $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$ *define the underlying flat DCR Graph as*

$$G^\flat = (\mathsf{atoms}(\mathsf{E}), \mathsf{M}, \rightarrow\!\bullet^\flat, \bullet\!\rightarrow^\flat, \rightarrow\!\diamond^\flat, \pm^\flat, \mathsf{Act}^+, l^\flat, \mathsf{R}, \mathsf{P}, \mathsf{as}^\flat),$$

where $rel^\flat = \trianglerighteq rel \trianglelefteq$ *for some relation* $rel \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond\}$ *and* $\pm(e, e')^\flat = \pm(e_s, e'_s)$ *if* $\pm(e_s, e'_s)$ *is defined and* $e \trianglerighteq e_s$ *and* $e' \trianglelefteq e'_s$ *and* $l^\flat(e_0) = a_0.a_1.a_2 \ldots a_k$ *if* $e_0 \rhd e_1 \rhd e_2 \ldots \rhd e_k$ *and* $l(e_i) = a_i$ *for* $0 \leq i \leq k$ *and* $\mathsf{as}^\flat(a_0.a_1.a_2 \ldots a_k) = \{\mathsf{as}(a_i) \mid 0 \leq i \leq k\}$ *and* $as^\flat(p) = p$ *for* $p \in \mathsf{P}$.

It is easy to see that the size of the relations may increased by an order of at most n^2 where n is the number of atomic events.

3 Semantics

Below we define the semantics of DCR Graphs with milestones by giving a labelled transition semantics and a mapping to Büchi-automata.
Notation: For a set A we write $\mathcal{P}(A)$ for the power set of A. For a binary relation $\rightarrow \subseteq A \times A$ and a subset $\xi \subseteq A$ of A we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{a \in A \mid (\exists a' \in \xi \mid a \rightarrow a')\}$ and the set $\{a \in A \mid (\exists a' \in \xi \mid a' \rightarrow a)\}$ respectively.

Definition 3. *For a dynamic condition response graph with milestones* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, l, \mathsf{Act}, \mathsf{R}, \mathsf{P}, \mathsf{as})$*, we define the corresponding labelled transition systems* $T(G)$ *to be the tuple* $(S, \mathsf{M}, \rightarrow \subseteq S \times \mathsf{Act} \times S)$ *where* $S = \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$ *is the set of markings of* G *and* $\mathsf{M} = (R, I, E) \in S$ *is the initial marking,* $\rightarrow \subseteq S \times \mathsf{E} \times (\mathsf{P} \times \mathsf{Act} \times \mathsf{R}) \times S$ *is the transition relation given by* $\mathsf{M}' \xrightarrow{(e,(p,a,r))} \mathsf{M}''$ *where*

(i) $\mathsf{M}' = (E', R', I')$ *is the marking before transition*
(ii) $\mathsf{M}'' = (E' \cup \{e\}, R'', I'')$ *is the marking after transition*
(iii) $e \in I$, $l(e) = a$, $p \,\mathsf{as}\, r$, *and* $a \,\mathsf{as}\, r$,
(iv) $\rightarrow\!\bullet e \cap I' \subseteq E'$,
(v) $\rightarrow\!\diamond e \cap I' \cap R' = \emptyset$,
(vi) $I'' = (I' \cup e\!\rightarrow\!+) \setminus e\!\rightarrow\!\%$,
(vii) $R'' = (R' \setminus \{e\}) \cup e\!\bullet\!\rightarrow$,
(viii) $E'' = E' \cup \{e\}$

We define a run $(e_0, (p_0, a_0, r_0)), (e_1, (p_1, a_1, r_1)), \ldots$ *of the transition system to be a sequence of labels of a sequence of transitions* $\mathsf{M}_i \xrightarrow{(e_i,(p_i,a_i,r_i))} \mathsf{M}_{i+1}$ *starting from the initial marking. We define a run to be accepting if* $\forall i \geq 0, e \in R_i.\exists j \geq i.(e = e_j \vee e \notin I_j)$*. In words, a run is accepting if no response event is included and pending forever, i.e. it must either happen at some later state or become excluded.*

Condition *(iii)* in the above definition expresses that, only events e that are currently included, can be executed, and to give the label (p, a, r) the label of the event must be a, p must be assigned to the role r, which must be assigned to a. Condition *(iv)* requires that all condition events to e which are currently included should have been executed previously. Condition *(v)* states that the currently included events which are milestones to event e must not be in the set of pending responses (R'). Conditions *(vi)*, *(vii)* and *(viii)* are the updates to the sets of included events, pending responses and executed events respectively. Note that an event e' can not be both included and excluded by the same event e, but an event may trigger itself as a response.

If one considers only finite runs then the acceptance condition degenerates to requiring that no pending response is included at the end of the run. If infinite runs are also of interest (as e.g. for reactive systems and LTL) the acceptance condition can be captured by a mapping to a Büchi-automaton with τ-event defined as follows.

Definition 4. *A* Büchi-automaton with τ-event *is a tuple* $(S, s, Ev_\tau, \rightarrow \subseteq S \times Ev_\tau \times S, F)$ *where* S *is the set of* states, $s \in S$ *is the* initial state, Ev_τ *is the set of* events *containing the special event* τ, $\rightarrow \subseteq S \times Ev_\tau \times S$ *is the* transition relation, *and* F *is the set of* accepting states. *A (finite or infinite) run is a sequence of labels not containing the* τ *event that can be obtained by removing all* τ *events from a sequence of labels of transitions starting from the initial state. The run is* accepting *if the sequence of transitions passes through an accepting state infinitely often.*

Since we at any given time may have several pending responses we must make sure in the mapping to Büchi-automata that all of them are eventually executed or excluded. To do this we assume any fixed order of the finite set of events E of the given dynamic condition response graph. For an event $e \in E$ we write $rank(e)$ for its rank in that order and for a subset of events $E' \subseteq E$ we write $min(E')$ for the event in E' with the minimal rank.

Definition 5. For a finite distributed dynamic condition response graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$ where $\mathsf{E} = \{e_1, \ldots, e_n\}$, marking $\mathsf{M} = (E, R, I)$ and $rank(e_i) = i$, we define the corresponding Büchi-automaton with τ-event to be the tuple $B(G) = (S, s, \rightarrow \subseteq S \times Ev_\tau \times S, F)$ where

- $S = \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \{1, \ldots, n\} \times \{0, 1\}$ is the set of states,
- $Ev_\tau = (\mathsf{E} \times (\mathsf{P} \times \mathsf{Act} \times \mathsf{R})) \cup \{\tau\}$ is the set of events,
- $s = (\mathsf{M}, 1, 1)$ if $I \cap R = \emptyset$, and $s = (\mathsf{M}, 1, 0)$ otherwise
- $F = \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \{1, \ldots, n\} \times \{1\}$ is the set of accepting states and
- $\rightarrow \subseteq S \times Ev_\tau \times S$ is the transition relation given by $(\mathsf{M}', i, j) \xrightarrow{\tau} (\mathsf{M}', i, j')$ where

(a) $\mathsf{M}' = (E', R', I')$ is the marking
(b) $j' = 1$ if $I' \cap R' = \emptyset$ otherwise $j' = 0$.

$$\text{and } (\mathsf{M}', i, j) \xrightarrow{\text{(e,(p,a,r))}} (\mathsf{M}'', i', j') \text{ where}$$

(i) $\mathsf{M}' = (E', R', I') \xrightarrow{\text{(e,(p,a,r))}} (E'', R'', I'') = \mathsf{M}''$ is a transition of $T(D)$.

(ii) For $M = \{e \in I' \cap R' \mid rank(e) \rangle i\}$ let $j' = 1$ if
   (a) $I'' \cap R'' = \emptyset$ or
   (b) $min(M) \in (I' \cap R' \backslash (I'' \cap R'')) \cup \{e\}$ or
   (c) $M = \emptyset$ and $min(I' \cap R') \in (I' \cap R' \backslash (I'' \cap R'')) \cup \{e\}$
   otherwise $j' = 0$.
(iii) $i' = rank(min(M))$ if $min(M) \in (I' \cap R' \backslash (I'' \cap R'')) \cup \{e\}$ or else
(iv) $i' = rank(min(I' \cap R'))$ if $M = \emptyset$ and $min(I' \cap R') \in (I' \cap R' \backslash (I'' \cap R'')) \cup \{e\}$ or else
(v) $i' = i$ otherwise.

We prove that the mapping from the labelled transition semantics to Büchi-automata is sound and complete in the full version of the paper [9].

The formal semantics of DCR graphs mapped to Büchi-automata enabled us to perform model checking and formal verification of processes specified in DCR graphs. The prototype implementation allows us to perform verification of both safety and liveness properties using the SPIN [17] model checker and only verification of safety properties using the ZING [14] model checker. The prototype has also been extended to support runtime verification, for monitoring of properties specified using Property Patterns [6].

4 Conclusion and Future Work

We have given a conservative extension of the declarative process model Distributed DCR Graphs [7] to allow for nested sub-graphs motivated and guided by a case study carried out jointly with our industrial partner. A detailed description of the case study and tool support for DCR Graphs can be found in [8]. The main technical challenge was to formalize the execution and in particular *completion* of sub-graphs. We do this by introducing a new *milestone* relation $A \rightarrow\!\diamond B$, which blocks the event B as long as there are events in A required to be executed (i.e. required responses). We believe this is the right notion of completeness of nested sub-graphs. First of all, it coincides with the definition of acceptance of finite runs in DCR Graphs [7] recalled in Sec. 3 above. Secondly, its formalization is a simple extension of the labelled transition semantics given in [7, 11] since it is a condition on the set of pending responses already included in the states. Finally, it allows for a nested sub-graph to alternate between being completed and not completed, as is often the case in ad hoc case management. This is not possible in the related ad-hoc sub-process activity in BPMN 2.0. Future work within the Trust-Care and CosmoBiz projects, which are the context of the work, includes exploring the expressiveness of DCR Graphs, extending the theory and tools for analysis, verification and model-driven engineering, extending the model to be able to express other relevant features such as multi-instance sub-graphs, time, exceptions, data, types and run-time adaption, i.e. dynamic changes of the model.

References

1. Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *LNCS*, volume 4714, pages 288–304, 2007.

2. Christoph Bussler and Stefan Jablonski. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*, pages 53–59, Feb 1994.
3. David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
4. Hasam Davulcu, Michael Kifer, C. R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of ACM SIGACT-SIGMOD-SIGART*, pages 1–3. ACM Press, 1998.
5. Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 252–267, New York, NY, USA, 2009. ACM.
6. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the second workshop on Formal methods in software practice*, FMSP '98, pages 7–15, New York, NY, USA, 1998. ACM.
7. Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Programming Language Approaches to Concurrency and communication-cEntric Software 2010 (PLACES10)*. EPTCS, 2010.
8. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Accepted for IEEE International EDOC Conference*, 2011.
9. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using nested dynamic condition response graphs. Technical Report TR-2011-141, IT University of Copenhagen, 2011.
10. Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Proceedings ProHealth 08 workshop*, 2008.
11. Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010.
12. Raghava Rao Mukkamala, Thomas Hildebrandt, and Janus Boris Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008.
13. Object Management Group BPMN Technical Committee. Business Process Model and Notation, version 2.0, 2010. http://www.omg.org/cgi-bin/doc?dtc/10-06-04.pdf.
14. Microsoft Research. Zing model checker. Webpage, 2010. http://research.microsoft.com/en-us/projects/zing/.
15. Pinar Senkul, Michael Kifer, and Ismail H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *VLDB*, pages 694–705, 2002.
16. Munindar P. Singh, Greg Meredith, Christine Tomlinson, and Paul C. Attie. An event algebra for specifying and scheduling workflows. In *Proceedings of DASFAA*, pages 53–60. World Scientific Press, 1995.
17. Spin. On-the-fly, ltl model checking with spin. Webpage, 2008. http://spinroot.com/spin/whatispin.html.
18. Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
19. Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings of DPM 2006*, LNCS. Springer Verlag, 2006.
20. Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.

Designing a Cross-organizational Case Management System using Dynamic Condition Response Graphs

Thomas Hildebrandt
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
hilde@itu.dk

Raghava Rao Mukkamala
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
rao@itu.dk

Tijs Slaats
IT University of Copenhagen and
Exformatics A/S
2100 Copenhagen, Denmark
tslaats@itu.dk

***Abstract*—We present a case study of the use of Dynamic Condition Response (DCR) Graphs, a recently introduced declarative business process model, in the design of a cross-organizational case management system being developed by Exformatics A/S, a Danish provider of knowledge and workflow management systems. We show how DCR Graphs allow to capture directly both the behavioral constraints identified during meetings with the customer and the operational execution as markings of the graph. In comparison, imperative models such as BPMN, Petri Net, UML Sequence or Activity diagrams are only good at describing the operational way to fulfill the constraints, leaving the constraints implicit. In particular, we point out that the BPMN ad-hoc sub process activity, intended to support more loosely structured goal driven ad-hoc processes, is inconsistently described in the final version of the BPMN 2.0 standard. The case study motivated an extension of the DCR Graphs model to nested graphs and the development of graphical design and simulation tools to increase the understanding of the models. The study also revealed a number of challenges for future research in techniques for model-driven design of cross-organizational process-aware information systems combining declarative and imperative models.**

***Index Terms*—Case Study, Declarative Workflow, Model-driven Design**

I. Introduction

The purpose of a Case Management System as used in for instance Human Resource (HR) departments, hospitals, financial, and governmental institutions, is to guide case workers to perform the right tasks and to record the history of the case.

Since the initial work on office automation and workflow systems [9], [10], [32] it has been advocated to base the implementation of such systems, subsequently referred to as process-aware information systems [8], on explicit process descriptions described in some high-level process notation such as Petri Net or UML activity diagrams. The key motivations for using explicit process models are to allow the system to be more easily adapted to different work processes and to make the rules governing the system more visible to the users.

Authors listed alphabetically. This research is supported by the Danish Research Agency through a Knowledge Voucher granted to Exformatics (grant #10-087067, www.exformatics.com), the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu) and the Computer Supported Mobile Adaptive Business Processes project (grant #274-06-0415, www.cosmobiz.dk).

The rise of web service standards such as SOAP, WSDL and WS-BPEL has given new momentum to process-aware information systems. SOAP and WSDL standardize how to access external IT systems as web services in a service oriented architecture and WS-BPEL provides a standard high-level programming language for combining individual service calls into process flows, also referred to as a process orchestration. Following WS-BPEL, the BPEL4People [1] and WS-HumanTask [20] specifications were the first attempt to standardize the inclusion of human tasks into BPEL to encompass workflows. Moreover, W3C started in 2004 developing the Web Services Choreography Description Language (WS-CDL) [29] which can be used to provide a global view of the intended interactions between different actors of a system, similar to the view of interactions provided by UML sequence diagrams. Within the last 5 years focus has moved from WS-BPEL and BPEL4People to the development of Business Process Model and Notation (BPMN) [19] which standardizes the graphical notation used for business processes, encompassing both human and automated tasks, and including both notations for orchestrations and choreographies.

However, as pointed out in e.g. [10], [27], the imperative process notations with explicit control and message flows underlying all of the above models describe the operationalization of business process goals and constraints, and not the goals and constraints themselves. Consequently, the notations are best suited for well-defined, rigid and repeatable workflows following a predefined sequence of service invocations and human tasks and one need to use ad hoc annotations to record the constraints and goals of the process. Moreover, it has proven to be non-trivial to support changes of the processes on-the-fly [26]. This does not match well the typical more ad-hoc nature of case work where it is often needed to redo and skip tasks and possibly adapt the set of tasks and their mutual constraints dynamically [25].

An alternative approach studied by several research groups is the use of declarative process models [3], [6], [22], [23], [27], [28], which describes the temporal constraints on process flows, not how to fulfill them. As part of the PhD project of the second author within the Trustworthy Pervasive Healthcare Services (TrustCare) research project [11] we have developed a declarative process model called Dynamic Condition Response Graphs (DCR Graphs) [12]–[14], [17]. The model is both a generalization of the Process Matrix model [16], [18] developed by Resultmaker, a danish provider of workflow and case-management systems and the classical event structure model for concurrency [30], [31]. DCR Graphs relate to DECLARE [28], which is a graphical notation that allows any temporal constraint pattern expressible as Linear-time Temporal Logic (LTL) formulas. However, instead of allowing the generality of expressing any constraint expressible in LTL, DCR Graphs only has a fixed handful of constraints which can be understood without reference to LTL. Still it maintains the full expressive power of LTL (described in a follow up paper). Moreover, it is possible to give an operational semantics expressed directly as transitions between a novel type of markings of the tasks. In this way DCR Graphs combine the declarative view (constraints between tasks) with the imperative view (markings of tasks) allowing to trace the constraints even at run-time.

In the present paper we first briefly review the definition of DCR Graphs in Sec. II and then in Sec. III describe a case study of applying DCR graphs in the design phase of the development of a cross-organizational case management system. In Sec. IV we briefly describe the current status of our development of tools for supporting design, simulation and verification of DCR Graphs. Finally, in Sec. VI we outline challenges identified in the case study and the proposal for the continued development of the DCR Graphs model, technologies and tools to make them applicable to component and model based design of distributed process-aware information systems.

II. Dynamic Condition Response Graphs

A Dynamic Condition Response Graph as introduced in [13] and extended in [14] consists of a set of events, a *marking* defining the execution state, and five binary relations between the events defining the conditions for the execution of events, the required responses and a novel notion of dynamic inclusion and exclusion of events. Hereto comes a set of actions, a labeling function assigning an action to each event, a set of roles, a set of principals and a relation assigning roles to actions and principals.

Notation: For a set A we write $\mathcal{P}(A)$ for the power set of A. For a binary relation $\rightarrow \subseteq A \times A$ and a subset $\xi \subseteq A$ of A we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{a \in A \mid (\exists a' \in \xi \mid a \rightarrow a')\}$ and the set $\{a \in A \mid (\exists a' \in \xi \mid a' \rightarrow a)\}$ respectively.

Formally we define a DCR Graph as follows.

Definition 1: A Dynamic Condition Response Graph is a tuple $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$, where

(i) E is the set of *events*
(ii) $\mathsf{M} \in \mathcal{M}(G)$ is the *marking*, and $\mathcal{M}(G) =_{def} \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$
(iii) $\rightarrow\!\bullet \subseteq \mathsf{E} \times \mathsf{E}$ is the *condition* relation
(iv) $\bullet\!\rightarrow \subseteq \mathsf{E} \times \mathsf{E}$ is the *response* relation
(v) $\rightarrow\!\diamond \subseteq \mathsf{E} \times \mathsf{E}$ is the *milestone* relation
(vi) $\pm : \mathsf{E} \times \mathsf{E} \rightharpoonup \{+, \%\}$ is a partial function defining the *dynamic inclusion and exclusion* relations by $e \rightarrow\!+\, e'$ if $\pm(e, e') = +$ and $e \rightarrow\!\%\, e'$ if $\pm(e, e') = \%$
(vii) Act is the set of *actions*
(viii) $l : \mathsf{E} \rightarrow \mathsf{Act}$ is a *labeling* function mapping events to actions.
(ix) R is a set of *roles*,
(x) P is a set of *principals* (e.g. actors, persons, processors, services) and
(xi) $\mathsf{as} \subseteq (\mathsf{P} \cup \mathsf{Act}) \times \mathsf{R}$ is the *role assignment* relation to principals and actions.

An event labelled with an action, e.g. Create Case, thus represents an execution of a (human or automated) task/activity/action Create Case in the workflow process. There may be several events with the same label, but in all our examples a label is assigned to a unique event, and thus we simply assume the set of events to be identical to the set of actions.

By default an event may be executed at any time and any number of times. However, the marking (defining the run-time state of the graph) and the five relations defined in (iii)-(vi) constrain the execution. The marking $\mathsf{M} = (Ex, Re, In) \in \mathcal{M}(G)$ consists of three sets of events, capturing respectively which events have *previously been executed* (Ex), which events are *pending responses required to be executed* (Re), and finally which events are currently *included* (In). Only events $e \in In$, i.e. that are currently included, can be executed, and only if all currently included condition events e', as specified by the condition relation $e' \rightarrow\!\bullet\, e$, have been executed and no currently included events e' which are milestones for e, as specified by the milestone relation $e' \rightarrow\!\diamond\, e$, are pending responses.

When an event e is executed, it is added to the set of executed events (Ex) of the marking and all response events e', as specified by the response relation $e \bullet\!\rightarrow e'$, are added to the set of pending responses Re. Moreover, the set of included events is updated by adding (removing) all events e' included (excluded) by e as specified by the inclusion (exclusion) relation $e \rightarrow\!+\, e'$ ($e \rightarrow\!\%\, e'$).

The execution semantics of DCR Graphs is defined [12], [14] as a labelled transition system between markings as follows.

Definition 2: For a DCR Graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \pm, l, \mathsf{Act}, \mathsf{R}, \mathsf{P}, \mathsf{as})$, we define the corresponding labelled transition systems $T(G)$ to be the tuple $(\mathcal{M}(G), \mathsf{M}, \rightarrow \subseteq \mathcal{M}(G) \times \mathcal{L}(G) \times \mathcal{M}(G))$ where $\mathcal{L}(G) =_{def} \mathsf{E} \times (\mathsf{P} \times \mathsf{Act} \times \mathsf{R})$ is the set of transition labels, $\mathsf{M} = (Ex, Re, In) \in \mathcal{M}(G)$ is the initial marking, $\rightarrow \subseteq \mathcal{M}(G) \times \mathcal{L}(G) \times \mathcal{M}(G)$ is the transition relation given by $\mathsf{M}' \xrightarrow{(e,(p,a,r))} \mathsf{M}''$

where

(i) $\mathsf{M}' = (Ex', Re', In')$ is the marking before transition
(ii) $\mathsf{M}'' = (Ex' \cup \{e\}, Re'', In'')$ is the marking after transition
(iii) $e \in In'$, $l(e) = a$, $p\,\mathsf{as}\,r$, and $a\,\mathsf{as}\,r$,
(iv) $(\rightarrow\!\bullet\, e \cap In') \subseteq Ex'$,
(v) $(\rightarrow\!\diamond\, e \cap In') \cap Re' = \emptyset$,

(vi) $In'' = (In' \cup e{\rightarrow}{+}) \setminus e{\rightarrow}\%$,

(vii) $Re'' = (Re' \setminus \{e\}) \cup e{\bullet}{\rightarrow}$,

We define a run $(e_0,(p_0,a_0,r_0)),(e_1,(p_1,a_1,r_1)),\ldots$ of the transition system to be a sequence of labels of a sequence of transitions $\mathsf{M}_i \xrightarrow{(e_i,(p_i,a_i,r_i))} \mathsf{M}_{i+1}$ starting from the initial marking. We define a run to be accepting if $\forall i \geq 0, e \in Re_i.\exists j \geq i.(e = e_j \vee e \notin In_{j+1})$. In words, a run is accepting if no response event is pending forever, i.e. it must either happen at some later state or become excluded.

Condition *(iii)* in the above definition expresses that, only events e that are currently included, can be executed, and to give the label (p,a,r) the label of the event must be a, p must be assigned to the role r, which must be assigned to a. Condition *(iv)* requires that all condition events to e which are currently included should have been executed previously. Condition *(v)* states that the currently included events which are milestones to event e must not be in the set of pending responses (Re'). Condition *(vi)* and *(vii)* are the updates to the sets of included events and pending responses respectively. Note that an event e' can not be both included and excluded by the same event e, but an event may trigger itself as a response.

In this paper we only consider finite runs. In this case, the acceptance condition degenerates to requiring that no pending response is included at the end of the run. This corresponds to defining all states where $Re \cap In = \emptyset$ to be accepting states and define the accepting runs to be those ending in an accepting state. Infinite runs are also of interest especially in the context of reactive systems and the LTL logic. The execution semantics and acceptance condition for infinite runs are captured by mapping to a Büchi-automaton with τ-event as formalized in [12], [17].

During the case study (Sec. III), we realized the need to extend our model with nested sub-graphs to allow for modeling of hierarchical sub structures. To address this need, so-called Nested DCR Graphs were introduced in [14]. It can be defined as an incremental extension to DCR Graph given in Def. 1 above as follows.

Definition 3: A Nested dynamic condition response graph is a tuple $(\mathsf{E}, \rhd, \mathsf{M}, {\rightarrow}{\bullet}, {\bullet}{\rightarrow}, {\rightarrow}{\diamond}, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$, where $\rhd : \mathsf{E} \rightharpoonup \mathsf{E}$ is a partial function mapping an event to its super-event (if defined) and $(\mathsf{E}, \mathsf{M}, {\rightarrow}{\bullet}, {\bullet}{\rightarrow}, {\rightarrow}{\diamond}, \pm, \mathsf{Act}, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$ is a DCR Graph, subject to the condition that the marking $\mathsf{M} = (Ex, Re, In) \subseteq \mathsf{atoms}(\mathsf{E}) \times \mathsf{atoms}(\mathsf{E}) \times \mathsf{atoms}(\mathsf{E})$ where $\mathsf{atoms}(\mathsf{E}) = \{e \mid \forall e' \in \mathsf{E}. \rhd(e') \neq e\}$ is the set of *atomic* events.

A nested DCR Graph can be mapped to a flat DCR Graph by extending all relations to the sub events and by preserving only the atomic events. This flattening of a nested DCR Graph into a DCR Graph is defined formally in [14]. In particular, the semantics of a Nested DCR Graph is given as the labelled transition semantics for its corresponding flattened DCR Graph.

III. Case Study: A Cross-Organizational Case Management System

In this section we demonstrate how we have applied DCR Graphs in practice within a project that our industrial partner Exformatics carried out for one of their customers. In the process, we acted as consultants, applying DCR Graphs in meetings with Exformatics and the customer to capture the requirements in a declarative way, accompanying the usual UML sequence diagrams and prototype mock-ups. Sequence diagrams typically only describe *examples* of runs, and even if they are extended with loops and conditional flows they do not capture the constraints explicitly.

The customer of the system is *Landsorganisationen i Danmark* (LO), which is the overarching organization for most of the trade unions in Denmark. Their counterpart is *Dansk Arbejdsgiverforening* (DA), which is an overarching organization for most of the Danish employers organizations.

At the top level, the workflow to be supported is that a case worker at the trade union must be able to create a case, e.g. triggered by a complaint by a member of the trade union against her employer. This must be followed up by a meeting arranged by LO and subsequently held between case workers at the trade union, LO and DA. After being created, the case can at any time be managed, e.g. adding or retrieving documents, by case workers at any of the organizations.

Fig. 1 shows the graphical representation of a simple DCR Graph capturing these top level requirements of our case study.

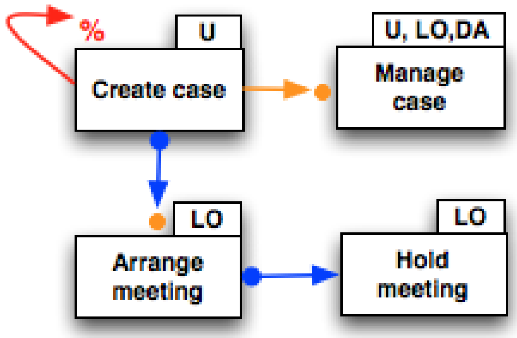


Figure 1. Top level requirements as a DCR Graph

Four top-level events were identified, shown as boxes in the graph labelled Create case, Manage case, Arrange meeting and Hold meeting. For the top-level events we identified the following requirements:

1) A case is created by a union case worker, and only once.
2) The case can be managed at the union, LO and DA after it has been created.
3) After a case is created, LO can and must arrange a meeting between the union case worker, the LO case worker and the DA case worker.
4) After a meeting is arranged it must be held (organized by LO).

The requirements translate to the following DCR Graph role assignments (shown as ”ears” on the event boxes) and relations shown as different types of arrows between the events in Fig. 1:

1) Create case has assigned role U and excludes itself.
2) Create case is a condition for Manage case, which has assigned role U, LO and DA.
3) Create case has Arrange meeting as response, which has assigned role LO.
4) Arrange meeting has Create case as a condition and Hold meeting as response, which has assigned role LO.

For example, the U on Create case indicates that only a case worker at the trade union (U) can create a case, and the U, LO, DA on Manage case indicate that both the trade union, LO and DA can manage the case.

The arrow Create case→•Manage case denotes that Manage case has Create case as a (pre) *condition*. This simply means that Create case must have happened before Manage case can happen. Dually, Arrange meeting has Hold meeting as *response*, denoted by the arrow Arrange meeting•→Hold meeting This means that Hold meeting *must* eventually happen after Arrange meeting happens. Finally, the arrow Create case →%Create case denotes that the event Create case excludes itself.

In the subsequent meetings, we came to the following additional requirements:

1) a) To create a case, the case worker should enter metadata on the case, inform about when he/she is available for participating in a meeting and then submit the case.
   b) When a case is submitted it may get a local id at the union, but it should also subsequently be assigned a case id in LO.
   c) When a case is submitted, LO should eventually propose dates.
2) a) Only after LO has assigned its case id it is possible to manage the case and for LO to propose dates.
   b) Manage case consists of three possible activities (in any order): editing case meta data, upload documents and download documents. All activities can be performed by LO and DA. Upload and download documents can also be performed by the Union.
3) a) The meeting should be arranged in agreement between LO and DA: LO should always propose dates first - and then DA should accept, but can also propose new dates. If DA proposes new dates LO should accept, but can also again propose new dates. This could in principle go on forever.
   b) The union can always update information about when they are available and edit the metadata of the case.
4) a) No meeting can be held while LO and DA are negotiating on a meeting date. Once a date has been agreed upon a meeting should eventually be held.

These requirements led to the extension of the model allowing *nested* events as recalled in the previous section and given in full detail in [14].

The requirements could then be described by first adding the following additional events to the graph: A new super event Edit (E) which has the sub events: Metadata (E-M) and Dates available (E-D) and is itself a sub event to Create case (CC). The Create case (CC) event has two sub events: Submit (SC) and Assign case Id (ACI). The Manage case (MC) event has two sub events: Edit metadata (EM) and Document (D), which in turn has two sub events: Upload (D-U) and Download (D-D). The Arrange meeting (AM) event has four sub events: Propose dates-LO (PLO), Propose dates-DA (PDA), Accept LO (ALO) and Accept DA (ADA). The Hold meeting (HM) event remains an atomic top-level event.

Subsequently, the relations was adapted to the following (Nested) DCR Graph relations, as shown in Fig 2:

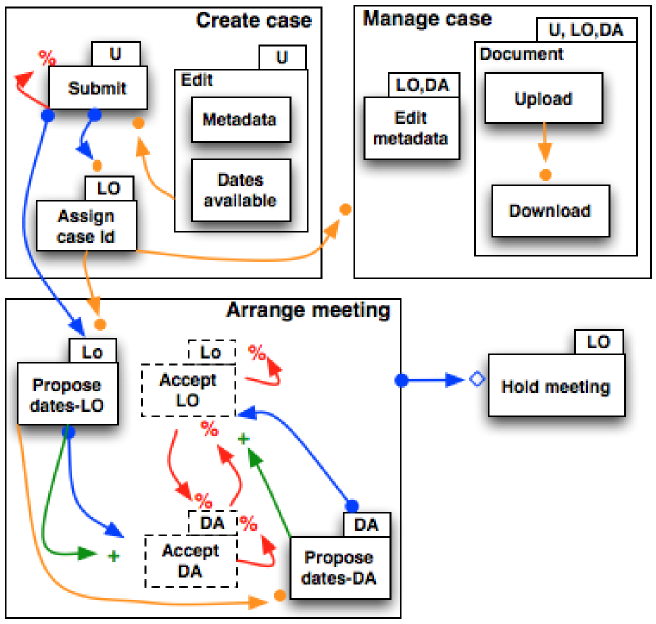


Figure 2. Case Handling Process

1) Edit is a condition to Submit and is assigned role U.
2) Within the Create case superevent:
   a) Submit is a condition to Assign case Id and also requires it as a response.
   b) Assign case Id is a condition for Manage case (and therefore also all it's sub events).
   c) Assign case Id is now the condition for Propose dates-LO and Submit requires it as a response.
3) Within the Arrange meeting superevent:
   a) Arrange meeting still has Hold meeting as response, but is now also required as a milestone for Hold meeting
   b) Propose dates-LO is a condition for Propose dates-DA
   c) Propose dates-LO includes Accept DA and requires it as a response
   d) Propose dates-DA includes Accept LO and requires it as a response
   e) Accept LO excludes itself and Accept DA
   f) Accept DA excludes itself and Accept LO
4) Within the Manage case superevent:

a) Edit metadata has roles LO and DA assigned to it.
b) Upload and Download have been grouped under a superevent Document with roles U, LO and DA assigned to it.
c) Upload is a condition for Download.

The case handling process described above and shown in figure 2 can be represented formally as follows.

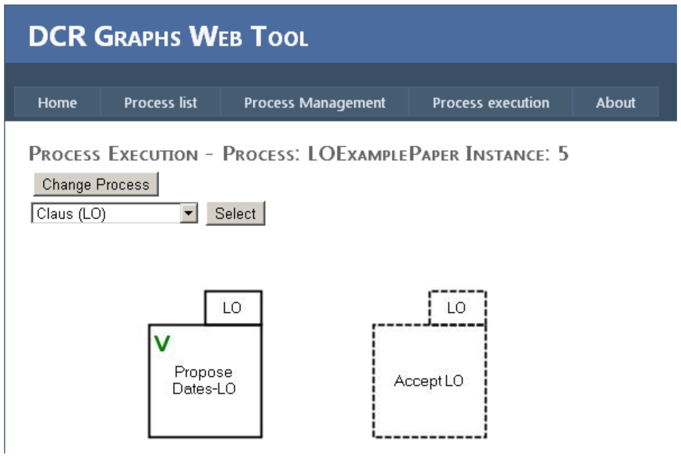
G = (E,▷, M, →•, •→, →◇, ±, Act,l, R,P, as), where
Act = atoms(E) = {E-M, E-D, SC, ACI, EM, D-U, D-D, PLO, PDA, ALO, ADA, HM}
E = {CC, AM, MC, E, D} ∪ atoms(E)
▷ = {(E-M, E), (E-D, E), (E, CC), (SC, CC), (ACI , CC), (PLO, AM), (PDA, AM), (ALO, AM), (ADA, AM), (ALO, D-D), (D-U, D), (D, MC), (EM, MC)}
M = (∅, ∅, atoms(E) \ { ALO, ADA })
→• = {(E, SC), (SC, ACI), (ACI, MC), (ACI, PLO), (D-U, D-D), (PLO, PDA)}
•→ = {(SC, ACI), (SC, PLO), (PLO, ADA), (PDA, ALO), (AM, HM)}
→◇ = {(AM, HM)}
→+ = {(PLO, ADA), (PDA, ALO)}
→% = {(SC, SC), (ALO, ALO), (ALO, ADA), (ADA, ADA), (ADA, ALO)}
l = {e ∈ atoms(E) | (e, e) }
R = {U, LO, DA} and P= {U, LO, DA}
as = {(SC, U), (E, U), (D, U), (ACI, LO), (EM, LO), (D, LO), (PLO, LO), (ALO, LO), (HM, LO), (EM, DA), (D, DA), (PDA, DA), (ADA, DA), (U, U), (LO, LO), (DA, DA)}

During the case study it became clear that it would be useful to have design tools allowing to quickly create and simulate models. In the following section we describe the tools developed so far. In Sec. VI we describe the plans for future development of tools along with the challenges for extending the theory identified in the case study.

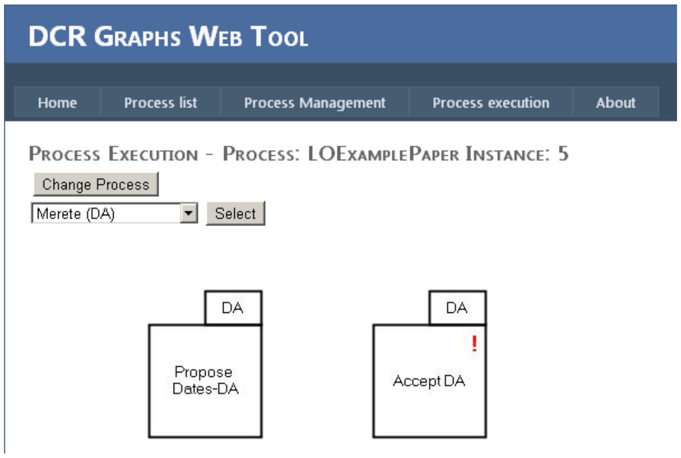
IV. Prototype Tools

To support designing with DCR Graphs, making the model available to a wider audience and allow interested parties to experiment with the notation, we are developing prototype implementations of various tools for DCR Graphs. Development up to this point includes:

1) A process repository; a service which can be used to store and retreive DCR processes and process instances.
2) An execution host; a service which can be used to execute DCR process instances.
3) A windows-based graphical editor; which can be used to model DCR Graphs and run simple simulations on them.
4) A windows-based desktop client for executing process instances.
5) A platform independent web client; which can also be used to execute process instances. In the future we aim to support the creation of processes through this webinterface as well. (Fig. 3)


DCR Graphs Web Tool
Home | Process list | Process Management | Process execution | About
Process Execution - Process: LOExamplePaper Instance: 5
Change Process
Claus (LO) | Select

(a) Execution by LO


DCR Graphs Web Tool
Home | Process list | Process Management | Process execution | About
Process Execution - Process: LOExamplePaper Instance: 5
Change Process
Merete (DA) | Select

(b) Execution by DA

Figure 3. Execution in the Web Tool

6) A model checking and runtime verification tool; which interfaces to SPIN [24] and ZING [21] model checkers for model checking.
7) A runtime-monitor that can subscribe to the execution host and verify that the execution of processes adheres to given properties.

Fig. 4 shows how these tools interact: Usually, a process modeller will first create a process in the graphical editor (Fig. 5) , which will be stored in the process repository. The process modeller can use the verification tool to check if his process adheres to the properties that he desires. Both safety and liveness properties on models can be verified with the help of SPIN [24] model checker, where as only safety properties on DCR Graphs can be verified using ZING [21] model checker, as ZING does not support liveness properties. A user can login to the web or desktop-client and select the process for execution. The client will request that the process repository start a new instance and the repository will provide the client with the description of the process and runtime information on the process instance. Execution requests are made to the execution server, which handles these requests atomically,

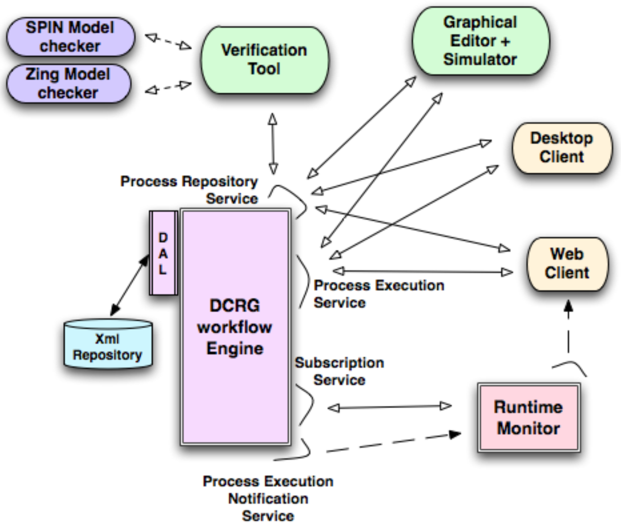


Figure 4. Protoype Architecture

making updates to the instance stored on the repository. If a request is invalid, the execution server will notify the user and leave the process instance in its original state. The runtime monitor can subscribe to the execution server and will get notified of every execution request. It will then check if the execution of the process follows the properties described for it.

Listing 1 shows a brief overview of the XML format of DCR Graphs that is being used in all prototype tools. A single XML format is used to contain information about both the specification and the runtime of a DCR Graph. The resources section of the specification contains information about roles, principals, events and actions, whereas the access controls section contains the mapping of principals and actions to roles. The last part of the specification contains the binary relations between the events. Note that the XML format supports nesting of events and the binary relations in between them and that flattening of nested events and their relations will be done at the beginning of executing a DCR Graph.

The second part of the XML format for a DCR Graph holds the runtime information, which primarily contains the execution trace and information about the current state. The execution trace records the actual sequence of events executed and the current state holds the information about the current marking which contains sets of included, executed and pending response events. In addition to the marking, the current state also holds additional information such as index of state copy, state accepted to support the acceptance condition for infinite computations that were characterized by mapping to Büchi-automata in [12], [17].

Listing 1. Overview of DCR Graph Xml
```
<?xml version="1.0" encoding="utf-8"?>
<dcrg:process xmlns:dcrg="http://itu.dk/trustcare/dcr/2011/">

  <dcrg:specification processId="" modelName="">
    <dcrg:resources>
        <dcrg:roles>.....</dcrg:roles>
        <dcrg:principals>.....</dcrg:principals>
        <dcrg:events>.....</dcrg:events>
        <dcrg:actions>.....</dcrg:actions>
     </dcrg:resources>

     <dcrg:accessControls>
        <dcrg:rolePrincipalAssignments>.....</
             dcrg:rolePrincipalAssignments>
        <dcrg:actionRoleAssignments>.....</
             dcrg:actionRoleAssignments>
      </dcrg:accessControls>

      <dcrg:constraintSets>
        <dcrg:constraintSet type="condition">...</
             dcrg:constraintSet>
        <dcrg:constraintSet type="response">...</
             dcrg:constraintSet> .....
       </dcrg:constraintSets>
  </dcrg:specification>

  <dcrg:runtime processInstanceId="">
      <dcrg:executionTrace> </dcrg:executionTrace>
      <dcrg:currentState stateId="">

        <dcrg:eventsIncluded>.....</dcrg:eventsIncluded>

        <dcrg:eventsExecuted>.....</dcrg:eventsExecuted>

        <dcrg:eventsPendingResponses>.....</
             dcrg:eventsPendingResponses>

        <dcrg:stateAccepting> </dcrg:stateAccepting>

        <dcrg:stateIndex> </dcrg:stateIndex>

        <dcrg:eventsEnabled>.....</dcrg:eventsEnabled>

        </dcrg:currentState>
  </dcrg:runtime>

</dcrg:process>
```

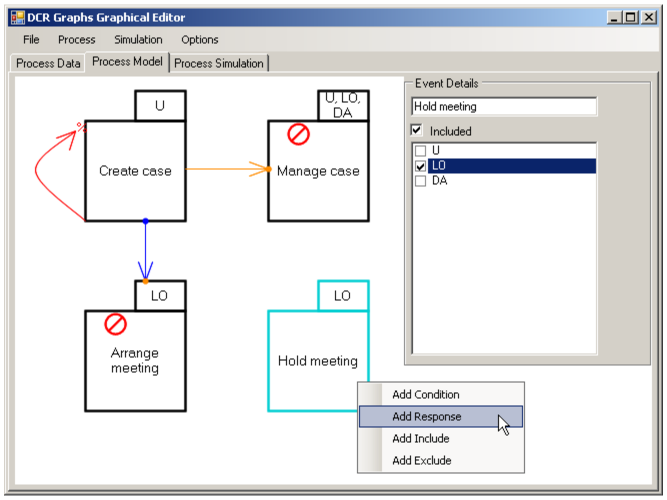
The specification section of the XML document for the Case Handling Process shown in the figure 2 is given in listing 2.

Listing 2. DCRG specification in Xml
```
<dcrg:specification>

<dcrg:resources>
<dcrg:roles>
   <dcrg:role>U</dcrg:role>
   <dcrg:role>LO</dcrg:role>
   <dcrg:role>DA</dcrg:role>
</dcrg:roles>
<dcrg:principals>
   <dcrg:principal>u</dcrg:principal>
   <dcrg:principal>lo</dcrg:principal>
   <dcrg:principal>da</dcrg:principal>
</dcrg:principals>
<dcrg:events>
    <dcrg:event eventId="0" name="Create case" actionId="
         Create case">
        <dcrg:event eventId="1" name="Submit" actionId="
             Submit" />
        <dcrg:event eventId="2" name="Assign case Id"
             actionId="Assign case Id" />
        <dcrg:event eventId="3" name="Edit" actionId="Edit">
                <dcrg:event eventId="4" name="Metadata"
                     actionId="Metadata" />
                <dcrg:event eventId="5" name="Dates available
                     " actionId="Dates available" />
        </dcrg:event>
    </dcrg:event>
    <dcrg:event eventId="6" name="Manage case" actionId="
         Manage case">
        <dcrg:event eventId="7" name="Edit metadata" actionId
             ="Edit metadata" />
        <dcrg:event eventId="8" name="Document" actionId="
             Document">
```

```
                <dcrg:event eventId=”9” name=”Upload”
                    actionId=”Upload” />
                <dcrg:event eventId=”10” name=”Download”
                    actionId=”Download” />
            </dcrg:event>
        </dcrg:event>
        <dcrg:event eventId=”11” name=”Arrange Meeting” actionId
            =”Submit”>
            <dcrg:event eventId=”12” name=”Propose dates–LO”
                actionId=”Propose dates–LO” />
            <dcrg:event eventId=”13” name=”Accept LO” actionId=”
                Accept LO” />
            <dcrg:event eventId=”14” name=”Accept DA” actionId=”
                Accept DA” />
            <dcrg:event eventId=”15” name=”Propose dates–DA”
                actionId=”Propose dates–DA” />
        </dcrg:event>
        <dcrg:event eventId=”16” name=”Hold meeting” actionId=”
            Hold meeting” />
</dcrg:events>
<dcrg:actions>
            <dcrg:action actionId=”Create case” />
            <dcrg:action actionId=”Submit” />
            <dcrg:action actionId=”Edit” />
            <dcrg:action actionId=”Metadata” />
            <dcrg:action actionId=”Dates available” />....
</dcrg:actions>
</dcrg:resources>
<dcrg:accessControls>
        <dcrg:rolePrincipalAssignments>
            <dcrg:rolePrincipalAssignment role–name=”U”>
                <principal>u</principal>
            </dcrg:rolePrincipalAssignment>
            <dcrg:rolePrincipalAssignment role–name=”LO”>
                <principal>lo</principal>
            </dcrg:rolePrincipalAssignment>
        </dcrg:rolePrincipalAssignments>
        <dcrg:actionRoleAssignments>
            <dcrg:actionRoleAssignment actionId=”Submit”>
                <dcrg:role>U</dcrg:role>
            </dcrg:actionRoleAssignment>
            <dcrg:actionRoleAssignment actionId=”Document”>
                <dcrg:role>U</dcrg:role>
                <dcrg:role>LO</dcrg:role>
                <dcrg:role>DA</dcrg:role>
            </dcrg:actionRoleAssignment>    ....
          </dcrg:actionRoleAssignments>
</dcrg:accessControls>
<dcrg:constraintSets>
        <dcrg:constraintSet type=”condition”>
            <dcrg:constraint source=”1” target=”2” />
            <dcrg:constraint source=”3” target=”1” />.....
        </dcrg:constraintSet>
        <dcrg:constraintSet type=”response”>
            <dcrg:constraint source=”1” target=”2” />.....
        </dcrg:constraintSet> .....
</dcrg:constraintSets>

</dcrg:specification>
```

All the prototype tools support the basic DCR Graph notation containing condition, response, include and exclude relations. We are currently working on extending the prototype to support milestone relations and nested events. In Fig. 6, 7, 8, we have illustrated how the execution state of the case-handling process may be visualized in the simulator in the future.

The graph in the figure. 6 shows the state after a run where the union started by creating a case: they edited meta-data, indicated the dates they were available and submitted. When LO received the case they assigned their own case ID to it. Some time later LO proposed possible dates for a meeting to DA. DA did not agree with these dates and responded by proposing some of their own. In the graph both Accept LO and Accept DA are included and have a pending response because both LO and DA have proposed dates. Because of



Event Details

Hold meeting

☑ Included

☐ U
☑ LO
☐ DA

Figure 5. The Graphical Editor

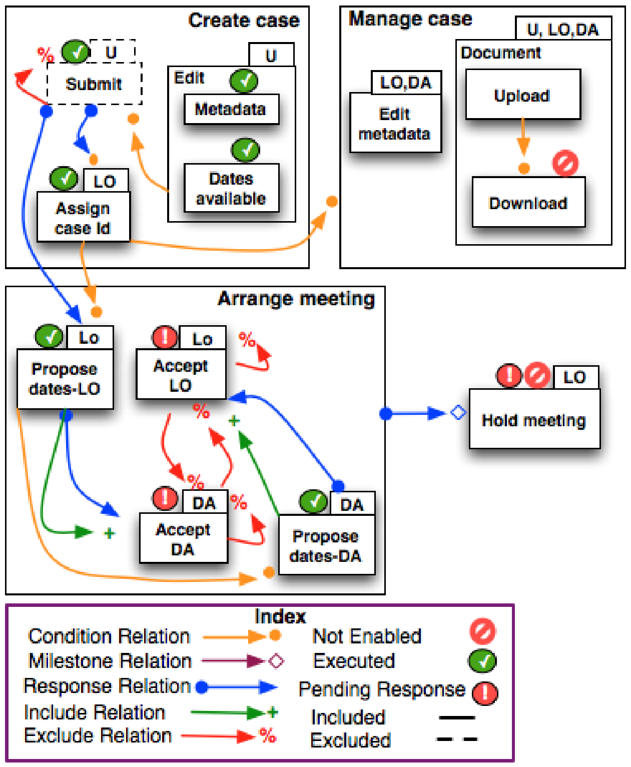


Figure 6. Case Handling Process Runtime

these pending responses Hold meeting is disabled. Because no files have been uploaded to the document yet, Download is also disabled. The listing 3 shows the runtime information for the case handling process from the figure 6.

Listing 3. DCRG Runtime in Xml
```
<dcrg:runtime processInstanceId=””>
      <dcrg:executionTrace>4,5,4,1,2,12,15</
            dcrg:executionTrace>
      <dcrg:currentState stateId=”S6”>
```

```
        <dcrg:eventsIncluded>2,4,5,7,9,10,12,13,14,15,16</
            dcrg:eventsIncluded>
        <dcrg:eventsExecuted>1,2,4,5,12,15</
            dcrg:eventsExecuted>
        <dcrg:eventsPendingResponses>13,14,16</
            dcrg:eventsPendingResponses>
        <dcrg:stateAccepting>0</dcrg:stateAccepting>
        <dcrg:stateIndex>0</dcrg:stateIndex>
        <dcrg:eventsEnabled>1,2,4,5,7,8,12,13,14,15</
            dcrg:eventsEnabled>
      </dcrg:currentState>
</dcrg:runtime>
```

The graph in the figure. 7 shows the runtime state after the union has uploaded an agenda for the meetings. Note that, since the union has uploaded a file to the case, Download is now enabled. But at the same time, Accept LO and Accept DA still remain the same as the previous graph, as the proposed dates have not been accepted yet by either LO or DA.

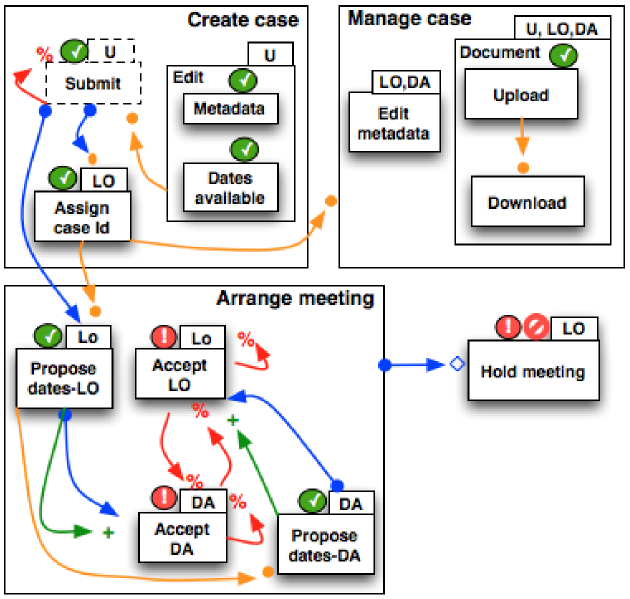


Figure 7. Case Handling Process Runtime After Upload Document

Figure 8 shows the graph representing the state after LO has accepted one of the dates proposed by DA. Note that both Accept LO and Accept DA are excluded due to the mutual exclude relation between them. Even though there is a pending response on Accept DA, it is not considered relevant as it is excluded and Hold meeting has become pending because of the response relation. Continuing by executing Hold meeting as LO will cause the graph to reach an accepting state, as there will be no included pending responses.

V. Comparison to Other Approaches

As already mentioned in the introduction, our approach is closely related to the work on DECLARE [27], [28]. In particular the condition and response relations are also considered in [27], [28], and we have used the same graphical notation as loc. cit. The crucial difference is that we focus on a few core constraints allowing to describe the state and operational semantics of processes as a labelled transitions between simple

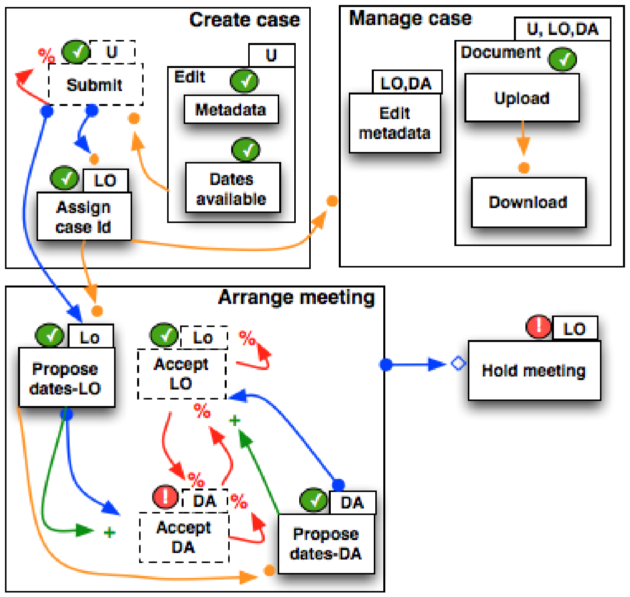


Figure 8. Case Handling Process Runtime After Accept Dates

markings consisting of three sets of respectively executed, included and required events. As also pointed out in [27], [28], the generality of LTL offers much flexibility with respect to specifying execution constraints but makes it more complex to execute processes given in DECLARE and to describe and understand their run-time state. It typically requires a translation of the constraints to LTL and subsequent using the standard mapping of an LTL formula to a Büchi-automaton. In particular, there is no obvious way to trace the graphical constraints in DECLARE to the states of the Büchi-automaton. Moreover, we show in a follow up paper that every set of traces expressible in LTL (and thus DECLARE) can also be expressed using DCR Graphs.

We have shown in [18] it is possible, but much more complex to represent in LTL the interplay between dynamic inclusion/exclusion and the other relations. Neither this novel notion of dynamic inclusion/exclusion relations nor nesting are considered in [27], [28].

The DCR Graphs model also relates to the independent work on the *Guard-Stage-Milestone* model [15] by Hull et al presented as an invited talk at the WS-FM 2010 workshop and part of the work on artifact-centric business processes [2], [5], [7].

Finally, BPMN 2.0 includes the *ad-hoc sub-process activity* which allows one to group a set of activities that can be carried out in an ad-hoc way. Fig. 9 below shows how one may attempt to describe the top level requirements described in Fig. 1 as a BPMN 2.0 ad-hoc sub-acitivity. According to the informal description of BPMN 2.0 ad-hoc sub-process activity in the current BPMN 2.0 specification ([19]) Create case is a *condition* for Manage case since the latter cannot start without the *data object* case as input, which is produced

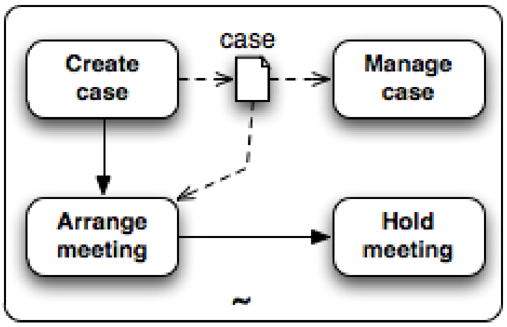


Figure 9. BPMN 2.0 ad-hoc sub-process activity

by the former. Moreover, quoting from the specification, the sequence flow between Create case and Arrange meeting (and similarly between Arrange meeting and Hold meeting): ”*creates a dependency where the performance of the first Task MUST be followed by a performance of the second Task. This does not mean that the second Task is to be performed immediately, but there MUST be a performance of the second task after the performance of the first Task.*”. This seems exactly to correspond to the response relation in DCR Graphs. However, when reading the semantics section of the specification ([19], Sec.13.2.5, 445-446) it appears that the sequence flow introduces just a standard precondition. Thus, the specification is not consistent in the description of sequence flows within ad-hoc sub-activities. Also, it is not clear how to specify roles on actions (swim lanes seem not to be allowed within ad-hoc sub-activities) nor how to specify that an activity within an ad-hoc sub activity only can be executed once. In particular, Create case can be executed any number of times in the above process.

VI. Conclusions and Future Work

Our case study showed that the DCR Graphs model is well suited to give a global description of the temporal constraints between the individual tasks which is helpful in capturing the requirements of the overall system.

However, there are still many points for future developments.

First of all there is the need to extend the expressiveness of DCR Graphs. In the ongoing PhD project of the second author we intend to extend the DCR Graphs model to be able to express relevant features such as multi-instance sub-graphs (allowing the dynamic creation of sub-graphs representing dynamic sub process instantiation), time, exceptions and data. Along with this we intend to continue developing the technology for model checking and run time verification and apply it within case studies.

Second, our industrial partner Resultmaker who already use a declarative process model based on the primitives in the DCR Graphs model expects to investigate the use of the formalization to support *safe* dynamic changes to the process constraints at run time.

Thirdly, the DCR Graphs model presently describe a global view of the process. Through our discussions with Exformatics during the case study we identified the wish to be able to automatically synthesize distributed views of the process. In particular, they wanted to be able to derive descriptions of communication protocols and message exchange between the individual local components in a distributed implementation of the system.

Derivations of descriptions of communication protocols between local components from a global model is been researched for the imperative choreography language WS-CDL in the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [4]. Put briefly, the work formalizes the core of WS-CDL as the global process calculus and define a formal theory of end point projections projecting the global process calculus to abstract descriptions of the behavior of each of the local ”end-points” given as pi-calculus processes typed with session types.

We are currently working on the challenge of synthesizing a distributed view of a DCR Graph as a set of interacting DCR Graphs, thus providing a declarative notion of end-point projections. As a challenge for future work we propose to provide a formal map between DCR Graphs and imperative choreographies formalized in the global process calculus [4].

References

[1] Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, SAP. Ws-bpel extension for people (bpel4people) version 1.0, 2007. http://www.adobe.us/content/dam/Adobe/en/devnet/livecycle/pdfs/bpel4people_spec.pdf.

[2] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *In preparation*, pages 288–304, 2007.

[3] Christoph Bussler and Stefan Jablonski. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*, pages 53–59, Feb 1994.

[4] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *16th European Symposium on Programming (ESOP'07)*, LNCS, pages 2–17. Springer, 2007.

[5] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

[6] Hasam Davulcu, Michael Kifer, C. R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of ACM SIGACT-SIGMOD-SIGART*, pages 1–3. ACM Press, 1998.

[7] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 252–267, New York, NY, USA, 2009. ACM.

[8] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.

[9] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Comput. Surv.*, 12:27–60, March 1980.

[10] Clarence A. Ellis and Gary J. Nutt. Workflow: The Process Spectrum. In Amit Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, May 1996.

[11] Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. http://www.trustcare.dk/.

[12] Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Post-proceedings of PLACES 2010*, 2010.

[13] Thomas Hildebrandt and Raghava Rao Mukkamala. Distributed dynamic condition response structures. In *Pre-proceedings of International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 10)*, March 2010.

[14] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Fundamentals of Software Engineering Confereence 2011 (to appear)*, April 2011.

[15] Richard Hull. Formal study of business entities with lifecycles: Use cases, abstract models, and results. In *Proceedings of 7th International Workshop on Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, 2010.

[16] Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Proceedings ProHealth 08 workshop*, 2008.

[17] Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010.

[18] Raghava Rao Mukkamala, Thomas Hildebrandt, and Janus Boris Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008.

[19] Object Management Group BPMN Technical Committee. Business Process Model and Notation, version 2.0, 2010. http://www.omg.org/spec/BPMN/2.0/.

[20] Organization for the Advancement of Structured Information Standards (OASIS). Web services human task (ws-humantask) specification, version 1.1, 2009. http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.pdf.

[21] Microsoft Research. Zing model checker. Webpage, 2010. http://research.microsoft.com/en-us/projects/zing/.

[22] Pinar Senkul, Michael Kifer, and Ismail H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *In VLDB*, pages 694–705, 2002.

[23] Munindar P. Singh, Greg Meredith, Christine Tomlinson, and Paul C. Attie. An event algebra for specifying and scheduling workflows. In *Proceedings of DASFAA*, pages 53–60. World Scientific Press, 1995.

[24] Spin. On-the-fly, ltl model checking with spin. Webpage, 2008. http://spinroot.com/spin/whatispin.html.

[25] Keith D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press, 2010.

[26] Wil M. P. van der Aalst and S Jablonski. Dealing with workflow change: Identification of issues and solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.

[27] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[28] Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006.

[29] W3C. Web services choreography description language, version 1.0, 2005. http://www.w3.org/TR/ws-cdl-10/.

[30] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.

[31] Glynn Winskel and Mogens Nielsen. Models for concurrency. pages 1–148, 1995.

[32] M. D. Zisman. *Representation, Specification and Automation of Office Procedures.* Philadelphia, Pa.: University of Pennsylvania, Wharton School, Department of Decision Sciences, Ph.D. Thesis, Sep, 1977.

Hierarchical Declarative Modelling with Refinement and Sub-processes

Søren Debois[1], Thomas Hildebrandt[1], and Tijs Slaats[1,2]

[1] IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
`{debois,hilde,tslaats}@itu.dk`
[2] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark

Abstract. We present a new declarative model with composition and hierarchical definition of processes, featuring (a) incremental refinement, (b) adaptation of processes, and (c) dynamic creation of sub-processes. The approach is motivated and exemplified by a recent case management solution delivered by our industry partner Exformatics A/S. The approach is achieved by extending the Dynamic Condition Response (DCR) graph model with *interfaces* and composition along those interfaces. Both refinement and sub-processes are then constructed in terms of that composition. Sub-processes take the form of hierarchical (complex) events, which dynamically instantiate sub-processes. The extensions are realised and supported by a prototype simulation tool.

1 Introduction

Business process design technologies today are predominantly based on *flow-oriented* process notations such as the Business Process Model and Notation (BPMN) standard [18], which *imperatively* describes how a process should proceed from start to end. Often, business processes are required to be compliant with regulations and constraints given by busines polices, standards and laws. E.g., a customer must be informed about alternatives and risks before getting a loan in a bank, or a decision on a grant application cannot be made before the deadline for submissions of applications has been reached.

Since the flow-oriented notations only captures *how* to fulfill the compliance rules, the description and verification of compliance rules require other notations and techniques. This leaves the process designers with three modelling tasks: To describe the compliance rules, to describe the process, and to verify that the process is compliant to the rules. Typically, compliance rules are described *declaratively* using a variant of temporal logic such as Linear-time Temporal Logic (LTL) [21]. Compliance can then be verified during execution using runtime verification techniques [12] and, if the flow-diagrams are based on a formal model, also at design time [6]. In most industrial design tools, the flow-diagrams are however *not* based on a formal model, and consequently, design time verification is not supported. This means that the process designers have to figure out manually how to interpret the constraints, and compliance is then subsequently verified informally and approved by, e.g., a lawyer. At best, a formal run-time or post-execution verification is performed against the execution log.

In these situations there is a high risk that processes become either non-compliant or over-constrained by design, to faciliate manual verification. Over-constrained processes, however, rarely fits reality or are simply not suitable for knowledge-intensive processes. A way to avoid these problems is to use the declarative approach (also) for the process design. Several declarative process modelling notations and techniques have been proposed in the last decade, including DECLARE [2,1], CLIMB [13], GSM [11] and Dynamic Condition Response (DCR) graphs [8,14]. However, sometimes the declarative approach makes it less clear from the end-user, how a process will proceed from start to end. Even with a graphical notation (as in DECLARE, GSM and DCR graphs), it may be difficult to comprehend the interactions between different constraints.

The DCR graph process modelling notation stands out by supporting a simple and efficient run-time execution, which mitigates the complexity of comprehending the constraints and allows for run-time adaptation [15], while still being more expressive than (propositional) LTL (and thus DECLARE), in that it allows to describe every union of a regular and an ω-regular language [3,16,14].

DCR Graphs were conceived as both a generalization of event structures [22] and a formalization and generalization of the Process Matrix [17] invented by Danish company Resultmaker. Since its inception, the DCR Graph notation and theory have been developed further in collaboration with Exformatics A/S, a Danish provider of case, document and knowledge management systems. A version of DCR graphs with a simple notion of nesting [9], an additional milestone relation, and support for data now forms the core their workflow engine [20,7]. However, DCR graph models as currently implemented become difficult to comprehend and present at a certain size. They seem to lack encapsulation, modularity and hierarchy; the key techniques to make large models comprehensible in both imperative [19] and declarative settings [23]. Also, practical modelling efforts by Exformatics A/S has revealed that DCR graphs emphatically needs a notion of "dynamically created" or "instantiated" sub-process.

In the present paper, we seek to remedy these shortcomings of DCR graphs. Our contributions are as follows.

1. We introduce refinement-by-composition for DCR graphs.
2. We add to DCR graphs a notion of *dynamically spawned sub-process*, defining Hi-DCR graphs.
3. We demonstrate the use of both incremental process design using an example exctracted from a recent case management solution delivered by Exformatics A/S to a Danish funding agency.
4. We provide a publicly available Hi-DCR graph tool.

The tool allows simulation, model-checking of the finite fragment, automatic visualisation and more. The compositions and refinements presented in examples were not made by hand, they were executed by the tool; all DCR diagrams in this paper has been generated by it, and all examples are fully executable by it.

Hi-DCR graphs are fully formalised; we prove both soundness of refinement—that refinement cannot accidentally remove constraints of the extant model—as well as Hi-DCR being strictly more expressive than ω-regular languages.

1.1 Related Work

Hierarchy for declarative languages was studied in [23], where the authors add *complex activities* to DECLARE [2,1]. The authors make a compelling case that hierarchy is a necessity for constructing understandable declarative models. Our industry partner's experiences fully supports this thesis; this is in part what has led to our investigation of sub-processes.

A complex activity is one which contains a nested DECLARE model governing when that activity may complete. The nested model starts when the complex activity opens, and the complex activity conversely may only close once the nested model completes. Otherwise, there is no interaction between the nested model and the parent model. In the present paper, a sub-process may interact with its parent process: there can be multiple ways to start the sub-process, it can have different observable outcomes, and it is allowed to interact with other activities in the parent process.

Questions about concurrency are left open by [23]: the authors do not report a formal semantics, and the paper has no examples of interleavings of complex activities. In the present approach, sub-process executions are naturally interleaved with other events and even other instances of the same sub-process; we shall see this in examples.

We believe it is straightforward to formalize complex activities of [23] in Hi-DCR Graphs: Use Hi-DCR relations to allow only a single start and end event for each sub-process, and cut off interaction between sub- and super-process by choosing only empty interfaces.

The Guard-Stage-Milestone (GSM) approach [11] to business modelling provides a data-centric notation with declarative flavour. The notation consists of *stages*, which in turn have *guards*, controlling when and how the stage may start, and *milestones*, controlling when and how a stage may close. Stages can contain sub-stages, giving GSM an inherent hierarchy. Where GSM is data-centric, the present formalism is event-based. Nonetheless, the sub-processes of Hi-DCR graphs are strongly reminiscent of GSM stages, with Hi-DCR interface events assuming the rôle of guards and milestones. In future work, we plan to further investigate the similarities between GSM and Hi-DCR Graphs, in the hope of providing a formal connection between them.

2 DCR graphs

In this section, we recall DCR graphs as introduced in [8,14] and introduce our running example. The example is based on a workflow of a Danish funding agency; our industry partner, Exformatics A/S, has implemented system support for this workflow using the basic DCR graphs of this Section [20]. While vindicating DCR graphs as a flexible and practical modelling tool, that work also highlighted the potential need for refinement and sub-processes, which we introduce in the following Sections. One key idea will be the development of models by *refinement*: start from a very abstract model, then successively refine it until it becomes suitably concrete. In this section, we introduce DCR graphs alongside such a very abstract model.

As the name suggests, DCR graphs are *graphs*, and we tend to represent them visually, as in Fig. 1. This figure depicts a highly abstracted model of the funding agency workflow. Events (boxes) with labels (the text inside them) are related to other events by various arrows. In this model there are only four events: the beginning of the application round Start round; receiving an application (Receive application); the deadline for application submission occurring (Application deadline); and finally the board meeting (Board meeting), at which the board of the funding institution decide which applications warrant grants and which do not.

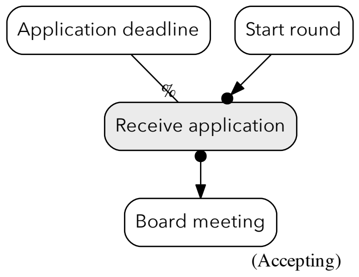


Fig. 1. A basic DCR graph

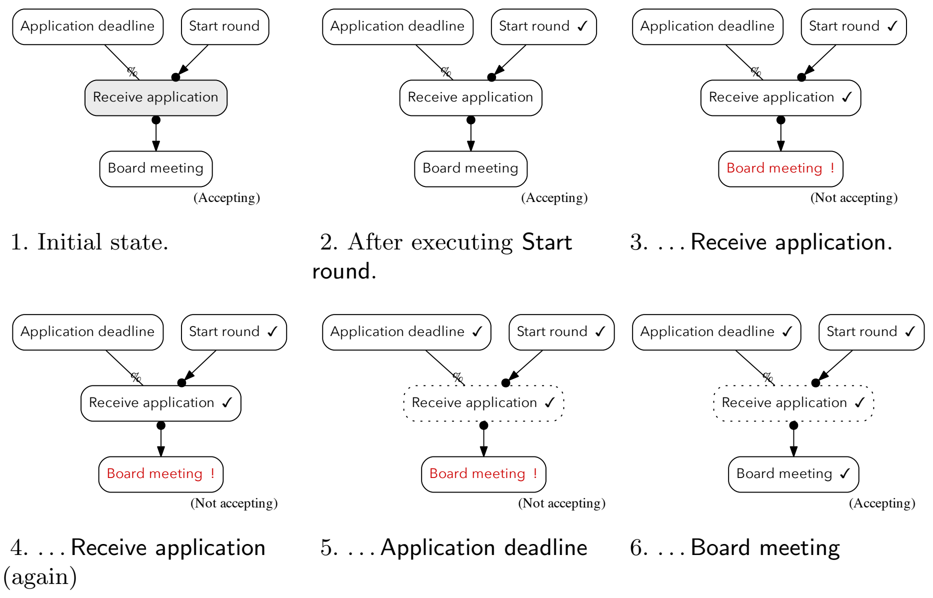
Relations between these events govern their relative order of occurrence. When not constrained by any relations, events can happen in any order and any number of times.

The *condition* relation, $e \rightarrow\!\bullet\ e'$, seen between Start round and Receive application indicates that the former must occur before the latter: we do not receive applications before the round has started. In the initial state of the DCR graph, Start round have yet to happen, and so Receive application cannot execute; hence it has been greyed out in the visual representation. The notation and semantics of this relation is similar to the precedence constraint in DECLARE [2,1].

Between Receive application and Board meeting we have a *response* relation, $e \bullet\!\!\rightarrow e'$. This indicates that if Receive application happens, then Board meeting must subsequently happen. This does not necessarily mean that each occurrence of the former is followed by a unique occurrence of the latter; it's quite all right to receive seven applications, have a board meeting, receive five more applications, then have a final board meeting. If Receive application has been executed without a following Board meeting, we say that Board meeting is *pending*.

Finally, between the Application deadline and Receive application events we have an *exclusion* relation, $e \rightarrow\!\% \ e'$. Once the Application deadline event occurs, the Receive application event becomes excluded, which means that it is from then on considered irrelevant for the rest of the workflow. While excluded, it cannot execute; any response obligations on it are considered void; and if it is a condition for some other event, that condition is disregarded. Dual to the exclusion relation is the *inclusion* relation. It is not exemplified in this DCR graph, but its meaning is straightforward: it re-includes an event in the workflow. DCR graph have also a fifth and final relation, the *milestone* relation $e \rightarrow\!\diamond\ e'$; we will postpone explaining that until we use it in the next Section.

A key advantage of DCR graphs is that the graph *directly* represents the state of execution. There is no distinction between design-time and run-time. We will illustrate this by example: in Fig. 2 we have a finite execution of Fig. 1. In the upper-left corner, (1) is the initial state, the DCR graph presented in



1. Initial state.

2. After executing Start round.

3. . . . Receive application.

4. . . . Receive application (again)

5. . . . Application deadline

6. . . . Board meeting

Fig. 2. Execution of the DCR graph of Fig. 1.

Fig. 1. The Start round event executes, taking us to (2). We can observe events having been executed in the state of the graph: executed events have little check-marks next to them, so in (2), Start round has such a check-mark. Also, with Start round executed, the condition for Receive application is fulfilled; it is now executable and thus no longer greyed out. We execute it to get to (3). Because there is a response from Receive application to Board meeting, that execution puts a pending response on Board meeting. This is indicated in (3) by the red text and the exclamation mark.

We execute Receive application to get to (4). This execution brings no change to the graph, which already had Receive application marked as previously executed, and already had a response on Board meeting. So we execute Application deadline, getting to (5). Because of the exclusion relation from that to Receive application, the latter becomes excluded, indicated by its box being dotted in (5). Even though excluded events cannot be executed, we do not grey them out; the dotted box is enough. Finally, we execute Board meeting to get to (6). This of course fulfils the pending response, which disappears: the text of Board meeting goes back to black, and the exclamation mark disappears.

A DCR graph is *accepting* if it has no included pending responses. (An infinite run is accepting if every incurred response is eventually executed or excluded.) The acceptance state of the graph is indicated in the lower-right corner of each graph. That indication is technically superfluous: the graph will be accepting exactly if it has no red labels/labels with exclamation marks. For large graphs, it can be convenient to have the single indicator anyway.

3 Hierarchy & Refinement

We now come to the core contributions of this paper. We present a notion of “refinement” of DCR graphs, defined in terms of a more primitive notion of “composition” of DCR graphs. Refinement is always achieved by composing an *abstract DCR Graph* with a *refinement DCR Graph*, which introduces new events and/or add additional constraints.

3.1 Refinement

We wish to refine our model to express in greater detail the decision mechanics of the board. We will model board meetings by the DCR graph in Fig. 3. The results of an application round must be gathered in a report. This report is updated and approved repeatedly during the application round. This gives rise to two new events: Update report and Approve report.

Applications are discussed over the course of several board meetings and the results of the board meetings must be worked into the report. To allow the secretary to work efficiently she is *not* required to formally update the report after every single board meeting, but she may combine the outcomes of several of them in a single update. This constraint is represented by the response relation from Board meeting to Update report.

While there are such pending changes to the report, it can of course not be approved. This is modelled using a *milestone* relation Update report $\rightarrow\!\diamond$ Approve report. This relation means that *while* Update report is pending, Approve report can not execute.

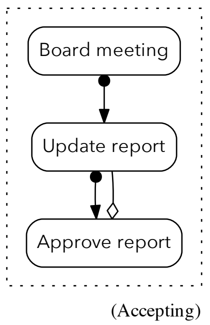


Fig. 3. Expanded model of the Board meeting

Note that this model does not preclude the board from re-approving a report that has not been updated. While not a particularly sensible thing to do, it is not against the rules, and as such *should* be permitted by the model.

Now, we wish to add these new details about board meetings to our original abstract model of Fig. 1; that is, we wish to *refine* Fig. 1 by Fig. 3. We do so by *composing* them: we fuse together events that are the same in both graphs. In this case only Board meeting. The result can be seen in Fig. 4. (The dashed box in that figure has no semantic ramifications; it is there are simply to make the graph easier to understand. See also [23].)

It is of course important that such a refinement does not accidentally *remove* constraints of the original model. Because of the inclusion and exclusion relations, that might happen, e.g., inclusions in the refining model might cause events excluded in the abstract one to be suddenly allowed. We shall prove in

Theorem 4.10 that, roughly, when the two models agree on when fused events are included or excluded, the composition will not admit new behaviour; in this case we call it a refinement. In the present case, the only fused event is Board meeting, which has no inclusions or exclusions going into it in either model, so this composition is really a refinement.

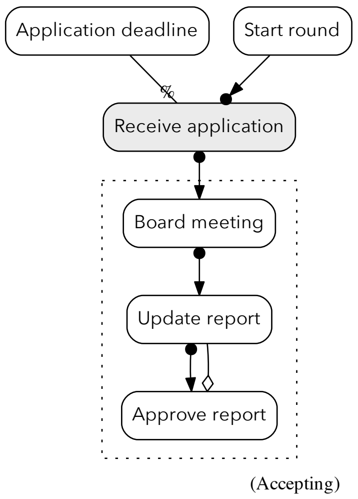


Fig. 4. Refinement of Fig. 1 by Fig.3.

Refinement-as-composition in conjunction with DCR graphs having no distinction between design-time and run-time means that we can *refine a running model.* Suppose, for instance, that we have deployed our initial abstract model of Fig. 1, and have reached state (5) in Fig. 2 when it is decided that compliance with the board meeting report procedure of Fig. 3 must be enforced. We may add in these new constraints by refining the running model (Fig. 2, part 5) with the new constraints (Fig. 3). Doing so yields the new DCR graph seen in Fig. 5. Note how the pending state of the fused Board meeting event is preserved. And again, by virtue of the refinement Theorem 4.10, we can be assured that all constraints on execution of the original model is still preserved in this new refined one.

3.2 Subprocesses

Refinement gives us a disciplined method for extending models with new components; thus it gives us a hierarchical notion of process design. However, it does not fully capture the notion of sub-processes in traditional business modelling notations. Here, a sub-processes is a complex activity in the model that has underlying behaviour which is *instantiated* when the sub-process is started and closed when the sub-process ends. Such sub-processes can both be single-instance, meaning that only one instance of the sub-process will be active at any time, or multi-instance, meaning that multiple instances of the sub-processes can execute concurrently.

To enable modelling such sub-processes we extend DCR graphs to Hi-DCR graphs. In these, we may associate with an event an entire *other* Hi-DCR graph which, when the event fires, is composed onto the current graph. We exemplify Hi-DCR Graphs by adding to our funding agency model a more detailed description of the process for an individual application. As many applications may be received and evaluated at the same time, we need a notion of sub-processes (and in particular multi-instance sub-processes) to fully capture this behaviour.

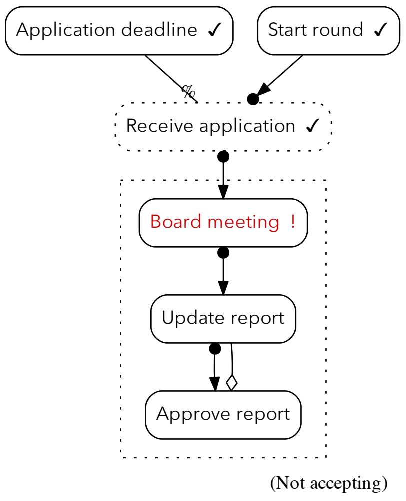


Fig. 5. Refinement of Fig. 2 part 5 by Fig. 3

An application must receive some number of reviews, with at least one from a lawyer. The reviews are collected in a review report. Based on the review report, the application is accepted or rejected and the round report is updated with this decision. It is not uncommon that the decision on an application is reverted, changing an "accept" to a "reject" or vice versa, and this may even happen several times as discussions progress. Of course, each change in the decision requires an update to the round report. Finally, each applications cannot remain in limbo and must always eventually be either accepted or rejected.

The DCR graph in Fig. 6 models this process. At the top is Lawyer review and Other review. Of these two only Lawyer review is a condition for Review report, with the effect that we cannot write the review report unless we have at least a review from a lawyer.

After the Review report is completed, the reviewers may Accept or Reject the application; as mentioned, there is no restriction that these events happen only once. However, each new verdict requires Update report because of the response relation from Accept and Reject to Update report.

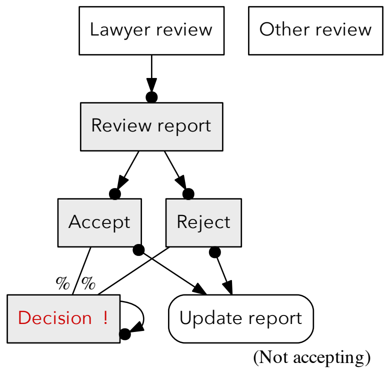


Fig. 6. The per-application subprocess.

Finally we need to model the fact that either Accept or Reject needs to occur at least once, similar to the *choice* construct in DECLARE. Hi-DCR graphs contain no construct directly analogue to choice; but fortunately, there is a straightforward way—a DCR graph idiom, if you will—to achieve the intended semantics. We explicitly model the fact that a decision is needed as an event Decision. We make this event a condition of itself, meaning that it cannot possibly be executed. We also make it initially included and pending, so that once the application is started, a decision needs to eventually be made. Finally we let both Accept and Reject exclude Decision, indicating that these two both represents a valid decision. Once Decision becomes excluded, it no longer prevents the larger graph from achieving an accepting state.

Now, we wish the entire sub process of Fig. 6 to be instantiated *once per application*. In Hi-DCR graphs this is achieved by associating with the event

Receive application in Fig. 5 the entire application processing DCR graph of Fig. 6. After executing Receive application a new copy of the application DCR Graph is composed with the main DCR Graph, and we get the DCR graph of Fig. 7.

Observe that once again, the common event Update report has been fused between the two DCR graphs. So far, this should be unsurprising: it is a straightforward application of the composition mechanism of DCR graphs. The key difference is that a Hi-DCR graphs is equipped also with a partitioning of its events into *interface events* (indicated by boxes with rounded corners in Fig. 6) and *local events* (indicated by boxes with non-rounded corners). This partitioning has the effect that under composition, only interface events are fused, whereas local events are not, even if their labelling overlap. The effect of these interfaces and local events will be apparent if we consider what happens when Receive application executes a *second* time; refer to Fig. 8. Here, we see that the

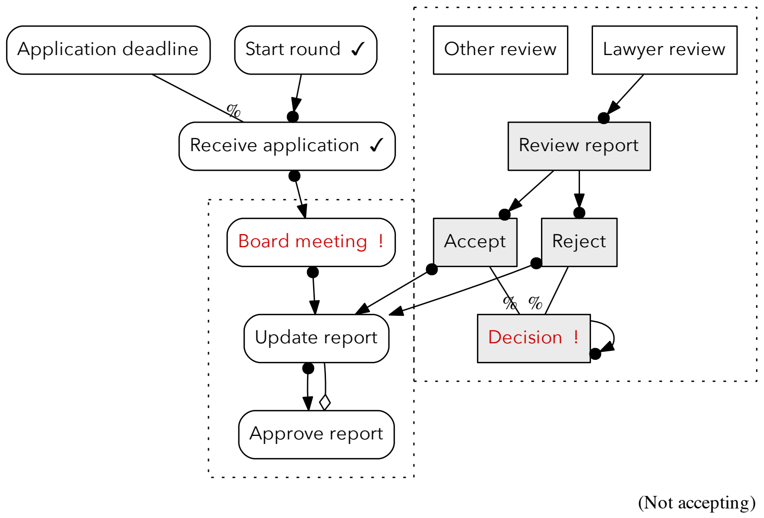


Fig. 7. Updated model with one spawned subprocess.

second application process has fused its interface event Update report, but has duplicated its remaining events, which are all local. This has the following two important consequences:

1. Each application process is represented separately.
2. Approve report effectively synchronises decisions: whenever the decision on any application is changed, the report needs to be updated.

Connecting local and interface event is a highly expressive mechanism. For instance, if we want to have a review report ready for every application before the board meeting commences, it is enough to have, in the sub-process definition in Fig. 6 a condition from the local event Review report to a new interface event Board meeting.

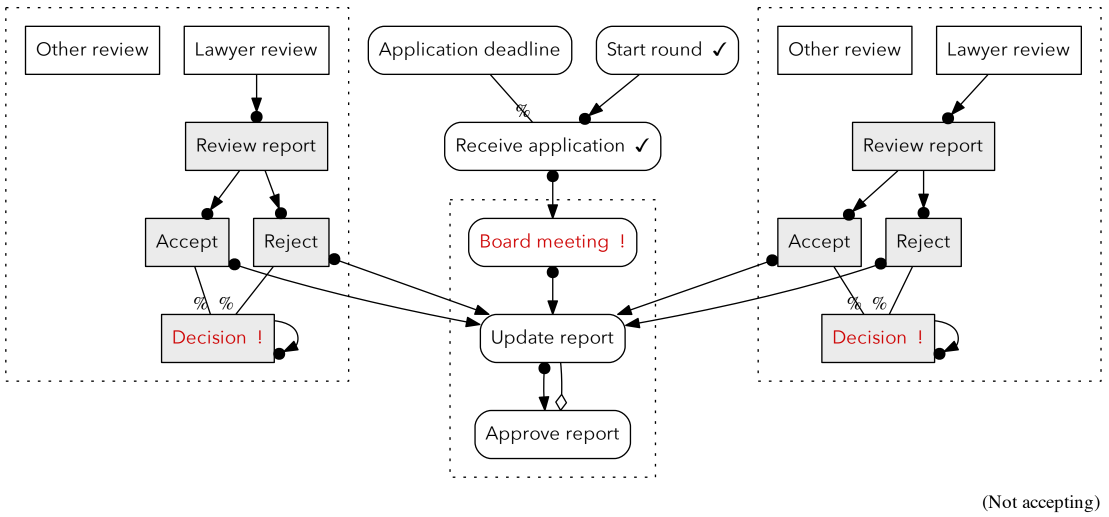


Fig. 8. After spawning a *second* subprocess in Fig. 7.

4 Foundations

In this section, we review the formal theory of DCR graphs, then formally introduce their refinement and their generalisation to Hi-DCR graphs.

We distinguish between events and labels. In a single workflow, the same label may occur multiple times. For instance, the label "Review report" occurs twice in Fig. 8. We accommodate such multiplicity by considering *events* (the boxes), as distinct from their *label* (the text in the boxes). When the distinction between events and labels does not matter, we use the words interchangeably. For instance, in Fig. 1 and 3 we speak of "the event Board meeting", since the label Board meeting uniquely identifies an event. To simplify the presentation we will let the labelling of events remain implicit in the formal definitions.

Definition 4.1 (DCR Graph [8]**).** *A* DCR graph *is a tuple* $(\mathsf{E}, \mathsf{R}, \mathsf{M})$ *where*

- E *is a finite set of (labelled)* events*, the nodes of the graph.*
- R *is the edges of the graph. Edges are partioned into five kinds, named and drawn as follows: The* conditions $(\rightarrow\!\bullet)$*,* responses $(\bullet\!\rightarrow)$*,* milestones $(\rightarrow\!\diamond)$*,* inclusions $(\rightarrow\!+)$*, and* exclusions $(\rightarrow\!\%)$*.*
- M *is the* marking *of the graph. This is a triple* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *of sets of events, respectively the previously executed* (Ex)*, the currently pending* (Re)*, and the currently included* (In) *events.*

When G is a DCR graph, we write, e.g., $\mathsf{E}(G)$ *for the set of events of G, as well as, e.g.,* $\mathsf{Ex}(G)$ *for the executed events in the marking of G.*

Notation. For a binary relation $\rightarrow \subseteq X \times Y$ we write "$\rightarrow Z$" for the set $\{x \in X \mid \exists z \in Z.\ x \rightarrow z\}$, and similarly for "$Z \rightarrow$". For singletons we usually omit the curly braces, writing $\rightarrow e$ rather than $\rightarrow \{e\}$.

With the definition of DCR graphs and notation in place, we define the dynamic semantics of a DCR graph. First, the notion of an event being *enabled*, ready to execute.

Definition 4.2 (Enabled events). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. We say that an event* $e \in \mathsf{E}$ *is* enabled *and write* $e \in \mathsf{enabled}(G)$ *iff (a)* $e \in \mathsf{In}$*, (b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$*, and (c)* $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

That is, enabled events (a) are included, (b) their included conditions already executed, and (c) have no included milestones with an unfulfilled responses.

Definition 4.3 (Execution). *Let* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M})$ *be a DCR graph with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. Suppose* $e \in \mathsf{enabled}(G)$*. We may* execute *e obtaining the* resulting *DCR graph* $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$ *with* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *defined as follows.*

1. $\mathsf{Ex}' = \mathsf{Ex} \cup e$
2. $\mathsf{Re}' = (\mathsf{Re} \setminus e) \cup (e\bullet\!\rightarrow)$
3. $\mathsf{In}' = (\mathsf{In} \setminus (e\!\rightarrow\!\%)) \cup (e\!\rightarrow\!+)$

That is, to execute an event e one must: (1) add e to the set Ex of executed events. (2) Update the currently required responses Re by first removing e, then adding any responses required by e. (3) Update the currently included events by first removing all those excluded by e, then adding all those included by e.

Definition 4.4 (Transitions, runs, traces). *Let G be a DCR graph. If* $e \in \mathsf{enabled}(G)$ *and executing e in G yields H, we say that G has* transition on *e* to *H and write* $G \longrightarrow_e H$*. A* run *of G is a (finite or infinite) sequence of DCR graphs* G_i *and events* e_i *such that:* $G = G_0 \longrightarrow_{e_0} G_1 \longrightarrow_{e_1} \ldots$
A trace *of G is a sequence of labels of events* e_i *associated with a run of G. We write* $\mathsf{runs}(G)$ *and* $\mathsf{traces}(G)$ *for the set of runs and traces of G, respectively*

Not every run or trace represents an acceptable execution of the graph: We need also that every response requested is eventually fulfilled or excluded.

Definition 4.5 (Acceptance). *A run* $G_0 \longrightarrow_{e_0} G_1 \longrightarrow_{e_1} \ldots$ *is* accepting *iff for all n with* $e \in \mathsf{In}(G_n) \cap \mathsf{Re}(G_n)$ *there exists* $m \geq n$ *s.t. either* $e_m = e$*, or* $e \notin \mathsf{In}(G_m)$*. A* trace is accepting *iff it has an underlying run which is.*

Acceptance tells us which workflows a DCR graph accepts, its *language*.

Definition 4.6 (Language). *The* language *of a DCR graph G is the set of its accepting traces. We write* $\mathsf{lang}(G)$ *for the language of G.*

We now know enough to formalise the first DCR graph we saw.

Example 4.7. The DCR graph of Fig. 1 and 2 has events a, s, r, b labelled Application deadline (a), Start round (s), Receive application (r), and Board meeting (b). It has relation R given by $\rightarrow\!\% = \{(a, r)\}$, $\rightarrow\!+ = \emptyset$, $\rightarrow\!\bullet = \{s, r\}$, $\bullet\!\rightarrow = \{r, b\}$ and $\rightarrow\!\diamond = \emptyset$. We can find a run of this DCR graph in Fig. 2: $B_1 \longrightarrow_s B_2 \longrightarrow_r B_3 \longrightarrow_r B_4 \longrightarrow_a B_5 \longrightarrow_b B_6$
Here, B_1, B_2 are accepting, whereas B_3–B_5 have b pending and so are not.

4.1 Composition & interfaces

Composition of DCR graphs was originally introduced in [10].

Definition 4.8 (Composition of DCR graphs). *The composition $G \mid H$ of DCR graphs G and H is defined by taking the union of all components. Formally:* $G \mid H = (\mathsf{E} \cup \mathsf{E}', \mathsf{R} \cup \mathsf{R}', (\mathsf{Ex} \cup \mathsf{Ex}', \mathsf{Re} \cup \mathsf{Re}', \mathsf{In} \cup \mathsf{In}'))$
The empty *or* zero *DCR graph,* $\mathbf{0}$, *is the unique DCR graph with no events.*

Composition does not in itself give "refinement" in the classical sense: in DCR graphs, even if G, H share events and labels, the language of $G \mid H$ might actually be *larger* than either G or H. The following definition helps narrow down what are "good" compositions.

Notation. The *projection* of a sequence σ to a set E is obtained by removing every element of σ not in E. For instance, the projection of $\sigma = AABCABC$ to $E = \{A, C\}$ is $\sigma|_E = AACAC$. We lift projection to sets of sequences pointwise.

Definition 4.9 (Refinement). *H* refines *G iff* $\big(\mathsf{lang}(G \mid H)\big)|_{\mathsf{E}(G)} \subseteq \mathsf{lang}(G)$.

To help establish refinements, we have the following theorem, which states that a DCR graph G is refined by a DCR-graph H if they have no shared event which may be included or excluded unilaterally by H.

Theorem 4.10. *H refines G if in H shared labels are associated only with shared events, and for all $f \in \mathsf{E}(G) \cap \mathsf{E}(H)$ and $e \in \mathsf{E}(H)$ we have that:*

1. *If $e \rightarrow\%_H f$ then also $e \rightarrow\%_G f$,*
2. *if $e \rightarrow+_H f$ then also $e \rightarrow+_G f$,*
3. $\mathsf{Ex}(H) \cap \mathsf{E}(G) \subseteq \mathsf{Ex}(G)$,
4. $\mathsf{In}(H) \cap \mathsf{E}(G) \subseteq \mathsf{In}(G)$.

Conditions (1) and (2) mean that H cannot unilaterally include or exclude shared events; conditions (3) and (4) that the marking of H does not change the inclusion- or execution-state of shared events.

Example 4.11. As an example, taking G to be the DCR graphs of Fig. 1 and H to be the one of Fig. 3, then both G, H fulfil the criteria of Theorem 4.10. Thus, we can be sure that when we compose them to obtain $G \mid H$ in Fig. 4, their local behaviour is preserved: the valid execution orders of Application deadline, Start round, Receive application, and Board meeting in Fig. 4 are all also valid according to Fig. 1.

4.2 Hi-DCR graphs

Towards DCR-graphs with sub-processes, we need first DCR graphs with interfaces, i-DCR graphs.

Definition 4.12 (i-DCR graph). *An* i-DCR graph *is a tuple* $G = (\mathsf{E}, \mathsf{R}, \mathsf{M}, \mathsf{I})$ *such that* $(\mathsf{E}, \mathsf{R}, \mathsf{M})$ *is a DCR graph and* $\mathsf{I} \subseteq \mathsf{E}$. *Events* $\mathsf{L} = \mathsf{E} \setminus \mathsf{I}$ *are* local *events. An i-DCR graph inherits notions of enabled events, execution, and zero from its underlying DCR-graph.*

We note that once we can speak of *execution*, we have using Definitions 4.4, 4.5, and 4.6 also definitions of transistions, runs, traces, acceptance, and language.

The point of the interface I is to allow us to choose which events should fuse with similar events in composition, and which should be considered private. To avoid fusing of private events, we must sometimes employ renamings.

Definition 4.13 (Freshness, compatibility). *If G, H are i-DCR graphs we say an that G is* fresh *for H iff $\mathsf{L}(G) \cap \mathsf{E}(H) = \emptyset$. We say that they are* compatible *iff they are both fresh for the other. We say that G, H are equivalent if they are structurally identical up to the choice of local events.*

The composition of i-DCR graphs guarantees that local events of compatible graphs do not overlap.

Definition 4.14 (i-DCR composition). *The composition $G \mid H$ of i-DCR graphs G, H is defined as for DCR graphs, taking interfaces of the combined graph to be $\mathsf{I}(G) \setminus \mathsf{L}(H) \cup \mathsf{I}(H) \setminus \mathsf{L}(G)$.*

For compatible i-DCR graphs, this definition is equivalent to taking simply $\mathsf{I} \cup \mathsf{I}'$.

Definition 4.15 (Hi-DCR). *A Hi-DCR graph is a tuple $G = (\mathsf{E}, \mathsf{R}, \mathsf{M}, \mathsf{I}, \mathsf{S})$ where S is a map taking events to Hi-DCR graphs and $(\mathsf{E}, \mathsf{R}, \mathsf{M}, \mathsf{I})$ is the* underlying *i-DCR graph $G|_\iota$ of G. An event e of G is* enabled *in G iff it is in $G|_\iota$.*

Note that if one wants an event e to *not* spawn any sub-process, one simply maps it to sub-process definition to zero, i.e., takes $\mathsf{S}(e) = \mathbf{0}$.

Definition 4.16 (Hi-DCR execution). *Suppose e is an event of the Hi-DCR graph G, that $e \in \mathsf{enabled}(G)$ and that $\mathsf{S}(e) = H$. Then to execute e in G:*

1. *Pick some H' equivalent to H but fresh for G.*
2. *Execute e in $G \mid H'$ (considered a DCR graph) to obtain H.*

That is, if $G \mid H' = (\mathsf{E}, \mathsf{R}, \mathsf{M}, \mathsf{I}, \mathsf{S})$, we execute e in $(\mathsf{E}, \mathsf{R}, \mathsf{M})$ obtaining $(\mathsf{E}, \mathsf{R}, \mathsf{M}')$, then declare the execution of e in $G \mid H'$ to be $(\mathsf{E}, \mathsf{R}, \mathsf{M}', \mathsf{I}, \mathsf{S})$.

Example 4.17. The notion of i-DCR graph and the definition of Hi-DCR graph execution explains formally why the event Approve decisions is *not* duplicated when a subprocess is spawned between Fig. 7 and 8: it is an interface event, and so is fused in the composition that happens when new sub-processes are spawned. The event Review report in the sub-process, on the other hand, *is* duplicated: It is local, and because execution of Hi-DCR graphs choose fresh names for local events during spawning, it is duplicated.

Theorem 4.18. *Hi-DCR graphs are strictly more expressive then DCR Graphs and therefore also strictly more expressive then ω-regular languages.*

Proof. Hi-DCR graphs conservatively extend DCR graphs, which are known to express exactly ω-regular languages [14]. But in Fig. 8, every time we execute Receive application we are will execute at least one Accept or Reject. This requires *counting* Receive application which is impossible for ω-regular languages.

5 Implementation

For experimentation, we have implemented a prototype tool for working with Hi-DCR graphs. This tool features a simulation engine capable of executing transitions, and of dynamic re-configuration using both unconstrained composition (Definition 4.8) and refinement (Definition 4.9). For finite-state graphs, the tool can do also basic model-checking tasks, such as finding a path to dead-lock, termination, acceptance, or some event being enabled. Whereas in the other sections of this paper, we represented DCR graphs graphically, as figures produced by the tool, the tool inputs a textual representation. As an example of that representation, consider again Fig. 6. Its equivalent textual representation is below.

```
"Other review"                          1
"Lawyer review"                         2
-->* "Review report"                    3
-->* ("Accept" "Reject")                4
-->% !"Decision"                        5
"Decision" -->* "Decision"              6
("Accept" "Reject") -->% "Decision"     7
("Accept" "Reject")                     8
 *--> "Update report"                   9
                                        10
/("Other review" "Lawyer review"        11
  "Review report"                       12
  "Accept" "Reject" "Decision")         13
```

All events are interface events by default; local events are specified by prefixing them with a slash '/' (line 11-13). Events can also be prefixed '+', '%', and '!', (line 5) indicating that they are initially included, excluded, respectively pending. For convenience, the language allows both chaining of events and relations (line 2–5) as well as relating multiple things (line 7).

The tool uses Graphviz [5] to automatically produce diagrams. The diagrams in the present paper were all so generated. The tool is implemented in F# and runs on the major platforms. The executable and source code can be found at [4].

6 Conclusion

In this paper we first demonstrated how DCR Graphs can be used for incremental, declarative design of processes, by introducing a notion of compositional refinement that guarantees language inclusion (with respect to the labels of events present in the original process) and thus preserves compliance for accepting executions. We then used these techniques to introduce Hi-DCR Graphs, a conservative extension of DCR graphs, which allows events to spawn sub-processes. The extensions have been presented and motivated using as an example an abstraction of a real-world case supplied by our industry partner, Exformatics A/S. We provided a formal semantics for Hi-DCR Graphs and demonstrated by example that they are more expressive then ω-regular languages. Finally we reported on a prototype implementation of Hi-DCR Graphs which supports a programming-like syntax, automatic visualisation, simulation, and rudimentary model-checking.

6.1 Future Work

While the notion of refinement introduced in the present paper preserves compliance for accepting executions, it may in fact introduce errors such as livelocks and deadlocks. The simplest example would be to refine a process with a DCR graph containing a single, included (local) event having itself as condition and being initiatlly required as response. In [15] it is shown how adaptations can be verified for safety and liveness, by relying on the map from DCR Graphs to Büchi-automata [16], which is further mapped to Promela and verified in the SPIN model-checker. However, as demonstrated in [15], this approach is not very efficient. We are therefore currently investigating techniques for more efficient verification of DCR Graphs in the presense of adaptations.

As was mentioned in the related work section the development of Hi-DCR Graphs brings us closer to the GSM notation, and we plan to use this work in the future as a basis for formal mappings between GSM and DCR Graphs models.

The question of the precise expressive power of Hi-DCR graph remains open. We conjecture that they are Turing-equivalent. This points to another relevant question, namely how to constrain Hi-DCR graphs to allow safety and liveness guarantees. An obvious possible constraint would be to bound the number times each sub-process can be spawned by a constant. Because of the formalization of spawning based on composition, it follows that this constrains the model to the expressiveness of standard DCR Graphs, that is, to Büchi-automata.

Formal expressive power aside, in practice many idiomatic constructs, like the "disjunctive responses" used in Fig. 8 could be formalised as derived constructs in their own right, potentially making them more accessible to end-users, much like DECLARE is formalised in terms of LTL. Similarly, other questions regarding the usability of the approach will be investigated in future studies through empirical investigations undertaken in cooperation with our industrial partners and end-users.

Acknowledgments. We gratefully acknowledge fruitful discussions with Rik Eshuis. The work is supported by grant VELUX 33295, 2014-2017 and the Danish Agency for Science, Technology and Innovation.

References

1. van der Aalst, W., Pesic, M., Schonenberg, H., Westergaard, M., Maggi, F.M.: Declare. Webpage (2010), `http://www.win.tue.nl/declare/`
2. van der Aalst, W.M., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer (2006)
3. Carbone, M., Hildebrandt, T.T., Perrone, G., Wasowski, A.: Refinement for transition systems with responses. In: FIT. EPTCS, vol. 87, pp. 48–55 (2012)
4. Debois, S.: DCR exploration tool v.6. IT University of Copenhagen (2014), `http://www.itu.dk/research/models/wiki/index.php/DCR_Exploration_Tool`
5. Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G.: Graphviz— open source graph drawing tools. In: Graph Drawing, LNCS, vol. 2265, pp. 483–484. Springer (2002), `http://dx.doi.org/10.1007/3-540-45848-4_57`

6. Groefsema, H., Bucur, D.: A survey of formal business process verification: From soundness to variability. In: Proceedings of the Third International Symposium on Business Modeling and Software Design. p. 198–203 (2013), `http://www.cs.rug.nl/ds/uploads/pubs/groefsema-bmsd.pdf`
7. Hildebrandt, T.T., Marquard, M., Mukkamala, R.R., Slaats, T.: Dynamic condition response graphs for trustworthy adaptive case management. In: OTM Workshops. LNCS, vol. 8186, pp. 166–171. Springer (2013)
8. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES. EPTCS, vol. 69, pp. 59–73 (2010)
9. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: FSEN. LNCS, vol. 7141, pp. 343–350. Springer (2011)
10. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: SEFM. LNCS, vol. 7041, pp. 237–252. Springer (2011)
11. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: WS-FM. LNCS, vol. 6551, pp. 1–24. Springer (2010)
12. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of ltl-based declarative process models. In: Khurshid, S., Sen, K. (eds.) RV. Lecture Notes in Computer Science, vol. 7186, pp. 131–146. Springer (2011)
13. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, Lecture Notes in Business Information Processing, vol. 56. Springer (2010)
14. Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (June 2012)
15. Mukkamala, R.R., Hildebrandt, T., Slaats, T.: Towards trustworthy adaptive case management with dynamic condition response graphs. In: EDOC. pp. 127–136. IEEE (2013)
16. Mukkamala, R.R., Hildebrandt, T.T.: From dynamic condition response structures to büchi automata. In: TASE. pp. 187–190. IEEE Computer Society (2010)
17. Mukkamala, R.R., Hildebrandt, T.T., Tøth, J.B.: The resultmaker online consultant: From declarative workflow management in practice to ltl. In: EDOCW. pp. 135–142. IEEE Computer Society (2008)
18. Object Management Group BPMN Technical Committee: Business Process Model and Notation, version 2.0, `http://www.omg.org/spec/BPMN/2.0/PDF`
19. Reijers, H., Mendling, J., Dijkman, R.: On the usefulness of subprocesses in business process models. BPM Reports 1003, Eindhoven (2010)
20. Slaats, T., Mukkamala, R.R., Hildebrandt, T.T., Marquard, M.: Exformatics declarative case management workflows as DCR graphs. In: BPM. LNCS, vol. 8094, pp. 339–354. Springer (2013)
21. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G.M. (eds.) Banff Higher Order Workshop. Lecture Notes in Computer Science, vol. 1043, pp. 238–266. Springer (1995)
22. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets. LNCS, vol. 255, pp. 325–392. Springer (1986)
23. Zugal, S., Soffer, P., Pinggera, J., Weber, B.: Expressiveness and understandability considerations of hierarchy in declarative business process models. In: BMMDS/EMMSAD. Lecture Notes in Business Information Processing, vol. 113, pp. 167–181. Springer (2012)

A Case for Declarative Process Modelling: Agile Development of a Grant Application System

Søren Debois, Thomas Hildebrandt and Tijs Slaats
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
Email: {debois,hilde,tslaats}@itu.dk

Morten Marquard and Tijs Slaats
Exformatics A/S
Copenhagen, Denmark
www.exformatics.com
Email: {mmq,ts}@exformatics.com

***Abstract*—We report on a recent industrial project carried out by Exformatics A/S in which the company used the declarative DCR Graphs notation to model and implement the grant application process of a Danish foundation. We present the process and discuss the advantages of the approach and challenges faced both while modelling and implementing the process. Finally, we discuss current work on extensions to the DCR Graphs notation aiming to address the challenges raised by the case study and to support the declarative, agile approach.**

I. Introduction

In the private as well as public sector, front-end customer services and their corresponding back-end processes and workflows are increasingly being digitalised by so-called Process Aware Information Systems (PAIS) [1]. The technology is pushed forward by promises of more usable, efficient and agile business processes, ensured to be compliant with business rules and the law.

Traditionally, functionality and user-experience of software systems have been described in a requirement specification *before implementation*. The requirements for functionality may be supported by semi-formal models such as e.g. UML sequence and activity diagrams and requirements for user-experience may be supported by use cases, user-interface descriptions and scenarios.

The new generation of PAIS are based on explicit, graphical and executable process models such BPMN [2], which supports a more agile development process, in which developers jointly with domain-experts *during implementation*, can design and simulate business processes and evaluate user-experiences.

However, the need for changes and adaptation to a PAIS does not end, when the system is being delivered to the customer. It is more the rule than the exception that the processes need to be adapted *after the system has been put into use*, either because the requirements where wrong or the needs changed before the final delivery. Indeed, often adaptations due to changes in business processes will be needed repeatedly throughout the entire lifetime of the system.

Moreover, it has been recognised in research, that imperative process descriptions such as BPMN that are based on explicit process flows tend to capture processes too rigidly, introducing un-justified dependencies and order between activities [1], [3], thereby increasing the need for adaptations to processes, even *during execution* of a process.

An explanation for this is, that the explicit flow graphs describe *how* a process is to be carried out, not *why*. By analogy to a way-finding service, it corresponds to describing a single route through a city, perhaps remembering to take a few possible exceptions into account. However, if the goal or map later changes, or the route is questioned regarding a potential improvement the flow-graph does not contain the information needed to derive a new route.

As an alternative, declarative process descriptions have been proposed by several research groups [3], [4], [5], [6], [7] as a way to achieve more flexible PAIS, by having process models that capture the *why* and not the *how*. The declarative approach has not seen wide-spread industrial adaptation yet and tool support has generally been limited to academic prototypes, which had led to the question if practitioners see actual opportunities to use declarative techniques. This has led to a study investigating what process modelling practitioners think of declarative modelling [8], which showed that the audience was receptive to the underlying concepts of the declarative modelling paradigm and the Declare [3], [4] and DCR Graphs notations, but also discovered that they had some difficulty getting used to the new paradigm and that the graphical notations used by the current notations were found to not always be intuitive, which made them relatively difficult to work with.

Exformatics A/S have during the past three years jointly with researchers at IT University of Copenhagen been developing, applying and implementing the declarative DCR graphs model [5], [6] in their standard Adaptive Case Mangement system. A key feature of DCR graphs is that they do not rely on a transformation before execution: The graphs describe both the initial state and the intermediate states, and thus support process adaptation during execution.

In the present paper we report on a recent industrial project in which Exformatics used the declarative DCR Graphs notation to model and implement the grant application process of a Danish foundation, the Dreyer foundation.

In Sec. II we first present the background for introducing a PAIS at the Dreyer foundation, and then in Sec. III we describe the key process(es) to be supported. In Sec. IV we first give the formal definition of DCR Graphs used in the Exformatics ECM Standard System, then we describe some of the sub processes of the grant application process in detail, showing how the models look like in the Exformatics process design tool, and finally we review the formal execution semantics of DCR Graphs. In Sec. V we briefly describe the Exformatics ECM system. We then discuss in Sec. VI the lessons learned during the implementation and discuss the advantages of the approach and challenges faced both while modelling and implementing the process. Finally, we conclude and discuss in VII current work on extensions to the DCR Graphs notation aiming to address the challenges raised by the case study and to support the declarative, agile approach.

II. The Grant Application Process

The Dreyer foundation handles applications from architects and lawyers in two annual application rounds. Previously, the foundation received the applications by mail and send them to an appropriate board member who is an expert in the domain of the application (that is, either architect or lawyer) for initial review. The number of applications is around 500-700 annually and the amount of paper being distributed is considerable. After the initial expert review, the applications are send to the other board members for review. This manual distribution and review process could easily take more than a month. After the reviews, a board meeting was held, typically lasting two long working days, to go through the applications and come up with a final recommendation. It was assumed that this process could be simplified by digitalising it. The aim is to be able to start voting earlier than today and enable the board to focus on the important applications at the board meeting, while simpler cases could be handled before the meeting as the members already had voted and agreed. The foundation had three major requirements for the solution:

1) Applications should be filled out by the applicants electronically in order to avoid lots of paper and facilitate easy distribution of the applications. The electronic form can be seen at http://formular.dreyersfond.dk/ (in Danish)
2) The board should be able to review and vote on the applications in an easy way.
3) The system should support the unique application case management process used by the foundation. The processes should be flexible and adaptive, in that it should be possible to handle exceptions within the system.

Exformatics responded to a request for proposal with its standard Adaptive Case Management tool which uses DCR graphs for supporting individual tailored adaptive processes. As mentioned in the second requirement above, it was a critical user experience requirement that board members could easily access, comment and vote on applications for a specific round. An idea of using traffic lights proposed by the foundation was prototyped, as shown in Fig. 1, demonstrating how the board members could easily view and change the status of each application.

III. The Process(es) to be Supported

Based on the task descriptions in the requirement specification and a few initial meetings with the foundation the following three processes were outlined as fundamental for supporting their case management process and requirements.

a) Round: Twice each year, the foundation has an application round. A round consists of a preparation, receiving, reviewing/voting, board meeting, payment, and a final closed *phase*. In each phase various events can happen. In preparation for a round, amounts are first reserved to be granted to architects and lawyers. When the round is initiated, applications are received. Applications have their own individual process life-cycle described below, including in particular the reviewing/voting done by four board members, whereof two acts as *experts*, one architect and one lawyer. After the application deadline, the decisions for all applications are finalised after a board meeting when the board meeting summary is approved. After this, applicants are informed, and each granted application enter the payment phase described in more detail below, until the last payment has been made.

b) Application: Each application is associated with a round and goes through a process of its own. After it is received an initial assessment is done by a case worker. If approved by the case worker, the application continues to an expert board member which does the first review, i.e. architect and lawyer applications are forwarded to the architect and lawyer board member respectively.

Once the expert board member has voted, the rest of the board can view the application and vote. The case worker can, provided the four board members agree, process the application before the board meeting. If so, the application is pre-approved or rejected. Other applications await the board meeting where all applications are voted for and decided by the board.

Once all applications are decided on, a summary of the board meeting is created and distributed among the board members. Once all approves the summary, the decision for all applications associated with the round is final, i.e. either approved or rejected, and the applicants are informed. If approved, a payment schedule and any need for further informations from the applicant are decided before the applicant is informed.

Once the supplementary information is received (if requested), a granted application continues to payment. By default, only a single payment is carried out. However, the case worker can decide to split the payment into several payments each with its own payment date. After delivery of the system, it was discovered, that if the applicants account number is modified during the application, the accountant must approve the modification before any payment can be done. As we will see, this could be handled by simple addition of a requirement to the process model and implementing a database trigger observing changes of account numbers.

Once all payments are completed, the application enters a feedback phase. In this phase, the applicant must send a summary of the project to the foundation. This information might be used in marketing material. Once the summary is received, the application is closed.

c) Payment: Each month the foundation collects the payments from each application that is ready to be paid out. The case worker review and approves the list, which is then transferred to Microsoft Dynamics (MD). The accountant is then asked to post the list and prepare and approve the payouts. After the accountant has done her work, the case worker is asked to approve the payouts. Once done the payouts are automatically transferred to the applicant through the integrated banking solution. Further, the case worker retrieves tax information about the applicants which is uploaded to the Danish tax authorities. Once all steps are done all payments are marked as completed. Any errors during the payout, e.g. invalid account numbers, are handled simply by manually removing the payment event from MD and modifying the account number.

IV. The DCR Graph Process Model

In this section, we first recall DCR graphs introduced in [5], [6] and further developed (e.g. by adding data) and implemented in the Exformatics ECM [9], [10], [11]. We then show how some of the sub processes in the Dreyer case management process described above where modelled as DCR Graphs.

Definition 1 (DCR Graph and DCR Graph with Data): A *DCR Graph* is a tuple $(\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ where

- E is a finite set of *events* (the nodes of the graph).
- l is a *labelling* function, assigning a label to each event
- R is a finite set of relations between events (the edges of the graph). The relations are partioned into five kinds, the *conditions* $(\rightarrow\bullet)$, *responses* $(\bullet\rightarrow)$, *milestones* $(\rightarrow\diamond)$, *inclusions* $(\rightarrow+)$, and *exclusions* $(\rightarrow\%)$.
- M is the *marking* of the graph. A marking represent the state of the process and is a triple $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ of sets of events, respectively the executed (Ex), the pending (Re), and the included (In) events.

A *DCR Graph with data* is a tuple $(\mathsf{E}, l, \mathsf{R}, \mathsf{M}_d, g, \mathsf{V})$ where V is a set of *data variables*, $\mathsf{M}_d = (\mathsf{M}, ev)$, $(\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ is a DCR Graph, ev is an evaluation function mapping data variables to values, and g is a (partial) *relation guard* mapping from (some of) the relations to boolean expressions over variables.

Events are atomic and may represent the beginning or end of an activity, e.g. ”start application round” or ”review done”. An event may also represent a decision being made (e.g. ”vote reject”) or indeed an atomic event, e.g. account number changed. In the ECM implementation, data variables of DCR Graphs with data are mapped to entries in the case database, and events can be triggered from the case management user-interface, by the system itself or by triggers set up in the database. The database may be changed by events externally to the model. This provides a very flexible implementation, but also somewhat limits what can be verified formally.

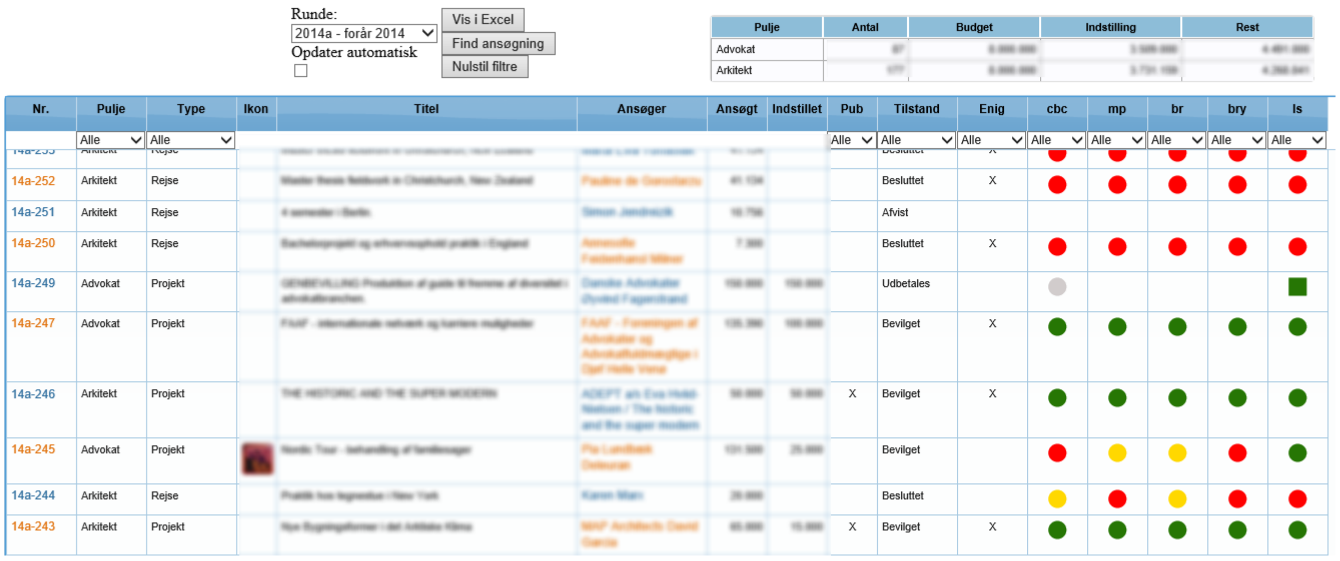
Before giving the formal semantics, we will give some example models of parts of the Dreyer process.

A. The review sub-process

As the name suggests, DCR graphs are *graphs*. They are represented visually in the DCRGraph editor [12] as in Fig. 2 illustrating the review sub process for an application.

Events (boxes) with labels (the text inside them) are related to other events by various arrows. If two events have the same label, the event ID is shown in italics in the bottom right corner. Otherwise, the editor chooses by default the event ID to be the same as the label and then it is not shown. In review process there are only four events, representing the four (submission of) review events by the four board members. The labels are Lawyer Review, Architect Review, Review, and Review. Note also that each box has an ”ear” with the role assignment.

Relations between the events govern their relative order of occurrence. When not constrained by any relations, events can happen in any order and any number of times. The review sub process only uses guarded condition relations. The guard expressions are referring to the variable R which indicates if



Pulje	Antal	Budget	Indstilling	Rest
Advokat	[redacted]	[redacted]	[redacted]	[redacted]
Arkitekt	[redacted]	[redacted]	[redacted]	[redacted]

Nr.	Pulje	Type	Ikon	Titel	Ansøger	Ansøgt	Indstillet	Pub	Tilstand	Enig	cbc	mp	br	bry	ls
	Alle	Alle						Alle	Alle	Alle	Alle	Alle	Alle	Alle	Alle
14a-252	Arkitekt	Rejse		[redacted]	[redacted]	[redacted]			Besluttet	X	🔴	🔴	🔴	🔴	🔴
14a-251	Arkitekt	Rejse		[redacted]	[redacted]	[redacted]			Afvist						
14a-250	Arkitekt	Rejse		[redacted]	[redacted]	[redacted]			Besluttet	X	🔴	🔴	🔴	🔴	🔴
14a-249	Advokat	Projekt		[redacted]	[redacted]	[redacted]	[redacted]		Udbetales		⚪				🟩
14a-247	Advokat	Projekt		[redacted]	[redacted]	[redacted]	[redacted]		Bevilget	X	🟢	🟢	🟢	🟢	🟢
14a-246	Arkitekt	Projekt		[redacted]	[redacted]	[redacted]	[redacted]	X	Bevilget	X	🟢	🟢	🟢	🟢	🟢
14a-245	Advokat	Projekt		[redacted]	[redacted]	[redacted]	[redacted]		Bevilget		🔴	🟡	🟡	🔴	🟢
14a-244	Arkitekt	Rejse		[redacted]	[redacted]	[redacted]			Besluttet		🟡	🔴	🟡	🔴	🔴
14a-243	Arkitekt	Projekt		[redacted]	[redacted]	[redacted]	[redacted]	X	Bevilget	X	🟢	🟢	🟢	🟢	🟢

Fig. 1. Board overview: Clicking on the round circles, grey, yellow, red or green, the board members can cast their vote and give a public and private comment.

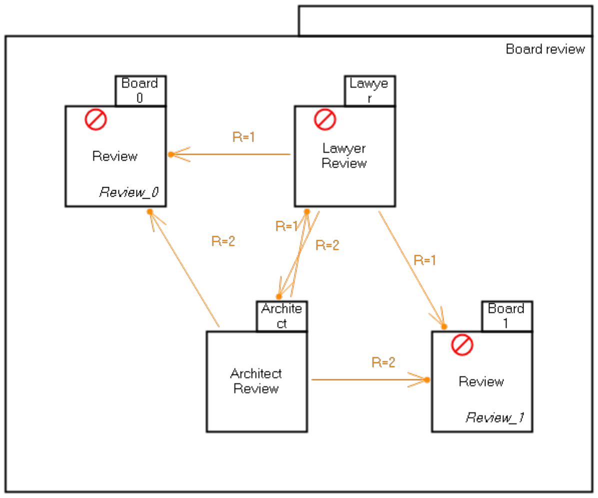


Fig. 2. A DCR graph for the review sub-process

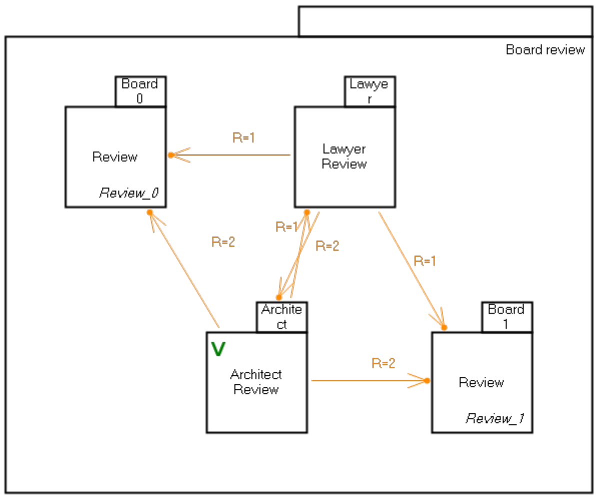


Fig. 3. A DCR graph for the review sub-process after the architect review

this is an application for Lawyer projects ($R = 1$) or Architect projects ($R = 2$). The meaning of the guarded relation is that it is only present if the guard is true. The figure shows the state if $R = 2$, and the "stop signs" indicate that Lawyer Review, Review, and Review are disabled because the guard on the condition from Lawyer Review, evaluates to true, while Architect Review is enabled because its constraint on the condition relation from Architect Review evaluates to false. The notation and semantics of the (unguarded) condition relation is similar to the precedence constraint in DECLARE [3], [4].

A key advantage of DCR graphs is that the graph *directly* represents the state of execution. There is no distinction between design-time and run-time. After executing the Architect review event, all events are possible. (Also Architect review, thereby allowing a new review to be submitted). The marking is visualised in the editor and simulation tool by adding a check mark to the Architect Review event indicating that it has been executed, as shown in Fig. 3.

B. *The payment sub-process*

A more complex example illustrating the use of all five kinds of relations is provided by the payment sub process in Fig. 7.

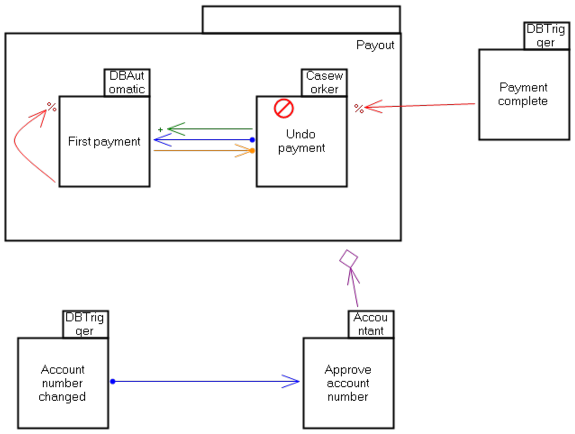


Fig. 4. A DCR graph for the payment sub-process

Changes of account numbers happen outside the case management system, but account numbers are registered in the database. By implementing a trigger that monitors account numbers in the database, any change can execute an event Account number changed in the case management process. Once this event is executed, the approve account number is made a required response by the (blue) response relation

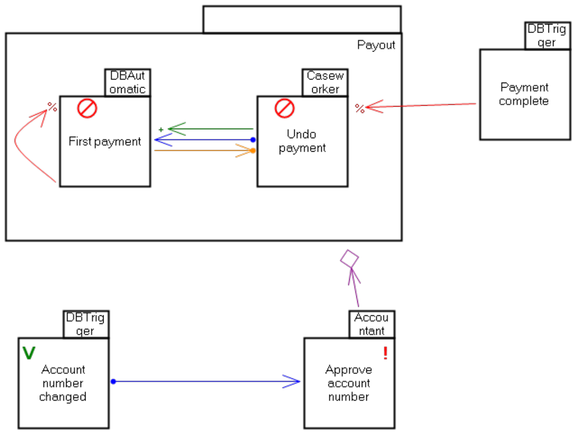


Fig. 5. A DCR graph for the payment sub-process after change of account number.

with a bullet at the tail. In the marking, this is recorded by adding Approve Account Number to the set Re of pending responses. In the editor, it is visualised graphically by adding a (red) exclamation mark to the event, as shown in Fig. 5. As the approve account number event is a milestone, indicated by the (purple) relation with diamond at the head, for the payment phase, any payment event will be blocked until the approve account number has been executed, e.g. completed by the accountant.

The event First payment has role DBAutomatic, which means that it is carried out by the system and not a use (technically, it is scheduled for a date). Once it is executed it excludes itself, since it is related by the (red) exclusion relation with a % at the end to itself. This in turns enables a new event Undo payment allowing the case worker to undo a payment. If the Undo payment is executed, it includes the payment again, due to the (green) inclusion relation with a + at the end, and makes it a required response, because of the (blue) response relation. Note that once the payment has been registered as completed (notified by the Payment complete database trigger, Undo payment is excluded and thus the payment can no longer be undone. The idea of this processes is, that if an error occurred in the payment (i.e. it is not completed) the error can be fixed by manually executing the Undo paymentevent, but once the payments is send to the bank without any errors the caseworker cannot redo the payment anymore by marking the payment as having an error.

C. Dynamic addition of sub processes

If the case worker adds one more payment to the application, the graph is dynamically modified resulting in the graph

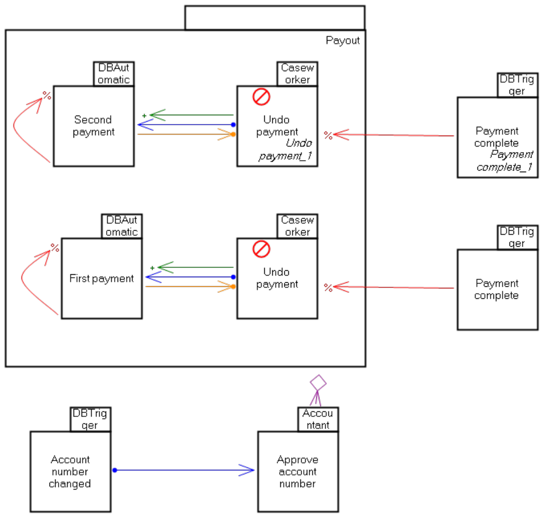


Fig. 6. A DCR graph for the payment sub-process with two payouts

shown in Fig. IV-C

Notice that adding a payment result in three new events, a pattern of events, which has the same structure as the initial payment event. The pattern is added dynamically to the DCR graph modifying the logic. Notice also that Approve account number is a milestone for all the nested events in the Payout phase. Doing so simplifies the logic as any change to account number will block any payment event.

D. Cross-process synchronization

In the example of payment sub-processes above, dynamic addition of processes was handled by directly modifying the graph at runtime, and there were no relations connected directly to the payment sub-processes.

We can also add sub processes by starting new instances of processes, and synchronise execution through database triggers (i.e. events with role DBtrigger) and auto-execution events (i.e. events with role DBautomatic).

Such synchronization exists allowing applications to be pre-approved before handled formally at the board meeting, if the board cast 4 equal votes, or the board simply informs the case worker to pre-approve the application and start payment. At a certain point, such pre-approval cannot happen anymore, but must await board meeting summary approval, i.e. approval from all four board members that the meeting summary has been approved. The round therefore has an event Block Further

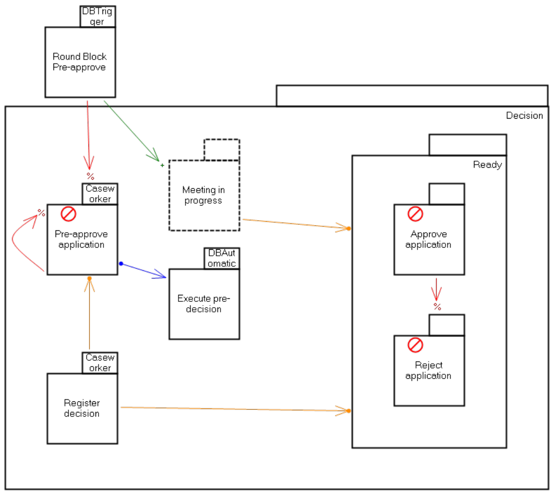


Fig. 7. A DCR graph for the payment sub-process

Decisions which once executed blocks the ability to pre-approve any applications. As the existing DCR model does not support cross-process events we have a database triggers which monitors this event. Once the Block Further Decision is executed the DBTrigger event Round block pre-approve is executed on all applications belonging to the round.

One this event is executed the event pre-approve application is excluded which means that the case worker cannot pre-approve the application anymore. Further the event Meeting in progress is included (green arrow). The reason for this event is to block the Ready nesting which contains the Approve and Reject application events. Using database triggers we can support events between processes. Each event acts as an interface for the process.

E. Formal Semantics

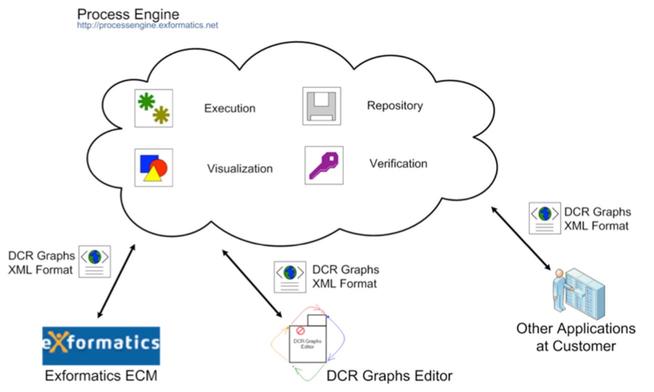
The formal execution semantics of DCR Graphs is given by notions of enabledness, transitions and acceptance. To keep the exposition simple we only give the formal semantics of plain DCR Graphs without data.

For a binary relation "$\rightarrow$" we write "$\rightarrow Z$" for the set $\{x \mid \exists z.\ x \rightarrow z\}$; similarly for "$Z \rightarrow$". We omit braces on singletons, writing $\rightarrow e$ rather than $\rightarrow \{e\}$.

Definition 2: Let $G = (\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ be a DCR Graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. We say that an event $e \in \mathsf{E}$ is *enabled* and write $e \in \mathsf{enabled}(G)$ iff (a) $e \in \mathsf{In}$ and (b) $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, and (c) $\mathsf{In} \cap (\rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

That is, enabled events (a) are included, (b) their included conditions already executed, and (c) have no included milestones with an unfulfilled responses.

Definition 3: Let $G = (\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ be a DCR Graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. If $e \in \mathsf{enabled}(G)$ we may *execute* e obtaining the DCR Graph $H = (\mathsf{E}, \mathsf{R}, (\mathsf{Ex} \cup e, \mathsf{Re} \setminus e \cup (e\bullet\!\!\rightarrow), \mathsf{In} \setminus (e\!\rightarrow\!\%) \cup (e\!\rightarrow\!+)))$. In this case we say that G has *transition on e to H* and write $G \xrightarrow{e} H$. A *run* of G is



sequence $G = G_0 \xrightarrow{e_0} \cdots$. A run is *accepting* iff for all n with $e \in \mathsf{In}(G_n) \cap \mathsf{Re}(G_n)$ there exists $m \geq n$ s.t. either $e_m = e$, or $e \notin \mathsf{In}(G_m)$.

V. The Exformatics ECM Standard System

Exformatics Adaptive Case Management (ACM) enables knowledge workers to handle structured as well as unstructured information along a process. Both unstructured information, such as emails, documents and small messages in activity streams similar to Yammer and Facebook, and structured information, such as process responsible, application amount, account number, applicant name, is registered on the case, i.e. within the context of the process. A case can support different processes, each defined by a DCR graph. Exformatics ACM leverages Microsoft SharePoint and SQL Servers for hosting documents, emails and data. The front-end is build using REST services and jQuery based web parts hosted in a SharePoint portal. Applications are received using a semantics based form where the foundation can configure columns and rules, such as a mandatory column, themselves. Applications can contain documents which are stored in SharePoint document libraries. Processes are described using DCR graphs, and are executed by the Exformatics Process Engine, running as a cloud based web service. For each event executed, the DCR graphs of the process is extracted and send to the process engine for executing, and the resulting DCR graph is received and imported into the ACM platform. Any exchange of DCR graph use a common DCR xml standard briefly described in [13].

Such a model prepares processes for cross-corporate integration as only a few processes are intra-company only. Further DCR graphs are prepared to be used by other ACM systems.

The events can be exposed at the user-interface using customized visualization, as exemplified by the overview screen shown in FIg.1. The vote events for each application is linked directly to the trafic light bullets.

VI. Lessons Learned

In the following we describe some of the key lessons learned during the implementation of the adaptive case management system for the Dreyer foundation.

A. On-the-fly Process adaptation and creation of sub processes

Since the run-time-state is simply a marking of the graph, processes can be modified on-the-fly simply by changing the DCR graph. We benefited from this to allow for payment sub-processes to be dynamically added. Initially, support for one payment is included, which consist of the pattern of three events described above and shown in Fig. IV-C, which can deal with payment errors, e.g. due to errors in Microsoft Dynamics or the banking integration. End-users can, while the process is being executed, add more payments, which triggers adding a pattern of events as outlined above.

Since this is an expected adaptation, it could be have been good to have a way to model in the graph that sub-processes are dynamically added when an event, e.g. Add payment, is executed. Simliarly, each individual applications can be considered a sub-process of a round.

As the current DCR model does not support sub-processes, we had to implement synchronization between the round and the set of applications using database triggers. An example database trigger can be the Round Approved, which occurs on the application once the Round has been approved, which happens once the board meeting summary has been approved. Using a database trigger we fire this event for all applications of a given round once the round is approved. Creating such database triggers are not easy as we have to handle the concept of a sub-process outside the DCR logic. The ability to describe relations between a process and its subprocesses as formalized in the submitted paper [14] simplifies this logic, and the ability to describe a sub-process within the context of a process would ease the description and implementation.

A possible challenge when dealing with sub-processes identified in the present case study, is how to handle and formalize moving an application from one round to another. This sometimes happens if the board cannot decide what to do and postpone decision to the next board meeting. This would likely require formalizing passing identifies of events between processes, similar to the passing of names in the π-calculus.

This is not possible in the version of DCR Graphs implemented in the Exformatics tool. However, the theoretical extension of dynamic creation of sub processes called *Hierarchical DCR Graphs* has been described in a recently submitted paper [14] and implemented in a prototype, web-based simulation tool for DCR Graphs to be found at [15].

To handle *un-expected* additions to the process, we need support for end-users to define such patterns and add them to a running process, which in the simplest case would ammount to adding (or removing) a single event or relations to a graph. As described in [13], such run-time adaptations could in principle benefit from the formal verification supported by DCR Graphs, guaranteeing that adaptations are robust, e.g. they do not introduce deadlocks, livelocks or other violations of specified properties. However, in practice this would require better verification techniques than what is currently implemented. Also, if extended with dynamic creation of sub processes, DCR Graphs becomes Turing-expressive [16] which in makes a general verification covering all possible processes impossible.

B. Iterative, agile process design

Defining and configuring the processes in DCR graphs was done in several phases where more and more details were added. During this process the process, e.g. the application, could be simulated in the DCR drawing tool, as well as in the system itself. This helped end-user understand the process and facilitated discussions on how the process should work and not work.

Despite a detailed analysis and many end user reviews, it (of course) turned out, that a requirement was still lacking in the designed and implemented process after delivery. The requirement that any modification to an account number requires accountant approval before payouts can be done was simply missing. Adding this requirement to the graph turned was very easy, as we just needed to adjust the DCR graph by adding the database trigger event event Account changed with a response relation to the approve account event, add logic to the customer object in the database to monitor changes to account numbers, and firing the Account changed if this occurs. Database triggers are simple to create and understand for such requirements and avoiding having the treatment of account numbers in the graph itself.

C. Understandability

DCR graphs as supported in the current editor are hard to read and edit. Support for compositional models and zooming in and out in details are needed for end users. Also, better descriptions of the DCR primitives and a modelling methodology is needed. The condition relation is easily understood, and the required response relation is also explainable. For the present case, milestones, inclusion and exclusions and also the nesting of events in phases were experienced as hard to explain and understand. Some other graphical representation for many-to-many relations than the nesting is likely needed. We also believe that it would be very useful if the system supported typical (happy) paths through the process, analalogous to a navigation system suggesting possible routes to a driver. The system should be able of adjusting the proposed paths to changing needs of the user, similar to how a navigation system can adapt to when a driver wants to make a detour to a gas station.

VII. Conclusion and Future Work

We have described an industrial project delivering an adaptive case management solution based on the Exformatics standard system and processes described as DCR Graphs.

DCR graphs have been a very valuable component to ensure Exformatics ACM can support the unique processes required by the foundation as well as run-time modifications to the processes as outlined in the payment event pattern. Rather than creating custom code the DCR graph describe the business rules and dependencies which enables us to support unique business requirements using a standard solution. As a product company this is important, as managing several code bases with unique customer requirements is too expensive.

The entire delivery was made in less than 4 months, from the first presentation of the requirement specification to the system as described in the present paper. The successful delivery demonstrated some of the claimed benefits of DCR Graphs in practice, in particular: 1) They support agile, rule based modelling that can be easily adapted when new requirements are discovered during and after delivery, 2) they support run-time addition of sub-processes which can be utilised for e.g. customised payment schemes, and 3) the resulting processes are allow maximal flexibility at run-time since they do not constrain the flow unnecessary - if some constraints are found unnecessary, they can simply be removed. The case study also demonstrated areas for future work and improvement, in particular: 1) support for dynamic creation of sub-processes should be part of the formal model, 2) if sub-processes are included in the model, formal verification of processes would me more likely to be possible, but we need to deal with the fact that the model becomes Turing-expressive, 3) the modelling notation, its graphical visualisation and design tools need to be improved to make the models more comprehensible and ultimately allow the business users to adapt processes themselves in a robust way. Enabling business users to take ownership of their processes, enables businesses to develop without changing the IT systems supporting the processes. As business processes evolves all the time we need to support such changes. Having to await a new product version from a vendor often does not suit the business requirement. Customer development does support the requirements either, as their require IT people for development and maintenance, is very time consuming and expensive.

Acknowledgment

This work was supported in part by the Computational Artifacts project (VELUX 33295, 2014-2017), by the Danish Agency for Science, Technology and Innovation, by Exformatics A/S, and by the IT University of Copenhagen. The authors would like to thank VisualIT for their contribution to the initial requirement elucidation.

References

[1] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[2] Object Management Group BPMN Technical Committee, "Business Process Model and Notation, version 2.0," http://www.omg.org/spec/BPMN/2.0/PDF.

[3] W. M. van der Aalst and M. Pesic, "DecSerFlow: Towards a truly declarative service flow language," in *WS-FM 2006*, ser. LNCS, vol. 4184. Springer, 2006, pp. 1–23.

[4] W. van der Aalst, M. Pesic, H. Schonenberg, M. Westergaard, and F. M. Maggi, "Declare," Webpage, 2010, http://www.win.tue.nl/declare/.

[5] T. T. Hildebrandt and R. R. Mukkamala, "Declarative event-based workflow as distributed dynamic condition response graphs," in *PLACES*, ser. EPTCS, vol. 69, 2010, pp. 59–73.

[6] R. R. Mukkamala, "A formal model for declarative workflows: Dynamic condition response graphs," Ph.D. dissertation, IT University of Copenhagen, June 2012.

[7] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín, "Introducing the guard-stage-milestone approach for specifying business entity lifecycles," in *WS-FM*, ser. LNCS, vol. 6551. Springer, 2010, pp. 1–24.

[8] H. A. Reijers, T. Slaats, and C. Stahl, "Declarative modeling-an academic dream or the future for bpm?" in *Proceedings of 11th International Conference on Business Process Management (BPM 2013)*, 2013, pp. 307–322.

[9] T. T. Hildebrandt, R. R. Mukkamala, and T. Slaats, "Safe distribution of declarative processes," in *SEFM*, ser. LNCS, vol. 7041. Springer, 2011, pp. 237–252.

[10] T. Slaats, R. R. Mukkamala, T. T. Hildebrandt, and M. Marquard, "Exformatics declarative case management workflows as DCR graphs," in *BPM*, ser. LNCS, vol. 8094. Springer, 2013, pp. 339–354.

[11] T. T. Hildebrandt, M. Marquard, R. R. Mukkamala, and T. Slaats, "Dynamic condition response graphs for trustworthy adaptive case management," in *OTM Workshops*, ser. LNCS, vol. 8186. Springer, 2013, pp. 166–171.

[12] T. Slaats, "DCR graph editor," 2013, http://www.itu.dk/research/models/wiki/index.php/DCR_Graphs_Editor.

[13] R. R. Mukkamala, T. Hildebrandt, and T. Slaats, "Towards trustworthy adaptive case management with dynamic condition response graphs," in *EDOC*. IEEE, 2013, pp. 127–136.

[14] S. Debois, T. Hildebrandt, and T. Slaats, "Hierarchical declarative modelling with refinement and sub-processes," *Submitted for publication*, 2014.

[15] S. Debois, "DCR exploration tool v.6," *IT University of Copenhagen*, 2014, http://www.itu.dk/research/models/wiki/index.php/DCR_Exploration_Tool.

[16] S. Debois, T. Hildebrandt, and T. Slaats, "Dynamic conditions, restless events and reproduction: ω-regular languages and beyond," *Submitted for publication*, 2014.

5 Dynamic Condition Response Graphs with Time and Data

Contracts for Cross-organizational Workflows as Timed Dynamic Condition Response Graphs

Thomas Hildebrandt[a,1], Raghava Rao Mukkamala[a], Tijs Slaats[a,b], Francesco Zanitti[a]

[a]*IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark*
[b]*Exformatics A/S, 2100 Copenhagen, Denmark*

Abstract

We conservatively extend the declarative Dynamic Condition Response (DCR) Graph process model, introduced in the PhD thesis of the second author, to allow for discrete time deadlines. We prove that safety and liveness properties can be verified by mapping finite timed DCR Graphs to finite state transition systems. We exemplify how deadlines can introduce time-locks and deadlocks and violate liveness. We then prove that the general technique for safe distribution of DCR Graphs provided in previous work can be extended to timed DCR Graphs. We exemplify the use of timed DCR Graphs and the distribution technique in praxis on a timed extension of a cross-organizational case management process arising from a previous case study. The example shows how a timed DCR Graph can be used to describe the global contract for a timed workflow process involving several organizations, which can then be distributed as a network of communicating timed DCR Graphs describing the local contract for each organization.

Key words: Time, Contracts, Declarative process models, Cross-organizational workflow, Safety, Liveness

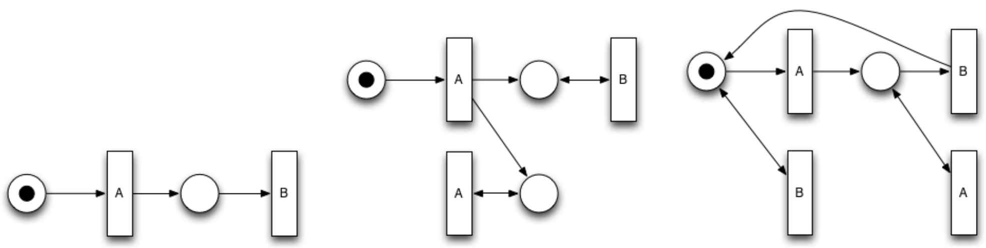
Email addresses: hilde@itu.dk (Thomas Hildebrandt), rao@itu.dk (Raghava Rao Mukkamala), tslaats@itu.dk (Tijs Slaats), frza@itu.dk (Francesco Zanitti)

[1]Authors listed alphabetically. This research is supported by the Danish Research Agency through the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu), an Industrial PhD grant, and the Genie Project (grant # 2106-080046)

1. Introduction

The idea of constructing process-aware information systems [1] to support correct execution and analysis of workflows based on explicit models of the processes dates back to the work on office-automation [2, 3, 4, 5] in the late 70ties. The early work was influenced by the Petri Net process model [6], which has also heavily influenced subsequent process modeling standards such as UML activity diagrams [7, 8] and BPMN [9] and the work on formal semantics and analysis of workflow and business processes (e.g. [10, 11, 12, 13]).

The heavy influence of the Petri Net model is quite natural. It is the first process model explicitly representing the concurrent execution of activities, it has a formal semantics supporting analysis and verification, and it has an intuitive graphical notation. To briefly recall, a Petri Net is a directed graph with nodes alternating between *transitions* and *places*, and a *marking* assigning zero or more *tokens* to each place. A transition is *enabled* if all places connected to it by an incoming edge is marked by at least one token. If an enabled transition is *fired*, one token from each place connected by an incoming edge is removed and one token is added to each place connected by an outgoing edge. Firing a transition thus represents the execution of an activity and the tokens represent resources needed to execute activities.



(a) First A, then B (b) A is a condition for B (c) B is a response to A

Figure 1: Condition and response relations as Petri Nets

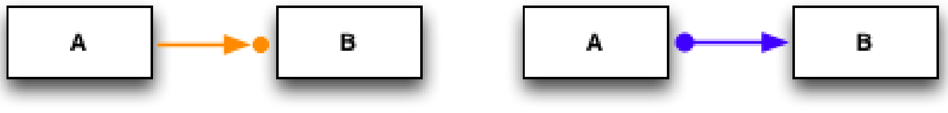
However, the notion of places and tokens in the Petri Net model is intrinsically linear and imperative. It describes a way to implement dependencies between activities where the ability to redo an activity must be explicitly modeled. This has been recognized, in particular in the incarnation of BPMN, to increase the risk of over-specification and be best suited for processes that follow a strict flow of control [14, 15]. As an example consider the property *activity A is a condition for activity B*, i.e. an execution of B must be preceded by an execution of A in the past. The Petri Net in Fig. 1(a) is likely to be the first choice of representation, but it really describes the more rigid implementation that activity A can be executed first (and only once) and then activity B can be executed (and only once). The Petri Net in Fig. 1(b) describes the general property, if all markings are allowed as accepting (terminating). The dual property, that *activity B is a response to activity A*, i.e. an execution of A must be followed by an execution of B at some point in the future, is also implemented by the Petri Net in Fig. 1(a). However, in order to allow all possible executions one needs a less restrictive Petri Net as the one in Fig. 1(c) with the initial marking being the only accepting marking.

As an alternative, declarative models such as temporal logics like LTL [16] or CTL [17] can be employed in order to maintain more flexibility of the execution and provide a specification of the dependencies between activities which abstracts from any particular implementation. This can in particular be used as a *contract* for a subsequent implementation or compliance verification [18]. Temporal logics are however in general considered to be difficult to understand by end-users and typically proposed to be replaced by *patterns* of properties, such as the condition and response patterns considered in Fig. 1(b) and Fig. 1(c) [19, 20, 18]. However, before the contract can be checked for a workflow process it is usually required that the contract is rewritten either to a particular (normal) form [21] or translated to an (imperative) automaton [20, 22, 23], which is difficult to understand and relate to the original contract.

This motivates finding a declarative process model which both can be understood by end-users of workflow management systems, e.g. by capturing patterns in a direct way, and supports verification and monitoring without rewriting the process. Dynamic Condition Response (DCR) Graphs [24, 25, 26, 27] is a candidate for such a declarative process language developed in the PhD project [28] of the second author as part of the Trustworthy Pervasive Healthcare Services (TrustCare) [29] research project.

From a practical perspective, DCR Graph model generalizes and formalizes the core primitives of the declarative Process Matrix model [30], developed, patented and used successfully for more than a decade by Resultmaker [31], the industrial partner of the project. The goal of the formalization was to provide the basis for formal verification and safe distributed execution of flexible, cross-organizational workflow processes as found in the healthcare sector.

From a technical perspective, a DCR Graph is a directed graph described by a 9-tuple $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$. The nodes of the graph are given by the set E of *events*. The event represents that some activity, as indicated by the labeling function $l : \mathsf{E} \rightarrow \mathcal{P}(\mathsf{L})$, happens in the workflow. The edges of the



(a) DCR Graph condition relation (b) DCR Graph response relation

Figure 2: Condition and response relations in DCR Graphs

graph are given by five relations: The *condition* ($\rightarrow\!\bullet$), *response* ($\bullet\!\rightarrow$), *milestone* ($\rightarrow\!\diamond$), *include*($\rightarrow\!+$), and *exclude* ($\rightarrow\!\%$) relation respectively. The condition and response relations thus directly captures the condition and response patterns as shown by the DCR Graphs in Fig.1 . Before explaining the milestone relation, it is helpful to take a look the 2nd element of the DCR Graph tuple, which is the *marking* M. The marking consists of a triple of sets of events $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ representing the state of the process. The set $\mathsf{Ex} \subseteq \mathsf{E}$ of *executed* events records which events have been executed in the past. The set $\mathsf{Re} \subseteq \mathsf{E}$ of *response* events records which events are required to be executed (i.e. responses) in the future in order for the entire execution to be accepting. The milestone relation means that an event can not execute while another event is pending response, it is particularly useful for descibing that a process can not continue on to a new phase while a previous phase hasn't been completed. Finally, the set $\mathsf{In} \subseteq \mathsf{E}$ of *included* events records the events which are currently included. If $e \rightarrow\!\% \; e'$ then e' is removed from the set In when e is executed, and if $e \rightarrow\!+ \; e'$ then e' is added to the set In when e is executed. This notion of dynamic inclusion and exclusion of events is the key new ingredient of DCR Graphs compared to other declarative approaches. Only currently included events in Re are required in order for the execution to be accepting, and only included condition events e' of an event e need to be in the set Ex in order for the event e to be enabled.

The markings can also be seen as the state of a (kind of Kripke) logical program given by the DCR Graph [32]. Then Ex is the set of basic facts that has been proven, Re is what has to be proven (possibly again) or excluded from the world to complete the proof, and In is the set of facts that are relevant in the current world. The markings also form the states of a Büchi-automata representation of DCR Graphs [25], which allows verification of safety and liveness properties using the SPIN model checking tool [28, 33].

The main new contribution of the present paper is to conservatively extend the DCR Graph model to allow for (discrete) time deadlines, while preserving the

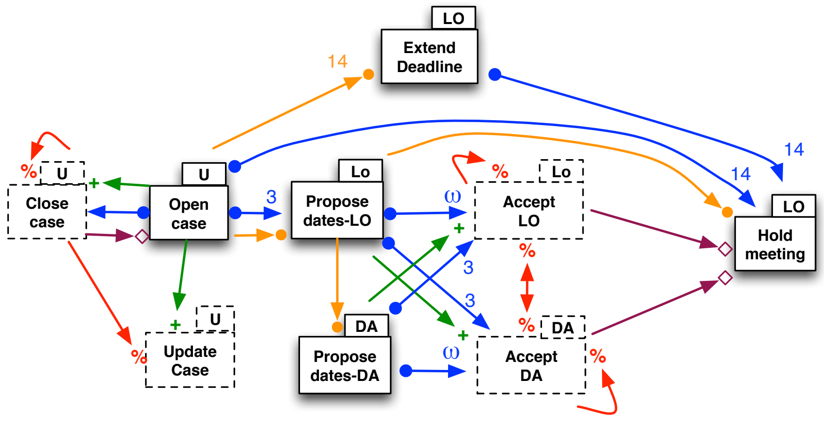


Figure 3: A timed DCR graph describing the global contract for a cross-organizational case management system.

simple operational semantics and that safety and liveness properties of the models can be formally verified by mapping the graphs to finite state Büchi-automata. Fig. 3 shows an example of a timed DCR Graph which we will use as example in the paper. The graph extends an untimed DCR Graph from a case study described in [34] which captures the *global* requirements of a distributed, cross-organizational case management system being developed by Exformatics A/S. The system spans three different types of organizations. The first is the *unions* for employees in Denmark, which creates, closes and updates cases on behalf of their members (represented by the events labelled with the U and respectively Open case, Close case and Update Case. The two other organizations are respectively the umbrella organization for the unions (LandsOrganisationen, LO) and the umbrella organization for the employers (Dansk Arbejdsgiverforening, DA). When a case is created by a union, LO and DA must agree on a date for a meeting and hold the meeting within 14 days from when the case was created. The deadline is expressed by the number 14 attached to the response relation from Open case to Hold meeting. If the meeting is not ready to be held after 14 days however, the deadline can be extended by LO by executing the event Extend Deadline. However, this event only becomes enabled 14 days after the case has been created, which is represented by the number 14 attached to the condition relation from Open case and Extend Deadline.

As described in [27], the DCR Graphs model admits a very general technique for distributing a graph describing a global contract for e.g. a cross-organizational process, as a network of synchronously communicating graphs describing the contracts for each local organization, and which combined describe the same process as the global contract. Another new contribution of the present paper is to extend this technique to timed DCR Graphs and provide a detailed proof of the correctness of the distribution technique. It is the first step of the further development of the DCR Graph model and its use for specification and safe execution of distributed, cross-organizational workflow processes to be carried out in the recently initiated industrial PhD project of the third author.

The rest of the paper is structured as follows. In Sec. 2 below we briefly survey related work. In Sec. 3 we first recall the formal definition of DCR Graphs and their semantics, and then proceed to define the extension to timed DCR Graphs. We then in Sec. 4 define and exemplify timed notions of safety and liveness for timed DCR Graphs. In Sec. 5 we exemplify and provide the technique for distributing a global timed DCR Graph, extending the technique for un-timed DCR Graphs given in [27]. Finally we conclude and comment on future work in Sec. 6.

2. Related Work

The DCR Graph model and graphical representation is similar to the Declare workflow language [35, 20]. Indeed the graphical representation of events and condition and response relations are the same in DCR Graphs and Declare. However, the two approaches differ in a crucial way. While the Declare model includes a fairly rich set of patterns as relations between events based on the patterns identified in [19], the DCR Graph model in addition to the condition and response relations only allows the milestone relation and the dynamic inclusion and exclusion relations. Still, the DCR Graphs model is more expressive: It allows to express all ω-regular languages [28], while Declare only allow expressing properties covered by finitary LTL. Moreover, Declare models are translated to either LTL or a new notion of colored automata [36] before they are verified or executed. The DCR Graph model directly supports execution of the process model based on the simple notion of markings of the graph.

There are many researchers [10, 11, 37, 12, 13, 38, 39] who have explicitly focussed on the problem of verifying the correctness of inter-organizational workflows based on variants of Petri nets. In [10], message sequence charts are used to model the interaction between local workflows modeled as Petri nets. In [11] Kindler et. al. follows a similar approach, using a set of scenarios given as sequence diagrams to specify the interactions. Criteria of local soundness guarantee the global soundness of an inter-organizational workflow. In [37], so-called *Query Nets* based on predicate/transition Petri nets are used to guarantee global termination. Besides the use of Petri nets, a key difference between the present paper and the work in [10, 11, 37] is that while we take a top down approach and *synthesize* models of the local workflows from a model of the global workflow, the latter work take a bottom up approach, assuming that the models of the local workflows are given.

A top down approach is taken in [13, 38], where a shared public view of an inter-organizational workflow given as a workflow net is partioned among the participating entities. A notion of projection inheritance is used to generate a private view that is a subclass to the relevant public view and guarantee deadlock and live-lock freedom. A more liberal and weaker notion than projection inheritance is used in [12] to guarantee weak termination in multiparty contracts based on open nets.

Modeling global behavior as a set of conversations among participating services has been studied by many researchers [40, 41, 42, 43, 44, 45] in the area business processes. An approach based on guarded automata studied in [40], for the realizability analysis of conversation protocols, whereas the authors in [41] used colored petri nets to capture the complex conversations. Similarly, but using process calculus to model service contracts, Bravetti-Zavattaro proposed conformance notion for service composition in [44] and further enhanced their correctness criteria in [45] by the notion of strong service compliance. The synthesis of local components from a global model has also been researched for process calculus formalizations of the imperative choreography language WS-CDL in the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [46]. To put it briefly, the work formalizes the core of WS-CDL as the global process calculus and defines a formal theory of end-point projections projecting the global process calculus to abstract descriptions of the behavior of each of the local ”end-points” given as pi-calculus processes typed with session types.

Also researchers [47, 48, 49, 50] in the web services community have been working on web service composition and decentralized process execution using BPEL [51] and other related technologies to model web services. A technique to partition a composite web service using program analysis is studied in [48] and [49] explore decomposition of a business process modeled in BPEL. Using a formal approach based on I/O automata representing the services, the authors in [50] study the problem of synthesizing a decentralized choreography strategy, that will have optimal overhead of service composition in terms of costs associated with each interaction. In [52, 53, 54] foundational work is presented on synthesizing distributed transition systems from global specification for the models of synchronous product and asynchronous automata[55]. In [54] structural and behavioral characterizations of the synthesis problem for synchronous and loosely cooperating communication systems are given, based on three different notions of equivalence: state space, language and bisimulation equivalence. Further Castellani et. al. [52] characterizes when an an arbitrary transition system is isomorphic to its product transition systems with a specified distribution of actions and they have shown that for finite state specifications, a finite state distributed implementation can be synthesized. Complexity results for distributed synthesis problems for the three notions of equivalences were studied in [53].

A methodology for deriving process descriptions from a business contract formalized in a formal contract language is studied in [56], while in [57] is proposed an approach to extract a distributed process model from collaborative business process. In [58, 47] is proposed a technique for flexible decentralization of a process specification with necessary synchronization between the processing entities using dependency tables, whereas the authors in [59] present a framework for optimizing the physical distribution of workflow schemas based on families of communicating flow charts.

Common to all the approaches discussed above are that they are confined to imperative process models such as Petri nets, workflow/open nets and automata. To the best of our knowledge, there exists very few works [60, 61] that have studied the synthesis problem in declarative modeling languages and none where both the global and local processes are given declaratively. Fahland [60] studies top down synthesis of Petri Net workflows from a limited subset of the declarative Declare/DecSerFlow [20] model, while Montali [61] studies the bottom-up composition of Declare [35] models with respect to conformance with a given choreography.

None of the work above study synthesis or conformance of cross-organizational workflow with time constraints. [62] provides a good overview of various temporal logics that have been used for specification of real-time systems. Similar to the extension to DCR Graph proposed in the present paper, Time Petri nets [63] extend regular Petri nets by allowing transitions to be labelled with time bounds $0 \leq a \leq b \wedge a \neq \infty$, such that a transition can only fire after a delay of a time steps after it has been enabled last and must either fire or be disabled before b time has passed. Time Petri nets have been studied as a formalism to model and verify time dependent concurrent systems in [64]. In [65, 66] the properties of reachability, boundedness and liveness for time Petri nets are studied, while [67] defines the semantics of time Petri nets in terms of timed automata. Time Workflow nets, a subclass of time Petri nets, have been introduced in [68] as a method for modeling time management in workflow processes. Timed Petri nets [69] are a variation of Petri nets where a (rational) time duration is assigned to every transition and transitions are required to fire as soon as they become enabled. Timed Petri nets are mainly used for performance evaluation [70]. Timed-arc Petri nets [71, 72] are another variation of Petri nets where tokens are annotated with an age and arcs from places to transitions are labelled with a time interval. The tokens that a transition can consume are then limited to those whose age falls within these time intervals. For Timed-arc Petri nets there is no notion of urgency. Transitions are neither required to fire when adequate tokens are available nor when those tokens are about to expire.

3. Timed Dynamic Condition Response Graphs

We employ the following notations in the rest of the paper.
Notation: We write ∞ for the set of finite ordinals and the least infinite ordinal ω. For an ordinal $k \in \infty$ we write $[k]$ for the (possibly infinite) set $\{i \mid 0 \leq i < k\}$. For a set E we write $\mathcal{P}(E)$ for the power set of E (i.e. set of all subsets of E). For a binary relation $\rightarrow \subseteq E \times E$ and a subset $\xi \subseteq E$ of E we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{e \in E \mid (\exists e' \in \xi \mid e \rightarrow e')\}$ and the set $\{e \in E \mid (\exists e' \in \xi \mid e' \rightarrow e)\}$ respectively, and abuse notation writing $\rightarrow e$ and $e \rightarrow$ for $\rightarrow \{e\}$ and $\rightarrow \{e\}$ respectively when $e \in E$.

We first recall in Def. 3.1 the formal definition of DCR Graphs and their semantics.

Definition 3.1. *A Dynamic Condition Response Graph (DCR Graph) G is a tuple* $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where*

(i) E *is a set of* events *(or activities),*

(ii) $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \in \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$ *is the* marking

(iii) $\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\% \subseteq \mathsf{E} \times \mathsf{E}$ *is the* condition, response, milestone, include *and* exclude *relation respectively.*

(iv) L *is the set of* labels *and* $l : \mathsf{E} \rightarrow \mathcal{P}(\mathsf{L})$ *is a labeling function mapping events to sets of labels.*

We define that an event $e \in \mathsf{E}$ *is* enabled*, written* $\mathsf{M} \vdash_G e$*, if* $e \in \mathsf{In} \land (\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$ *and* $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.
Finally, we define the result of executing an event e *as* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e =_{def} \big(\mathsf{Ex} \cup \{e\}, (\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\rightarrow, (\mathsf{In} \setminus e\!\rightarrow\!\%) \cup e\!\rightarrow\!+\big)$.

Intuitively, the marking of a DCR Graph records the state. An event is then e is enabled in a marking, if it is included and every event which is a condition for e is either excluded or executed or, and every event which is a milestone for e is either excluded or not required as a response, i.e. it is in a completed state. Below we give a simple example of a DCR Graph and execution of an event based on the graph in Fig. 3.

Example 3.1. *As an example consider the underlying (untimed) DCR Graph G of the timed DCR Graph G_t in Fig. 3: the initial marking of G is* $\mathsf{M} = (\emptyset, \emptyset, \mathsf{E} \setminus \{\textsf{\textit{Accept LO}}, \textsf{\textit{Accept DA}}, \textsf{\textit{Close Case}}, \textsf{\textit{Update Case}}\})$. *The only enabled event is* ***Open case****, all other events are either blocked through the condition relation, or not in the set of included events. After executing* ***Open case****, the new marking* M' *of G becomes:* $\mathsf{M}' = \mathsf{M} \oplus_G \textsf{\textit{Open case}} = (\{\textsf{\textit{Open case}}\}, \{\textsf{\textit{Propose dates-LO}}, \textsf{\textit{Hold meeting}}, \textsf{\textit{Close Case}}\}, \mathsf{E} \setminus \{\textsf{\textit{Accept LO}}, \textsf{\textit{Accept DA}}\})$. ***Open case*** *is added to the set of executed events,* ***Close Case, Propose dates-LO*** *and* ***Hold meeting*** *are responses to* ***Open case*** *and therefore added to the set of pending responses and finally* ***Close case*** *and* ***Update Case*** *are added to the set of included events through the include relation from* ***Open case****. The enabled events for* M' *are* ***Extend Deadline, Propose dates-LO, Update Case*** *and* ***Close case****. Note that the event* ***Open case*** *is not enabled because there is a milestone relation from* ***Close case****, and* ***Close case*** *is in the response set. In other words, the case has to be closed by the union before it can be opened again.*

We now proceed to define the conservative extension of DCR Graphs to allow (discrete) time constraints. The aim is to allow modeling interesting timed systems while preserving that finite DCR Graphs have tractable finite state semantics and the technique for distribution of DCR Graphs still applies. The basic idea is to annotate response relations with discrete (possibly infinite) time constraint $k \in \infty$ and condition relations with a finite time constraint $j \in \omega$. A response relation $e \overset{k}{\bullet\!\!\rightarrow} e'$ then specifies that the response event e' must happen within k time steps after the last time e happened. The condition relation $e \overset{j}{\rightarrow\!\!\bullet} e'$ specifies that the last time the condition event e happened must be at least j time steps before e' can happen. That is, the time constraint $e \overset{\omega}{\bullet\!\!\rightarrow} e'$ means that the event e' should happen eventually after e happens, and the time constraint $e \overset{0}{\rightarrow\!\!\bullet} e'$ means that event e should have happened before e' can happen, corresponding to respectively the response and condition constraints in un-timed DCR Graphs. For this reason we often write $e \bullet\!\!\rightarrow e'$ for $e \overset{\omega}{\bullet\!\!\rightarrow} e'$ and $e \rightarrow\!\!\bullet e'$ for $e \overset{0}{\rightarrow\!\!\bullet} e'$.

To be able to evaluate the timed constraints we extend markings to include two functions, a function $t_{ex} : \mathsf{Ex} \to \omega$ recording the time since the event was last executed, and $t_{re} : \mathsf{Re} \to \infty$ recording the time deadline of the pending responses.

If there exists a maximal condition deadline, which is always the case if the timed DCR Graph is finite, we say that it is bounded. For bounded timed DCR Graphs we only need to record the time since last execution if it is below the maximal time constraint and otherwise record it as the maximal condition deadline.

Definition 3.2. *A* Timed *Dynamic Condition Response Graph (Timed DCR Graph) G is a tuple* $(\mathsf{E}, \mathsf{M_t}, \to\bullet, t_c, \bullet\to, t_r, \to\diamond, \to+, \to\%, \mathsf{L}, l)$*, where*

(i) $\mathsf{M_t} = (\mathsf{M}, t_{ex}, t_{re}) \in \mathcal{M}(G)$ *is the* timed *marking, for* $\mathcal{M}(G) =_{def} (\mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})) \times \mathsf{Ex} \to \omega \times \mathsf{Re} \to \infty$ *if* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$,

(ii) $(\mathsf{E}, \mathsf{M}, \to\bullet, \bullet\to, \to\diamond, \to+, \to\%, \mathsf{L}, l)$ *is a DCR Graph, referred to as the underlying DCR Graph,*

(iii) $t_{ex} : \mathsf{Ex} \to \omega$ *is the time since the event happened,*

(iv) $t_{re} : \mathsf{Re} \to \infty$ *is the maximal time deadline of the required response,*

(v) $t_c : (\to\bullet) \to \omega$ *is the (minimal) required time delay since the condition event happened,*

(vi) $t_r : (\bullet\to) \to \infty$ *is the maximal time deadline of the response event.*

We write $e \overset{k}{\bullet\to} e'$ *for* $e \bullet\to e'$ *and* $t_r(e, e') = k$ *and similarly write* $e \overset{k}{\to\bullet} e'$ *for* $e \to\bullet e'$ *and* $t_c(e, e') = k$*. We define the maximal condition delay as* $maxc_G =_{def} \mathsf{sup}\{k \mid \exists e, e' \in \mathsf{E}.e \overset{k}{\to\bullet} e'\}$ *and the minimal included response deadline by* $minr_G = \mathsf{min}\{t_{re}(e) \mid \exists e \in \mathsf{Re} \cap \mathsf{In}\}$.

Notation: We introduce the following shorthand notation for annotating the events in the executed and response sets with the their timestamp according to respectively t_{ex} and t_{re}: For a marking $\mathsf{M_t} = ((\{e_1, e_2\}, \{e_2, e_3\}, \mathsf{In}), t_{ex}, t_{re})$ we write $(\{e_1 : t_{ex}(e_1), e_2 : t_{ex}(e_2)\}, \{e_2 : t_{re}(e_2), e_3 : t_{re}(e_3)\}, \mathsf{In})$.

In Def. 3.3 below we formalize when events are enabled and the result of executing an event in a timed DCR Graph. An event e is enabled in a timed DCR Graph if it is enabled in the underlying un-timed DCR Graph and for all condition events $e' \overset{k}{\to\bullet} e$ the time $t_{ex}(e')$ since the last execution of e' is greater or equal than

k. If an event e is executed the sets $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ are updated as for the un-timed DCR Graphs, and in addition, the execution time for e is set to 0 and all response deadlines for events e' where $e \overset{k}{\bullet\!\!\rightarrow} e'$ are set to k.

Note that the execution of an event does not advance time. Instead we introduce the notion of *time advance steps* $n \in \omega$, which increases the execution time of events in Ex and decreases the response deadlines for events in Re. A time advance step n is only enabled if $n \leq minr_G$, that is, all currently *included* response deadlines are greater than or equal to n. This means that time can not pass a response deadline of an included event, but it may pass a response deadline of an excluded event. In the latter case, the deadline becomes zero.

Definition 3.3. *For a timed Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M_t}, \rightarrow\!\bullet, t_c, \bullet\!\!\rightarrow, t_r, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, and timed marking* $\mathsf{M_t} = \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}), t_{ex}, t_{re}\big)$ *we define that an event* $e \in \mathsf{E}$ *is* enabled*, written* $\mathsf{M_t} \vdash_G e$*, if* $\mathsf{M} \vdash_{G'} e$*, where* G' *is the underlying un-timed DCR Graph of* G *and* $\forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\!\bullet} e \implies k \leq t_{ex}(e')$*. Moreover, for* $n \in \omega$ *we define that the* time advance step n *is enabled, written* $\mathsf{M_t} \vdash_G n$ *if* $minr_G \geq n$*.*

We define the result of executing an event e *as* $\mathsf{M_t} \oplus_G e =_{def} \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e, t'_{ex}, t'_{re}\big)$*, where*

(i) $t'_{ex}(e') = \begin{cases} 0 & \textit{if } e' = e \\ t_{ex}(e') & \textit{otherwise} \end{cases}$

(ii) $t'_{re}(e') = \begin{cases} k & \textit{if } e \overset{k}{\bullet\!\!\rightarrow} e' \\ t_{re}(e') & \textit{otherwise} \end{cases}$

We define the result of advancing time with n *by* $\mathsf{M_t} \oplus_G n =_{def} \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}), t_{ex} \oplus n, t_{re} \ominus n\big)$*, where* $t_{ex} \oplus n(e) =_{def} \mathsf{min}\{t_{ex}(e) + n, maxc_G\}$ *and* $t_{re} \ominus n(e) =_{def} \mathsf{max}\{t_{re}(e) - n, 0\}$.

Example 3.2. *As an example, consider again the Timed DCR Graph* G_t *from Fig. 3: the initial marking of* G_t *is* $\mathsf{M_t} = (\emptyset, \emptyset, \mathsf{E} \setminus \{$*Accept LO*, *Accept DA*, *Close Case*, *Update Case*$\})$*, note that because of our shorthand notation and the empty* Ex *and* Re *sets the marking looks the same as the marking* M *of the underlying DCR Graph* G*. The only enabled event for the initial marking* $\mathsf{M_t}$ *is* *Open case*. *Executing* *Open case* *results in the marking* $\mathsf{M_t}' = \mathsf{M_t} \oplus_G$ *Open case* $= (\{$*Open case* $: 0\}, \{$*Propose dates-LO* $: 3,$ *Hold meeting* $:$

14, ***Close Case*** : $\omega\}$, $\mathsf{E} \setminus \{$***Accept LO***, ***Accept DA***$\})$. *Except for the timed part* $\mathsf{M_t}$*' is the same as* M*' that resulted from executing* ***Open case*** *on the underlying DCR Graph G. In the timed marking we also record the time that has passed since* ***Open case*** *was executed and the deadlines for which the responses on* ***Propose dates-LO***, ***Hold meeting*** *and* ***Close case*** *need to be satisfied. Note that unlike the untimed example* ***Extend Deadline*** *is not enabled because of the time constraint on the condition.*

The timed response on ***Propose dates-LO*** *limits the time to advance from the marking* $\mathsf{M_t}$*' to a maximum of 3 steps. Doing a time advance step of size 3 results in the new marking* $\mathsf{M_t}'' = \mathsf{M_t}' \oplus_G 3 = (\{$***Open case*** $: 3\}, \{$***Propose dates-LO*** $: 0$, ***Hold meeting*** $: 11$, ***Close Case*** $: \omega\}$, $\mathsf{E} \setminus \{$***Accept LO***, ***Accept DA***$\})$. *We can now no longer advance time before* ***Propose dates-LO*** *has been executed or excluded.*

It follows directly from the definition above that if an event is enabled in a timed DCR Graph then it is also enabled in the underlying (un-timed) DCR Graph. Moreover, the result on the sets $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ in the marking when executing an event in a timed DCR Graph is the same as when it is executed in the underlying DCR Graph and time advance steps do not change the marking of the underlying DCR Graph. Conversely, a DCR Graph G can be regarded as a timed DCR Graph G_t having G as the underlying DCR Graph and all condition delays (and execution times in the marking) 0 and all response deadlines (in the graph and the marking) ω. It then holds that, if an event is enabled in the DCR Graph G, then it is also enabled in the corresponding timed DCR Graph G_t. Moreover, the execution times and response deadlines in the marking will always be 0 and ω respectively.

We define in Def. 3.4 timed (must) executions and the corresponding labelled transition system for timed DCR Graphs as for un-timed DCR Graphs, except that executions can now also contain time advance steps. Moreover, for an execution to be accepting it is required that it is accepting in the underlying DCR Graph and moreover contains infinitely many time advance steps. Thus, an accepting execution may contain only finitely many events, but then it will after the last event contain an infinite sequence of time advance steps. It follows directly that (accepting) executions in a timed DCR Graph correspond to (accepting) executions in the underlying DCR Graph (where the time advance steps are removed). Moreover, an (accepting) execution in a DCR Graph G will have infinitely many corresponding (accepting) executions in the corresponding timed DCR Graph G_t obtained by interleaving the untimed execution with any infinite sequence of time advance steps.

Definition 3.4. *For a timed Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M_t}, \rightarrow\!\bullet, t_c, \bullet\!\rightarrow, t_r, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *we define a* timed execution σ_{M} *of* G *of length* $k \in \infty$ *from* M *to be a (finite or infinite) sequence of tuples* $\sigma : [k] \rightarrow \mathcal{M}(G) \times (\mathsf{E} \times \mathsf{L} \cup \omega) \times \mathcal{M}(G)$ *such that if for* $i \in [k]$*,*

- $\sigma(i) = (\mathsf{M}_i, e_i, a_i, \mathsf{M}'_i) \wedge a_i = l(e_i) \wedge \mathsf{M}_i \vdash_G e_i \wedge \mathsf{M}'_i = \mathsf{M}_i \oplus_G e_i$ *or*

- $\sigma(i) = (\mathsf{M}_i, n, \mathsf{M}'_i) \wedge \mathsf{M}_i \vdash_G n \wedge \mathsf{M}'_i = \mathsf{M}_i \oplus_G n$

and $\mathsf{M} = \mathsf{M}_0$ *and* $\forall i \in [k-1].\mathsf{M}'_i = \mathsf{M}_{i+1}$.

We say the execution σ *is a* must *execution if* $\forall i \in [k].\sigma(i) = (\mathsf{M}_i, e_i, a_i, \mathsf{M}'_i) \implies e_i \in \mathsf{In}_i \cap \mathsf{Re}_i$ *and* accepting *if*

(i) $\forall i \in [k].\big(\forall e \in \mathsf{In}_i \cap \mathsf{Re}_i.\exists j \geq i.e_j = e \vee e \notin \mathsf{In}'_j)\big)$ *and*

(ii) $\forall i \in \omega.\exists k \in \omega.\exists j > i.\sigma(i) = (\mathsf{M}_i, k, \mathsf{M}'_i)$

where $\mathsf{M}_i = ((\mathsf{Ex}_i, \mathsf{In}_i, \mathsf{Re}_i), t_{exi}, t_{rei})$ *and* $\mathsf{M}'_j = ((\mathsf{Ex}'_j, \mathsf{In}'_j, \mathsf{Re}'_j), t'_{exj}, t'_{rej})$*. Let* $\mathsf{exe}_{\mathsf{M_t}}(G)$*,* $\mathsf{mexe}_{\mathsf{M_t}}(G)$*,* $\mathsf{acc}_{\mathsf{M_t}}(G)$*and* $\mathsf{macc}_{\mathsf{M_t}}(G)$ *denote respectively the set of all executions, all must executions, all accepting executions, and all accepting must executions of G starting in marking* $\mathsf{M_t}$.

We say that a marking M' *is reachable in G (from the marking* M*) if there exists a finite execution ending in* M' *and let* $\mathcal{M}_{\mathsf{M}\rightarrow^*}(G)$ *denote the set of all reachable markings from* M.

We define the corresponding labeled transition system for G as $\mathsf{TS}(G) = (\mathcal{M}(G), \mathsf{M_t}, \mathcal{EL}(G), \rightarrow)$ *where* $\mathcal{EL}(G) = \mathsf{E} \times \mathsf{L} \cup \omega$ *is the set of labels of the transition system,* $\mathsf{M_t}$ *is the initial marking, and* $\rightarrow \subseteq \mathcal{M}(G) \times \mathcal{EL}(G) \times \mathcal{M}(G)$ *is the transition relation defined by* $\mathsf{M} \xrightarrow{\phi} \mathsf{M}'$ *if there exists a timed execution* σ_{M} *of G of length* 1 *from* M *such that* $\sigma(0) = (\mathsf{M}, \phi, \mathsf{M}')$.

Finally we define a zeno-run to be an infinite run with only finitely many time steps.

It is worth noting that if the DCR Graph is finite, the reachable set of states for the corresponding labelled transition system will be finite.

Lemma 3.1. *The LTS for any finite timed DCR Graph G has a finite number of reachable states.*

Proof *The sets of executed events, pending responses and included events are limited to the power set of the finite set of events since* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \in (\mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}))$*. The execution times of executed events are limited to the maximum condition time, i.e.* $\forall e \in \mathsf{Ex} \mid t_{ex}(e) \in [0 \ldots maxc_G]$*. The response times of pending responses are limited to the maximum response time or* ω*, i.e.* $\forall e \in \mathsf{Re} \mid t_{re}(e) \in [0 \ldots \mathsf{max}\{k \mid \exists e, e' \in \mathsf{E}.e \overset{k}{\bullet\!\!\to} e'\}] \cup \omega$.

4. Safety and Liveness Properties

In this section we define safety and liveness properties of timed DCR Graphs and prove they are decidable for finite, bounded timed DCR Graphs.

First we identify the unwanted markings, referred to as time-locked, from where time can no longer progress. Note that zeno runs may still exist from a time-locked marking.

Definition 4.1. *For a timed Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M_t}, \to\!\bullet, t_c, \bullet\!\!\to, t_r, \to\!\diamond, \to\!+, \to\!\%, \mathsf{L}, l)$ *with a timed marking* $\mathsf{M_t} = \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}), t_{ex}, t_{re}\big)$ *we define that* $\mathsf{M_t}$ *is time-locked , written* $\mathsf{TL}(\mathsf{M_t})$*, if* $\forall \mathsf{M_t}' \in \mathcal{M}_{\mathsf{M_t} \to^*}.\neg \mathsf{M_t}' \vdash_G 1$*, meaning that there is no reachable marking from which time can progress. We define that* G *is* timelock free*, if* $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M} \to^*}.\neg \mathsf{TL}(\mathsf{M}')$*, meaning that there exists no reachable time-locked marking.*

Proposition 4.1. *Time-lock and time-lock freedom is decidable for finite bounded DCR Graphs.*

Proof *To determine if a marking* $\mathsf{M_t}$ *is time-locked, we must check for the existence of a transition labelled by a time-step from any of the reachable markings from* $\mathsf{M_t}$*, which is finite by Lem. 3.1. To determine time-lock freedom for a graph* G *we just have to check every of the finitely many reachable markings from the initial marking if it is time-locked.*

A timed DCR Graph is said to be *deadlock free* if and only if for any reachable marking, there is either an enabled event or no included required responses. Furthermore, it is to be *strongly deadlock free* if and only if for any reachable marking, there is either an enabled event which is also a required response or no included required responses.

Definition 4.2. *For a timed Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M_t}, \to\!\bullet, t_c, \bullet\!\!\to, t_r, \to\!\diamond, \to\!+, \to\!\%, \mathsf{L}, l)$ *with a timed marking* $\mathsf{M_t} = \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}), t_{ex}, t_{re}\big)$

we define that $\mathsf{M_t}$ *is in* deadlock*, written* $\mathsf{DL}(\mathsf{M_t})$*, if* $\forall e \in \mathsf{E}.\mathsf{M_t} \not\vdash_G e \wedge (\mathsf{In} \cap \mathsf{Re} \neq \emptyset)$ *and that* $\mathsf{M_t}$ *is in* strong deadlock*, written* $\mathsf{SDL}(\mathsf{M_t})$*, if* $\forall e \in \mathsf{Re}.\mathsf{M_t} \not\vdash_G e \wedge (\mathsf{In} \cap \mathsf{Re} \neq \emptyset)$ *We define that G is* deadlock free*, if* $\forall \mathsf{M_t}' \in \mathcal{M}_{\mathsf{M_t} \to^*}.\neg\mathsf{DL}(\mathsf{M_t}')$ *and we say that G is* strongly deadlock free*, if* $\forall \mathsf{M_t}' \in \mathcal{M}_{\mathsf{M_t} \to^*}.\neg\mathsf{SDL}(\mathsf{M_t}')$.

Proposition 4.2. *Deadlock and strong deadlock freedom is decidable for finite DCR Graphs.*

Proof *Follows easily from the definition and Lem. 3.1.*

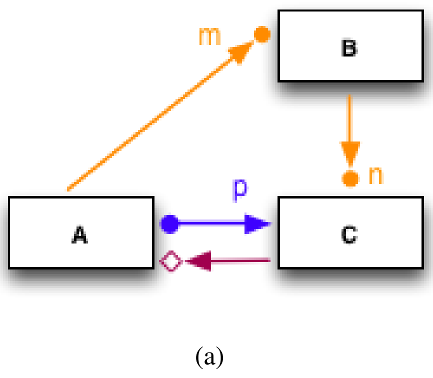
A timed DCR Graph is said to be *live* if and only if, in every reachable marking, it is always possible to continue along an accepting run. We say it is *strongly live* if and only if, from any reachable marking there exists an accepting must execution.

Definition 4.3. *For a timed Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M_t}, \to\bullet, t_c, \bullet\to, t_r, \to\diamond, \to+, \to\%, \mathsf{L}, l)$ *we define that the G is* live*, if* $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M} \to^*}.\mathsf{acc}_{\mathsf{M}'}(G) \neq \emptyset$*, and* strongly live*, if* $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M} \to^*}.\mathsf{macc}_{\mathsf{M}'}(G) \neq \emptyset$*,*

Proposition 4.3. *Liveness and strong liveness is decidable for finite DCR Graphs.*

Proof *(Outline) First of all note that it is sufficient to consider the sub graph where all time steps increment the time with* 1*. In [25, 28] is given a construction of a Büchi-automaton from an un-timed DCR Graph which accepts the same executions as the DCR Graph and essentially having markings of the DCR Graph as states. This construction can be extended to timed DCR Graphs by adding the time functions to the markings (increasing the number of states by a constant factor), adding time steps and adding new marked copies of all states which are states reached by a time advance step from a state which would have been accepting for the underlying un-timed DCR Graph. These marked states will be the accepting states of the automata. The definition guarantees that an execution is accepting if and only if does not leave a response pending and included continuously (as for the un-timed DCR Graphs), and it does make a time step infinitely often. This is the definition of acceptance for timed DCR Graphs. Decidability of liveness and strong liveness then follows from the decidability of language emptiness of finite Büchi-automata.*

Fig. 4(a) shows a timed DCR Graph which, depending on the time constraints as shown in table 1, may have time-locks, deadlocks and violate liveness. The



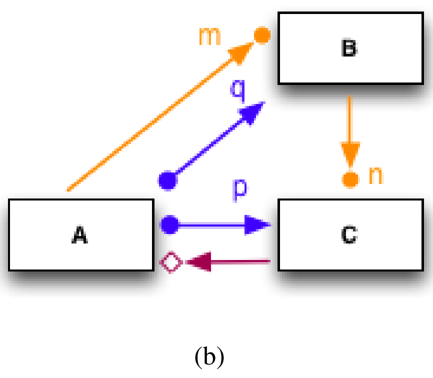


Figure 4: Examples of time-lock and (strongly) deadlock freedom in timed DCR Graphs

	Time-lock free	Deadlock free	Live
$m > p$			
$m \leq p \wedge n > 0 \wedge p < \omega$		x	
$(m \leq p \wedge n = 0) \vee p = \omega$	x	x	x

Table 1: Values for p, m and n in Fig. 4(a)

graph consist of three events: A, B, and C. The event A is a condition for B, and the time deadline requires that the last execution of A must have happened m time steps before B can happen. Similarly, the B is a condition for C, and the time deadline requires that the last execution of B must have happened n time steps before B can happen. The event C is a response to the event A, and the deadline requires that it must happen within p time steps after the last execution of A. The milestone relation from C to A is a kind of alternation pattern, it enforces that A can not happen as long as C is required as a response, i.e. two As can not happen without at least one C in between.

Now, if $m > p$ then the graph enters both a marking which is both time-locked and deadlocked if A happens and time advances with p steps. Since B is a condition for C it must be executed before C can be executed, but this is not possible since the delay constraint m is greater than p. The event A can not be executed before C has been executed due to the milestone relation, thus we have a deadlock. And since C is required urgently, i.e. within deadline 0, time can not progress either, i.e. we have a time-lock.

This deadlock can be resolved if $m \leq p$. Then B can be executed before the deadline p expires, and we can repeat doing B infinitely often. However, the graph may still contain a time-lock, and thus also violate liveness: If $n > 0$ and $p < \omega$ and we do B exactly p time steps after A, then we can not do C since it requires a delay of $n > 0$ after the execution of B, and time can not progress because C is required urgently, i.e. within deadline 0. The time-lock is resolved if $n = 0$ or $p = \omega$, since then we can either do C just after B (without advancing time) or the deadline on C will never expire and lead to time-lock.

The graph in Fig. 4(a) is neither strongly deadlock free nor strongly live for any of the parameters. If A is executed it is not possible to execute C if only required events are executed. This can be resolved by making B a response to A as shown in Fig. 4(b). Strongly deadlock and live DCR Graphs may be needed if the process is distributed between different roles as considered in the next section. If A and B are carried out by role N, and C by role M, then role N may believe that it is not necessary to do more after doing A. However, role M will then be stuck with a required action C which is blocked because B has not been executed.

5. Timed DCR Graphs as Global Contracts

The DCR Graphs model admits a very general technique for distributing a graph describing a global contract for e.g. a cross-organizational process, as a network of synchronously communicating graphs describing the contracts for each local organization [27]. The technique is based on a notion of projection, restricting the graph to a subset of the events and labels. The projection introduces a notion of *interface* events, that can be regarded as a subscription to executions of events in other components.

In Fig. 5 it is shown how the technique can be used to project the global graph in Fig. 3 to three local graphs, representing the part of the process to be carried out by respectively the unions (U), the umbrella organization for the unions (LandsOrganisationen, LO) and the umbrella organization for the employers (Dansk Arbejdsgiverforening, DA). The Open case event with the double border in the LO local graph indicate that it needs to subscribe to the event Open case in order to know when it can and must propose dates and hold a meeting. The unions on the other hand need not subscribe to any external events, they only need to consider their own events Open case, Close case and Update case.

5.1. Projection

First we define how to project a DCR Graph G with respect to a *projection parameter* $\delta = (\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$ where $\delta_{\mathsf{E}} \subseteq \mathsf{E}$ is a subset of the events of G and $\delta_{\mathsf{L}} \subseteq \mathsf{L}$ is a subset of the labels.

Intuitively, the projection $G_{|\delta}$ contains only those events and relations that are relevant for the execution of events in δ_{E} and the labeling is restricted to the set δ_{L}. This includes both the events in δ_{E} and any other event that can affect the marking, or the ability to execute an event in δ_{E}. The technical difficulty is to infer the events and relations not in δ_{E}, referred to as *external events* below, that should be included in the projection because they influence the execution of the workflow restricted to the events in δ_{E}. The external events are never executed by the local component but can be executed by other components, in which case the local component must be notified so that it can update its marking accordingly.

Definition 5.1. *If* $G = (\mathsf{E}, \mathsf{M}_{\mathsf{t}}, \rightarrow\!\bullet, t_c, \bullet\!\rightarrow, t_r, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *then* $G_{|\delta} = (\mathsf{E}_{|\delta}, \mathsf{M}_{\mathsf{t}|\delta}, \rightarrow\!\bullet_{|\delta}, t_{c|\delta}, \bullet\!\rightarrow_{|\delta}, t_{r|\delta}, \rightarrow\!\diamond_{|\delta}, \rightarrow\!+_{|\delta}, \rightarrow\!\%_{|\delta}, \delta_{\mathsf{L}}, l_{|\delta})$ *is the projection of* G *with respect to* $\delta = (\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$*,* $\delta_{\mathsf{E}} \subseteq \mathsf{E}$ *and* $\delta_{\mathsf{L}} \subseteq \mathsf{L}$ *where:*

(i) $\mathsf{E}_{|\delta} = \rightarrow \delta_{\mathsf{E}}$*, for* $\rightarrow = \bigcup_{c \in C} c$*, and* $C = \{\mathsf{id}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \bullet\!\rightarrow\rightarrow\!\diamond, \rightarrow\!+\rightarrow\!\bullet, \rightarrow\!\%\rightarrow\!\bullet, \rightarrow\!+\rightarrow\!\diamond, \rightarrow\!\%\rightarrow\!\diamond\}$

(ii) $l_{|\delta}(e) = \begin{cases} l(e) \cap \delta_{\mathsf{L}} & \textit{if } e \in \delta_{\mathsf{E}} \\ \emptyset & \textit{otherwise} \end{cases}$

(iii) $\mathsf{M_{t|\delta}} = ((\mathsf{Ex}_{|\delta}, \mathsf{Re}_{|\delta}, \mathsf{In}_{|\delta}), t_{ex|\delta}, t_{re|\delta})$ *where:*

- *(a)* $\mathsf{Ex}_{|\delta} = \mathsf{Ex} \cap \mathsf{E}_{|\delta}$
- *(b)* $\mathsf{Re}_{|\delta} = \mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$
- *(c)* $\mathsf{In}_{|\delta} = \mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\, \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$
- *(d)* $t_{ex|\delta}\,(e) = t_{ex}\,(e)$ *if* $e \in \mathsf{Ex}_{|\delta}$
- *(e)* $t_{re|\delta}\,(e) = t_{re}\,(e)$ *if* $e \in \mathsf{Re}_{|\delta}$

(iv) $\rightarrow\!\bullet_{|\delta} = \rightarrow\!\bullet \cap ((\rightarrow\!\bullet\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$

(v) $t_{c|\delta} : (\rightarrow\!\bullet_{|\delta}) \rightarrow \omega$
$t_{c|\delta}\,(e, e') = t_c\,(e, e')$

(vi) $\bullet\!\rightarrow_{|\delta} = \bullet\!\rightarrow \cap ((\bullet\!\rightarrow\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \times (\rightarrow\!\diamond\, \delta_{\mathsf{E}})) \cup ((\bullet\!\rightarrow\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}}))$

(vii) $t_{r|\delta} : (\bullet\!\rightarrow_{|\delta}) \rightarrow \infty$
$t_{r|\delta}\,(e, e') = t_r\,(e, e')$

(viii) $\rightarrow\!\diamond_{|\delta} = \rightarrow\!\diamond \cap ((\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$

(ix) $\rightarrow\!+_{|\delta} = \rightarrow\!+ \cap \Big(((\rightarrow\!+\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}} \Big) \cup \Big((\rightarrow\!+\rightarrow\!\bullet\, \delta_{\mathsf{E}}) \times (\rightarrow\!\bullet\, \delta_{\mathsf{E}} \Big) \cup \Big((\rightarrow\!+\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \times (\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \Big)$

(x) $\rightarrow\!\%_{|\delta} = \rightarrow\!\% \cap \Big(((\rightarrow\!\%\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}} \Big) \cup \Big((\rightarrow\!\%\rightarrow\!\bullet\, \delta_{\mathsf{E}}) \times (\rightarrow\!\bullet\, \delta_{\mathsf{E}} \Big) \cup \Big((\rightarrow\!\%\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \times (\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \Big)$

In *(i)* we define the set of events as the union of the set δ_{E} of events that we project over, any event that has a direct relation towards an event in δ_{E} and events that exclude or include an event which is either a condition or a milestone for an event in δ_{E}. The additional events will be included in the projection without labels, as can be seen from the definition of the labeling function in *(**??**)*. This means that the events can not be executed locally. However, when composed in a network containing other processes that can execute these events, their execution will be communicated to the process. For this reason we refer to these events as the (additional) external events of the projection. As proven in Prop. A.1-A.3 the communication of the execution of this set of external events in addition to the local events shared by others ensures that the local state of the projection stays consistent with the global state.

Further *(iii)* defines the projection of the marking: The executed events remain the same, but are limited to the events in $\mathsf{E}_{|\delta}$. The responses are restricted to events in δ_E and events that have a milestone relation to an event in δ_E because these are the only responses that will affect the local execution of the projected graph. Note that these events will by definition be events in $\mathsf{E}_{|\delta}$ but may be external events. In case of set of included events, we take the actual included status of the events in projection parameter along with the events that are conditions and milestones to the events in projection parameter, as the include status of those events will have an influence on the execution of events in local graph. All other external events of the projected graph are not included in the projected marking regardless of their included status in the marking of the global graph, because their include/exclude status will have no influence on the execution of events in local graph.

Finally, *(iv)*, *(vi)*, *(viii)*, *(ix)* and *(x)* state which relations should be included in the projection. For the events in δ_E all incoming relations should be included. Additionally inclusion and exclusion relations to events that are either a condition or a milestone for an event in δ_E are included as well.

Fig. 5 shows how the case management example from fig. 3 can be projected over the three roles. The projection parameters for these projections are: $\delta_U = (\{\mathsf{Open\ case}, \mathsf{Update\ Case}\}, \{(\mathsf{Open\ case}, \mathsf{U}), (\mathsf{Update\ Case}, \mathsf{U})\})$, $\delta_{LO} = (\{\mathsf{Propose\ Dates\ -\ LO}, \mathsf{Accept\ LO}, \mathsf{Extend\ Deadline}, \mathsf{Hold\ Meeting}\}, \{(\mathsf{Propose\ Dates\ -\ LO}, \mathsf{LO}), (\mathsf{Accept\ LO}, \mathsf{LO}), (\mathsf{Extend\ Deadline}, \mathsf{LO}), (\mathsf{Hold\ Meeting}, \mathsf{LO})\})$ and $\delta_{DA} = (\{\mathsf{Propose\ Dates\ -\ DA}, \mathsf{Accept\ DA}\}, \{(\mathsf{Propose\ Dates\ -\ DA}, \mathsf{DA}), (\mathsf{Accept\ DA}, \mathsf{DA})\})$. From the figure one can see that the union needs to know nothing about the global contract. It can only open, close and update the case and because these events are not depending on any of the events of LO and DA, the union does not have to be aware of any other events. LO on the other hand needs to know about most of the events and relations in the contract because many of its internal events depend directly on events of the union or DA. This reflects the role of LO as an intermediary between the union and DA, which gives it a central role in the process. DA is only involved in arranging a meeting and needs to be aware of a few of the events of LO to properly play its role in this, but all other events that do not directly affect the meeting arrangement part of the contract are not relevant for DA and therefore do not have to be projected to the local graph for DA.

Prop. 5.1 below states the key correspondence between global execution of events and the local execution of events in a projection. We have provided the

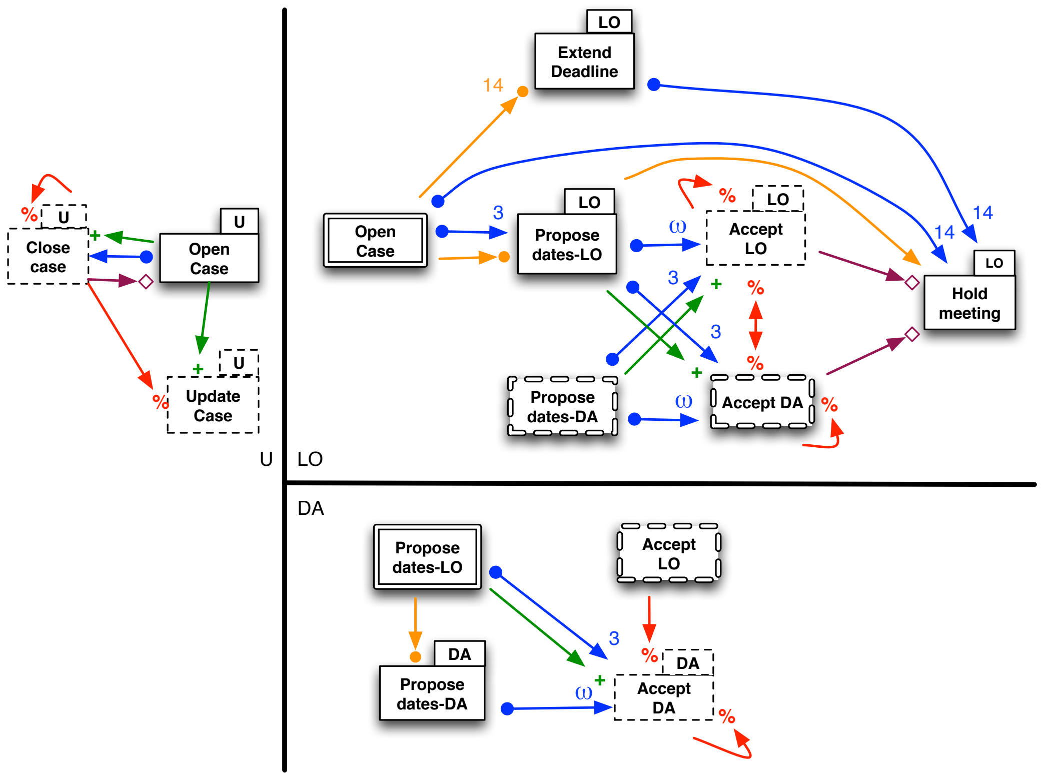


Figure 5: Projection of the timed DCR graph in fig. 3 over the roles U, LO and DA

details of the proof in the appendix. In order to prove the time extension, we first show the correspondence between global execution of events and the local execution of events in a projection of the underlying DCR Graph. We then use this result to prove the correspondence between the global and local execution of events for the time extension to the marking, t_{ex} and t_{re}.

Proposition 5.1. *Let* $G = (\mathsf{E}, \mathsf{M_t}, \rightarrow\bullet, t_c, \bullet\rightarrow, t_r, \rightarrow\diamond, \rightarrow+, \rightarrow\%, \mathsf{L}, l)$ *be a Timed DCR Graph and* $G_{|\delta}$ *its projection with respect to a projection parameter* $\delta = (\delta_\mathsf{E}, \delta_\mathsf{L})$. *Then,*

1. *if* $e \in \delta_\mathsf{E}$ *and* $l(e) \cap \delta_\mathsf{L} \neq \emptyset$ *then* $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}''$ *if and only if* $\mathsf{M_t}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M_t}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$
2. *if* $e \notin \mathsf{E}_{|\delta}$ *then* $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}'$ *implies* $\mathsf{M_t}_{|\delta} = \mathsf{M_t}'_{|\delta}$

3. if $e \in \mathsf{E}_{|\delta}$ (and $l(e) \cap \delta_{\mathsf{L}} = \emptyset$) then $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}'$ implies $\mathsf{M}_{\mathsf{t}|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}'_{|\delta}$.

Proof can be found in App. 1.

Example 5.1. *We consider the projection in Fig. 5. Recall from Ex. 3.2 that the execution of* ***Open case*** *in the initial marking*

$$\mathsf{M_t} = (\emptyset, \emptyset, \mathsf{E} \setminus \{\textit{Accept LO}, \textit{Accept DA}, \textit{Close Case}, \textit{Update Case}\})$$

gave rise to the marking

$$\mathsf{M_t}' = (\{\textit{Open case} : 0\},\\ \{\textit{Propose dates-LO} : 3, \textit{Hold meeting} : 14, \textit{Close case} : \omega\},\\ \mathsf{E} \setminus \{\textit{Accept LO}, \textit{Accept DA}\})$$

We now give the projected markings of $\mathsf{M_t}$ and $\mathsf{M_t}$' for the three projection parameters $\delta_U = (\delta_E^U, \delta_L^U)$, $\delta_{LO} = (\delta_E^{LO}, \delta_L^{LO})$, and $\delta_{DA} = (\delta_E^{DA}, \delta_L^{DA})$:

$$\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{U}}} = (\emptyset, \emptyset, \{\textit{Open case}\})$$

$$\mathsf{M_t}'_{|\delta_{\mathsf{U}}} = (\{\textit{Open case} : 0\}, \{\textit{Close Case} : \omega\},\\ \{\textit{Open case}, \textit{Close case}, \textit{Update case}\})$$

$$\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{LO}}} = (\emptyset, \emptyset, \{\textit{Open case}, \textit{Propose dates-LO},\\ \textit{Extend deadline}, \textit{Hold meeting}\})$$

$$\mathsf{M_t}'_{|\delta_{\mathsf{LO}}} = (\{\textit{Open case} : 0\}, \{\textit{Propose dates-LO} : 3, \textit{Hold meeting} : 14\},\\ \{\textit{Open case}, \textit{Propose dates-LO}, \textit{Extend deadline}, \textit{Hold meeting}\})$$

$$\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{DA}}} = \mathsf{M_t}'_{|\delta_{\mathsf{DA}}} = (\emptyset, \emptyset, \{\textit{Propose dates-LO}, \textit{Propose dates-DA}\})$$

Let us verify that each of the properties 1.-3. in Prop. 5.1 holds for the projection parameters δ_U, δ_{LO} and δ_{DA} when executing the event $e =$ ***Open case*** *in the initial marking $\mathsf{M_t}$.*

1. *This property applies to the projection over the union because* $e \in \delta_{\mathsf{E}}^{\mathsf{U}}$ *and* $l(e) \cap \delta_{\mathsf{L}}^{\mathsf{U}} \neq \emptyset$*. First of all the property requires us to verify that* e *is only enabled in the global graph if and only if it is enabled in the local graph. This is the case because both* $\mathsf{M_t} \vdash_G e$ *and* $\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{U}}} \vdash_G e$*. Secondly the property requires us to verify that the result of executing* e *on the global graph and then projecting the resulting marking over* δ_U *is the same as executing* e *on the projected graph. This is the case because:* $\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{U}}} \oplus_{G_{|\delta_{\mathsf{U}}}} e = \mathsf{M}'_{\mathsf{t}|\delta_{\mathsf{U}}}$.

2. *This property applies to the projection over DA because* $e \notin \mathsf{E}_{|\delta_{\mathsf{DA}}}$*. It requires us to verify that if* e *does not occur in the events of the projected graph* $G_{|\delta_{\mathsf{DA}}}$*, then the marking projected over* δ_{DA} *will be the same before and after the execution of* e*. This is the case because* $\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{DA}}} = \mathsf{M}'_{\mathsf{t}|\delta_{\mathsf{DA}}}$.

3. *This property applies to the projection over LO because* $e \in \mathsf{E}_{|\delta_{\mathsf{LO}}}$ *and* $l(e) \cap \delta_L^{\mathsf{LO}} = \emptyset$*. It requires us to show that if* e *is an external event in the projected graph* $G_{|\delta_{\mathsf{LO}}}$*, then it must be the case that if we execute* e *on* $G_{|\delta_{\mathsf{LO}}}$*, the resulting marking must be the same as if we had first executed* e *on* G *and then projected the result over* δ_{LO}*. This is the case because* $\mathsf{M}_{\mathsf{t}|\delta_{\mathsf{LO}}} \oplus_{G_{|\delta_{\mathsf{LO}}}} e = \mathsf{M}'_{\mathsf{t}|\delta_{\mathsf{LO}}}$.

5.2. Composition

Now we define the binary composition of two DCR Graphs. The *composition* of G_1 and G_2 is simply the component-wise union of the respective components.

Definition 5.2. $G_1 \cup G_2 = (\mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{M_t}, \rightarrow\!\bullet_1 \cup \rightarrow\!\bullet_2, t_{c1} \cup t_{c2}, \bullet\!\rightarrow_1 \cup \bullet\!\rightarrow_2, t_{r1} \cup t_{r2}, \rightarrow\!\diamond_1 \cup \rightarrow\!\diamond_2, \rightarrow\!+_1 \cup \rightarrow\!+_2, \rightarrow\!\%_1 \cup \rightarrow\!\%_2, \mathsf{L}_1 \cup \mathsf{L}_2, l_1 \cup l_2)$*, where*
$\mathsf{M_t} = ((\mathsf{Ex}_1 \cup \mathsf{Ex}_2, \mathsf{Re}_1 \cup \mathsf{Re}_2, \mathsf{In}_1 \cup \mathsf{In}_2), t_{ex1} \cup t_{ex2}, t_{re1} \cup t_{re2})$

Definition 5.3. *The composition* $G_1 \cup G_2$ *is* well-defined *when:*

(i) $\forall(e \in \mathsf{E}_1 \cap \mathsf{E}_2 \mid (e \in \mathsf{Ex}_1 \Leftrightarrow e \in \mathsf{Ex}_2))$

(ii) $\forall(e \in (\mathsf{E}_1^{\mathsf{i}} \cup \rightarrow\!\bullet \mathsf{E}_1^{\mathsf{i}} \cup \rightarrow\!\diamond \mathsf{E}_1^{\mathsf{i}}) \cap (\mathsf{E}_2^{\mathsf{i}} \cup \rightarrow\!\bullet \mathsf{E}_2^{\mathsf{i}} \cup \rightarrow\!\diamond \mathsf{E}_2^{\mathsf{i}}) \mid (e \in \mathsf{In}_1 \Leftrightarrow e \in \mathsf{In}_2)$

(iii) $\forall(e \in (\mathsf{E}_1^{\mathsf{i}} \cup \rightarrow\!\diamond \mathsf{E}_1^{\mathsf{i}}) \cap (\mathsf{E}_2^{\mathsf{i}} \cup \rightarrow\!\diamond \mathsf{E}_2^{\mathsf{i}}) \mid (e \in \mathsf{Re}_1 \Leftrightarrow e \in \mathsf{Re}_2)$

(iv) $\forall(e \in \mathsf{Ex}_1 \cap \mathsf{Ex}_2 \mid (t_{c1}\, e = t_{c2}\, e))$

(v) $\forall(e \in \mathsf{Re}_1 \cap \mathsf{Re}_2 \mid (t_{r1}\, e = t_{r2}\, e))$

(vi) $\forall (e, e' \in \rightarrow\!\bullet_1 \cap \rightarrow\!\bullet_2 |\ t_{c1}(e, e') = t_{c2}(e, e'))$

(vii) $\forall (e, e' \in \bullet\!\rightarrow_1 \cap \bullet\!\rightarrow_2 |\ t_{r1}(e, e') = t_{r2}(e, e'))$

Where: $\mathsf{E}^{\mathsf{i}}_i = \{e \in \mathsf{E}_i \mid l(e) \neq \emptyset\}$ *for* $i \in \{1, 2\}$

(i) ensures that those events that will be glued together have the same execution marking. *(ii)* ensures that events that will be glued together and in both DCR Graphs belong to either the set of internal events or the set of events that have a condition/milestone relation towards an internal event, have the same inclusion marking. *(iii)* ensures that events that will be glued together and in both DCR Graphs belong to the set of internal events have the same pending response marking. *(iv)* ensures that executed events that will be glued together have the same timed execution marking. *(v)* ensures that responses on events that will be glued together have the same deadline. *(vi)* ensures that shared conditions have the same required time delay. *(vii)* ensures that shared responses have the same maximal deadline. If $G_1 \cup G_2$ is well-defined, then we also say that G_1 and G_2 are *composable* with respect to each other.

Lemma 5.1. *The composition operator* $\cup$ *is commutative and associative.*

Proof According to definition 5.2, elements of the tuple defining the graph $G = G_1 \cup G_2$ are constructed from the union of the same elements in G_1 and G_2. Composition is therefore commutative and associative, because the union operator is commutative and associative.

Definition 5.4. *We call a vector* $\Delta = \delta_1 \ldots \delta_k$ *of projection parameters* covering *for some DCR Graph* $G = (\mathsf{E}, \mathsf{M_t}, \rightarrow\!\bullet, t_c, \bullet\!\rightarrow, t_r, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *if:*

1. $\bigcup\limits_{i \in [k]} \delta_{\mathsf{E}i} = \mathsf{E}$ *and*

2. $(\forall a \in \mathsf{L}. \forall e \in \mathsf{E}. a \in l(e) \Rightarrow (\exists i \in [k]. e \in \delta_{\mathsf{E}i} \wedge a \in \delta_{\mathsf{L}i})$

Proposition 5.2. *If some vector* $\Delta = \delta_1 \ldots \delta_k$ *of projection parameters is covering for some DCR Graph G then:* $\bigcup\limits_{i \in [k]} G_{|\delta_i} = G$

Proof *Since the vector of projection parameters is covering, every event and label is covered in at least one of the projections. Moreover the definition of composition 5.2, is defined over union of individual components. Hence when all projections are composed, we will get the same graph and hence* $\bigcup\limits_{i \in [k]} G_{|\delta_i} = G$.

Networks of timed DCR Graphs are defined exactly as networks of un-timed DCR Graphs introduced in [27].

Definition 5.5. *A* network of timed DCR Graphs *is a finite vector of timed DCR Graphs $\overline{G}$ sometimes written as $\Pi_{i\in[n]}G_i$ or $G_0\|G_2\|\ldots\|G_{n-1}$. Assuming $G_i = (\mathsf{E}_i, \mathsf{M}_i, \rightarrow\!\bullet_i, t_{ci}, \bullet\!\!\rightarrow_i, t_{ri}, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$, we define the set of events of the network by $\mathcal{E}(\Pi_{i\in[n]}G_i) = \cup_{i\in[n]}\mathsf{E}_i$ and the set of labels of the network by $\mathcal{L}(\Pi_{i\in[n]}G_i) = \cup_{i\in[n]}\mathsf{L}_i$ and we write the network marking as $\overline{\mathsf{M}} = \Pi_{i\in[n]}\mathsf{M}_i$. Finally, let $\mathcal{M}(\overline{G})$ denote the set of network markings of $\overline{G}$.*

We define when an event is locally enabled in one of the components and the result of executing an event locally as for un-timed DCR Graphs, except that we instead use the definition of timed enabled and timed update of the marking. That is, an event can be executed if it is locally enabled, and the result of executing it is that it is synchronously executed in all components of the network sharing the event. A time advance event is however defined to be enabled only if it is enabled in *all* components and the result of advancing the time is to advance it in all components. This ensures that time advances globally in the network.

Definition 5.6. *For a network of timed DCR Graphs $\overline{G} = \Pi_{i\in[n]}G_i$ where $G_i = (\mathsf{E}_i, \mathsf{M}_i, \rightarrow\!\bullet_i, t_{ci}, \bullet\!\!\rightarrow_i, t_{ri}, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$, an event $e \in \mathcal{E}(\Pi_{i\in[n]}G_i)$ is enabled at a location i in the distributed marking $\overline{\mathsf{M}} = \Pi_{i\in[n]}\mathsf{M}_i$, written $\overline{\mathsf{M}} \vdash_{\overline{G},i} e$, if $e \in \mathsf{E}_i \wedge \mathsf{M}_i \vdash_{G_i} e$, i.e. it is locally enabled in the ith timed dynamic condition response graph. The result of executing an event $e \in \mathcal{E}(\Pi_{i\in[n]}G_i)$ in a marking $\overline{\mathsf{M}} = \Pi_{i\in[n]}\mathsf{M}_i$ is the new marking $\overline{\mathsf{M}} \oplus_{\overline{G}_i} e = \Pi_{i\in[n]}\mathsf{M}'_i$ where $\mathsf{M}'_i = \mathsf{M}_i \oplus_G e$ if $e \in \mathsf{E}_i$ and $\mathsf{M}'_i = \mathsf{M}_i$ otherwise. For a network marking $\overline{\mathsf{M}} = \Pi_{i\in[n]}\mathsf{M}_i$ we define that the time advance event n is enabled, written $\overline{\mathsf{M}} \vdash_{\overline{G}} n$, if $\mathsf{M}_i \vdash_{G_i} n$ for all $i \in [n]$ and the result of advancing time with n by $\overline{\mathsf{M}} \oplus_{\overline{G}} n = \Pi_{i\in[n]}\mathsf{M}_i \oplus_{G_i} n$.*

We define executions of networks as follows. As for un-timed DCR Graphs, an event can be executed if it is locally enabled in a component where it has assigned at least one label. Time can be advanced globally if the time advance event is enabled as defined above.

Definition 5.7. *For a network of timed DCR Graphs $\overline{G} = \Pi_{i\in[n]}G_i$ where l_i is the labeling function of G_i, we define a* timed execution *$\sigma_{\overline{\mathsf{M}}}$ of $\overline{G}$ of length $k \in \infty$ from marking $\overline{\mathsf{M}}$ to be a (finite or infinite) sequence of tuples $\sigma : [k] \rightarrow \mathcal{M}(\overline{G}) \times ([n] \times \mathcal{E}(\Pi_{i\in[n]}G_i) \times \mathcal{L}(\Pi_{i\in[n]}G_i) \cup \omega) \times \mathcal{M}(\overline{G})$ such that for $i \in [k]$*

- $\sigma(i) = (\overline{\mathsf{M}}_i, h_i, e_i, a_i, \overline{\mathsf{M}}'_i) \wedge a_i \in l_{h_i}(e_i) \wedge \overline{\mathsf{M}}_i \vdash_{\overline{G},h_i} e_i \wedge \overline{\mathsf{M}}'_i = \overline{\mathsf{M}}_i \oplus_{\overline{G}} e_i$ *or*
- $\sigma(i) = (\overline{\mathsf{M}}_i, n, \overline{\mathsf{M}}'_i) \wedge \overline{\mathsf{M}}_i \vdash_{\overline{G}} n \wedge \overline{\mathsf{M}}'_i = \overline{\mathsf{M}} \oplus_{\overline{G}} n$

and $\overline{\mathsf{M}}_0 = \overline{\mathsf{M}}$ *and* $\forall i \in [k-1].\overline{\mathsf{M}}'_i = \overline{\mathsf{M}}_{i+1}$.

We say the execution is accepting *if*

(i) $\forall i \in [k], h \in [n].\big(\forall e \in \mathsf{In}_{h,i} \cap \mathsf{Re}_{h,i}.\exists j \geq i.e_j = e \vee e \notin \mathsf{In}'_{h,j})\big)$, *where* $\overline{\mathsf{M}}_i = \Pi_{h\in[n]}(\mathsf{Ex}_{h,i}, \mathsf{In}_{h,i}, \mathsf{Re}_{h,i})$ *and* $\overline{\mathsf{M}}'_j = \Pi_{h\in[n]}(\mathsf{Ex}'_{h,j}, \mathsf{In}'_{h,j}, \mathsf{Re}'_{h,j})$

(ii) $\forall i \in \omega.\exists h \in \omega.\exists j > i.\sigma(i) = (\overline{\mathsf{M}}_i, h, \overline{\mathsf{M}}'_i)$

We define the global transition system for a network of timed DCR Graphs as follows.

Definition 5.8. *For a network of timed DCR Graphs* $\overline{G} = \Pi_{i\in[n]}G_i$ *where* $G_i = (\mathsf{E}_i, \mathsf{M}_i, \rightarrow\!\bullet_i, t_{ci}, \bullet\!\rightarrow_i, t_{ri}, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$ *and* $\overline{\mathsf{M}} = \Pi_{i\in[n]}\mathsf{M}_i$, *we define the corresponding global transition system* $\mathsf{TS}(\overline{G})$ *to be the tuple*

$$(\mathcal{M}(\overline{G}), \overline{\mathsf{M}}, \mathcal{EL}(\overline{G}), \rightarrow_N)$$

where $\mathcal{EL}(\overline{G}) = (\mathcal{E}(\Pi_{i\in[n]}G_i) \times \mathcal{L}(\Pi_{i\in[n]}G_i)) \cup \omega$ *is the set of labels of the transition system,* $\overline{\mathsf{M}}$ *is the initial marking, and* $\rightarrow_N \subseteq \mathcal{M}(\overline{G}) \times \mathcal{EL}(\overline{G}) \times \mathcal{M}(\overline{G})$ *is the transition relation defined by* $\overline{\mathsf{M}}' \xrightarrow{(e,a)}_N \overline{\mathsf{M}}''$ *if there is a timed execution* $\sigma_{\overline{\mathsf{M}}'}$ *from* $\overline{\mathsf{M}}'$ *of length* 1 *such that* $\sigma_{\overline{\mathsf{M}}'}(0) = (\overline{\mathsf{M}}', i, e, a, \overline{\mathsf{M}}'')$ *for some* $i \in [n]$ *and* $\overline{\mathsf{M}}' \xrightarrow{h}_N \overline{\mathsf{M}}''$ *if there is a timed execution* $\sigma_{\overline{\mathsf{M}}'}$ *from* $\overline{\mathsf{M}}'$ *of length* 1 *such that* $\sigma_{\overline{\mathsf{M}}'}(0) = (\overline{\mathsf{M}}', h, \overline{\mathsf{M}}'')$. *(Accepting) executions of length* k *of the transition system is defined as sequences transitions obtained similarly from (accepting) executions* σ *of length* k *of the graph.*

We are now ready to state the main theorem which follows from Def 3.4, Def. 5.8 and Prop. 5.1.

Theorem 5.1. *For a timed DCR Graph* G, *a covering vector of projection parameters* $\Delta = \delta_1 \ldots \delta_n$ *and* $\overline{G}_\Delta = \Pi_{i\in[n]}G_{|\delta_i}$ *it holds that the relation* $\mathcal{R} = \{(\mathsf{M}, \overline{\mathsf{M}}_\Delta) \mid \mathsf{M} \in \mathcal{M}(G)$ *and* $\overline{\mathsf{M}}_\Delta = \Pi_{i\in[n]}\mathsf{M}_{|\delta_i}\}$ *is a bisimulation between* $\mathsf{TS}(G)$ *and* $\mathsf{TS}(\overline{G}_\Delta)$ *such that an execution is accepting in* $\mathsf{TS}(G)$ *if and only if the bisimilar execution is accepting in* $\mathsf{TS}(\overline{G}_\Delta)$.

Proof (outline) In order to prove bisimilarity, we show (1) $\exists \mathsf{M} \xrightarrow{(e,a)} \mathsf{M}'$ in $TS(G)$ if and only if $\exists \overline{\mathsf{M}}_\Delta \xrightarrow{(e,a)} \overline{\mathsf{M}'}_\Delta$ in $\mathsf{TS}(\overline{G}_\Delta)$ and (2) $\exists \mathsf{M} \xrightarrow{h} \mathsf{M}'$ in $TS(G)$ if and only if $\exists \overline{\mathsf{M}}_\Delta \xrightarrow{h} \overline{\mathsf{M}'}_\Delta$ in $\mathsf{TS}(\overline{G}_\Delta)$.

1. According to Def 3.4, $\mathsf{M} \xrightarrow{(e,a)} \mathsf{M}'$ in $TS(G)$ implies $\mathsf{M} \xrightarrow{(e,a)} \mathsf{M} \oplus_G e$, $\mathsf{M} \vdash_G e$ and $a \in \mathrm{l}(e)$. From Prop. 5.1 it then follows that $\overline{\mathsf{M}}_\Delta \xrightarrow{(e,a)} \overline{\mathsf{M}'}_\Delta$. Similarly, if $\overline{\mathsf{M}}_\Delta \xrightarrow{(e,a)} \overline{\mathsf{M}'}_\Delta$ in $\mathsf{TS}(\overline{G}_\Delta)$ then again using Prop. 5.1, there exists in $TS(G)$ a transition $\mathsf{M} \xrightarrow{(e,a)} \mathsf{M}'$.

2. Since the time advances globally and synchronously in the network, it easily follows that time advance steps can be mutually simulated and result in consistent updates of the deadlines in the markings.

Now we have to prove that a timed execution in $\mathsf{TS}(G)$ is accepting if and only if the corresponding execution is accepting in $\mathsf{TS}(\overline{G}_\Delta)$. If an execution in $\mathsf{TS}(\overline{G}_\Delta)$ is accepting, then

1. according to Def. 5.8, in the network of projected graphs, $\forall i \in [k], h \in [n].\big(\forall e \in \mathsf{In}_{|\delta_{h,i}} \cap \mathsf{Re}_{|\delta_{h,i}}.\exists j \geq i.e_j = e \vee e \notin \mathsf{In}'_{|\delta_{h,i}}\big)\big)$, where $\overline{\mathsf{M}}_{\Delta_i} = \Pi_{h\in[n]}(\mathsf{Ex}_{|\delta_{h,i}}, \mathsf{In}_{|\delta_{h,i}}, \mathsf{Re}_{|\delta_{h,i}})$ and $\overline{\mathsf{M}}'_{\Delta_j} = \Pi_{h\in[n]}(\mathsf{Ex}'_{|\delta_{h,i}}, \mathsf{In}'_{|\delta_{h,i}}, \mathsf{Re}'_{|\delta_{h,i}})$. According to proposition 5.1 and since $\mathsf{TS}(G) \sim \mathsf{TS}(\overline{G}_\Delta)$, if there exists an execution where an included response is eventually executed or excluded in $\mathsf{TS}(\overline{G}_\Delta)$, then we will also have the corresponding execution satisfying that an included response is eventually executed or excluded in $\mathsf{TS}(G)$.

2. As the time advances globally in the network, $\forall i \in \omega.\exists h \in \omega.\exists j > i.\sigma(i) = (\overline{\mathsf{M}}_i, h, \overline{\mathsf{M}}'_i)$ in $\mathsf{TS}(\overline{G}_\Delta)$ implies that $\forall i \in \omega.\exists h \in \omega.\exists j > i.\sigma(i) = (\mathsf{M}_i, h, \mathsf{M}'_i)$ in $\mathsf{TS}(G)$.

Therefore, if a timed execution is accepting in $\mathsf{TS}(\overline{G}_\Delta)$ then it is also accepting in $\mathsf{TS}(G)$. On the similar lines, it trivially follows that if a timed execution in $\mathsf{TS}(G)$ is accepting the corresponding execution in $\mathsf{TS}(\overline{G}_\Delta)$ is also accepting.

6. Conclusion

We have conservatively extended the declarative Dynamic Condition Response (DCR) Graph process model introduced in the PhD thesis of the second author [24, 28] to allow for (discrete) time deadlines. In particular, the simple operational semantics of DCR Graphs is conservatively extended with time steps, preserving that safety and liveness properties of the models can be formally verified by mapping bounded timed DCR Graphs to finite state automata. We proceeded to extend to timed DCR Graphs the general technique provided in [27] for safe distribution of a global DCR Graph as a network of communicating DCR Graphs which has the same behavior as the global DCR Graph. We have exemplified timed DCR Graphs and the distribution technique on a timed extension of the cross-organizational case management process studied in [34, 27]. The example shows how a timed DCR Graph can be used to describe the global contract for a timed workflow process involving several organizations, which can then be distributed as a network of communicating timed DCR Graphs, having a graph describing the local contract for each organization. Finally we have exemplified how the time deadlines may introduce both deadlocks and so-called time-locks where time cannot proceed in the model.

We plan for future work to study the application of the techniques in the present paper for contract-oriented programming of distributed event-based systems and context-sensitive services. In particular, we are currently developing an event-based programming language based on an extension of DCR Graphs with data and sub processes. We will also study the formal relation between timed DCR Graphs and other timed process models, in particular timed LTL [21], variants of Petri Net with time [63, 68, 71, 72] and timed automata [], and explore verification of timed DCR Graphs using existing tools for these models. Finally we intend to investigate the relation to the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [46].

References

[1] M. Dumas, W. M. van der Aalst, A. H. ter Hofstede, Process Aware Information Systems: Bridging People and Software Through Process Technology, Wiley-Interscience, 2005.

[2] M. D. Zisman, Representation, Specification and Automation of Office Procedures, Ph.D. thesis, Wharton School, University of Pennsylvania, 1977.

[3] J. C. BURNS, The evolution of office information systems, Datamation vol. 23,no. 4 (1977) 60–64.

[4] C. A. Ellis, Information control nets: A mathematical model of office information flow, Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, ACM SIGMETRICS Performance Evaluation Review 8 (1979) 225–240.

[5] C. A. Ellis, G. J. Nutt, Office information systems and computer science, ACM Comput. Surv. 12 (1980) 27–60.

[6] C. A. Petri, Kommunikation mit Automaten, Ph.D. thesis, Universitet Hamburg, 1962.

[7] H. Eshuis, Semantics and Verification of UML Activity Diagrams for Workflow Modelling, Ph.D. thesis, Univ. of Twente, 2002. CTIT Ph.D.-thesis series No. 02-44.

[8] R. Eshuis, R. Wieringa, Tool support for verifying uml activity diagrams, Software Engineering, IEEE Transactions on 30 (2004) 437 – 447.

[9] Object Management Group BPMN Technical Committee, Business Process Model and Notation, version 2.0, Webpage, 2011. `http://www.omg.org/spec/BPMN/2.0/PDF`.

[10] W. M. P. van der Aalst, Interorganizational workflows: An approach based on message sequence charts and petri nets., Systems Analysis - Modelling - Simulation 34 (1999) 335–367.

[11] E. Kindler, A. Martens, W. Reisig, Inter-operability of workflow applications: Local criteria for global soundness, in: Business Process Management, Models, Techniques, and Empirical Studies, Springer-Verlag, London, UK, 2000, pp. 235–253.

[12] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, K. Wolf, Multiparty Contracts: Agreeing and Implementing Interorganizational Processes, The Computer Journal 53 (2010) 90–106.

[13] W. M. P. v. d. Aalst, M. Weske, The p2p approach to interorganizational workflows, in: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, CAiSE '01, pp. 140–156.

[14] W. Vanderaalst, M. Weske, D. Grunbauer, Case handling: a new paradigm for business process support, Data & Knowledge Engineering 53 (2005) 129–162.

[15] M. Pesic, M. H. Schonenberg, N. Sidorova, W. M. P. Van Der Aalst, Constraint-based workflow models: change made easy, in: Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I, OTM'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 77–94.

[16] A. Pnueli, The temporal logic of programs, in: Proceedings of 18th IEEE FOCS, pp. 46–57.

[17] E. M.Clarke, O. Grumberg, D. A.Peled, Model Checking, MIT Press, 1999.

[18] M. Papazoglou, Making business processes compliant to standards and regulations, in: Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International, pp. 3 –13.

[19] M. Dwyer, G. Avrunin, J. Corbett, Patterns in property specifications for finite-state verification, in: Software Engineering, 1999. Proceedings of the 1999 International Conference on, pp. 411 –420.

[20] W. M. van der Aalst, M. Pesic, DecSerFlow: Towards a truly declarative service flow language, in: M. Bravetti, M. Nunez, G. Zavattaro (Eds.), Proceedings of Web Services and Formal Methods (WS-FM 2006), volume 4184 of *LNCS*, Springer Verlag, 2006, pp. 1–23.

[21] K. J. Kristoffersen, C. Pedersen, H. R. Andersen, Runtime verification of timed ltl using disjunctive normalized equation systems, Electronic Notes in Theoretical Computer Science 89 (2003) 210 – 225.

[22] G. J. Holzmann, The model checker spin, IEEE Trans. Softw. Eng. 23 (1997) 279–295.

[23] M. Y. Vardi, An automata-theoretic approach to linear temporal logic, in: Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 238–266.

[24] T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, in: K. Honda, A. Mycroft (Eds.), PLACES, volume 69 of *EPTCS*, pp. 59–73.

[25] R. Mukkamala, T. Hildebrandt, From dynamic condition response structures to büchi automata, in: Theoretical Aspects of Software Engineering (TASE), 2010 4th IEEE International Symposium on, pp. 187 –190.

[26] T. Hildebrandt, R. R. Mukkamala, T. Slaats, Nested dynamic condition response graphs, in: Proceedings of Fundamentals of Software Engineering (FSEN).

[27] T. Hildebrandt, R. R. Mukkamala, T. Slaats, Safe distribution of declarative processes, in: Proceedings of the 9th international conference on Software engineering and formal methods, SEFM'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 237–252.

[28] R. R. Mukkamala, A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs, Ph.D. thesis, IT University of Copenhagen, 2012. Forthcomming.

[29] T. Hildebrandt, Trustworthy pervasive healthcare processes (TrustCare) research project, Webpage, 2008. `http://www.trustcare.dk/`.

[30] R. R. Mukkamala, T. Hildebrandt, J. B. Tøth, The resultmaker online consultant: From declarative workflow management in practice to ltl, in: Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops, EDOCW '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 135–142.

[31] Resultmaker, 2008. `http://www.resultmaker.com/`.

[32] T. Hildebrandt, Dynamic condition response graphs - a dynamic temporal logic for event-based processes, in: 8th Scandinavian Logic Symposium, pp. 52–54.

[33] T. Hildebrandt, R. R. Mukkamala, T. Slaats, Verifying liveness and safety for declarative event-based workflows in spin, 2012. Sumitted for publication.

[34] T. Hildebrandt, R. Mukkamala, T. Slaats, Designing a cross-organizational case management system using dynamic condition response graphs, in: Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International, pp. 161 –170.

[35] M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: Proceedings of the 2006 international conference on Business Process Management Workshops, BPM'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 169–180.

[36] F. Maria Maggi, M. Montali, M. Westergaard, W. M. P. van der Aalst, Monitoring business constraints with linear temporal logic: An approach based on colored automata, in: Business Process Management (BPM) 2011, volume 6896 of *Lecture Notes in Computer Science*, pp. 32–147.

[37] A. ter Hofstede, R. van Glabbeek, D. Stork, Query nets: Interacting workflow modules that ensure global termination, in: Business Process Management, Springer Berlin / Heidelberg, 2003, pp. 184–199.

[38] W. van der Aalst, Inheritance of interorganizational workflows: How to agree to disagree without loosing control?, Information Technology and Management 4 (2003) 345–389.

[39] A. Martens, Analyzing web service based business processes, in: Proceedings of the 8th international conference, held as part of the joint European Conference on Theory and Practice of Software conference on Fundamental Approaches to Software Engineering, FASE'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 19–33.

[40] X. Fu, T. Bultan, J. Su, Realizability of conversation protocols with message contents, in: Proceedings of the IEEE International Conference on Web Services, ICWS '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 96–.

[41] X. Yi, K. Kochut, Process composition of web services with complex conversation protocols., in: Design, Analysis, and Simulation of Distributed Systems Symposium at Adavanced Simulation Technology.

[42] S. Rinderle, A. Wombacher, M. Reichert, Evolution of process choreographies in dychor, in: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, volume 4275 of *LNCS*, Springer, 2006, pp. 273–290.

[43] D. Wodtke, G. Weikum, A formal foundation for distributed workflow execution based on state charts, in: Proceedings of the 6th International Conference on Database Theory, Springer-Verlag, London, UK, 1997, pp. 230–246.

[44] M. Bravetti, G. Zavattaro, Contract based multi-party service composition, in: International Symposium on Fundamentals of Software Engineering (FSEN), volume 4767, Springer, 2007, pp. 207–222.

[45] M. Bravetti, G. Zavattaro, A theory of contracts for strong service compliance, Mathematical. Structures in Comp. Sci. 19 (2009) 601–638.

[46] M. Carbone, K. Honda, N. Yoshida, Structured Communication-Centred Programming for Web Services, in: 16th European Symposium on Programming (ESOP'07), LNCS, Springer, 2007, pp. 2–17.

[47] W. Fdhila, C. Godart, Toward synchronization between decentralized orchestrations of composite web services., in: CollaborateCom'09, pp. 1–10.

[48] M. G. Nanda, S. Chandra, V. Sarkar, Decentralizing execution of composite web services, SIGPLAN Not. 39 (2004) 170–187.

[49] R. Khalaf, F. Leymann, Role-based decomposition of business processes using BPEL, in: Web Services, 2006. ICWS '06. International Conference on, pp. 770 –780.

[50] S. Mitra, R. Kumar, S. Basu, Optimum decentralized choreography for web services composition, in: Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2, pp. 395 –402.

[51] OASIS WSBPEL Technical Committee, Web Services Business Process Execution Language, version 2.0, 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`.

[52] I. Castellani, M. Mukund, P. Thiagarajan, Synthesizing distributed transition systems from global specifications, in: Foundations of Software Technology and Theoretical Computer Science, volume 1738, Springer Berlin / Heidelberg, 1999, pp. 219–231.

[53] K. Heljanko, A. Stefanescu, Complexity results for checking distributed implementability, in: Proceedings of the Fifth International Conference on Application of Concurrency to System Design, pp. 78–87.

[54] M. Mukund, From global specifications to distributed implementations, in: Synthesis and Control of Discrete Event Systems, Springer, 2002, pp. 19–35.

[55] W. Zielonka, Notes on finite asynchronous automata., Informatique Théorique et Applications 21(2) (1987) 99–135.

[56] Z. Milosevic, S. Sadiq, M. Orlowska, Towards a methodology for deriving contract-compliant business processes, in: Business Process Management, volume 4102 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pp. 395–400.

[57] W. Sadiq, S. Sadiq, K. Schulz, Model driven distribution of collaborative business processes, in: Services Computing, 2006. SCC '06. IEEE International Conference on, pp. 281 –284.

[58] W. Fdhila, U. Yildiz, C. Godart, A flexible approach for automatic process decentralization using dependency tables, International Conference on Web Services (2009).

[59] G. Dong, R. Hull, B. Kumar, J. Su, G. Zhou, A framework for optimizing distributed workflow executions, in: Revised Papers from the 7th International Workshop on Database Programming Languages: Research Issues in Structured and Semistructured Database Programming, DBPL '99, Springer-Verlag, London, UK, 2000, pp. 152–167.

[60] D. Fahland, Towards analyzing declarative workflows, in: J. Koehler, M. Pistore, A. P. Sheth, P. Traverso, M. Wirsing (Eds.), Autonomous and Adaptive Web Services, volume 07061 of *Dagstuhl Seminar Proceedings*, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007, p. 6.

[61] M. Montali, Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach, volume 56 of *Lecture Notes in Business Information Processing*, Springer, 2010.

[62] P. Bellini, R. Mattolini, P. Nesi, Temporal logics for real-time system specification, ACM Comput. Surv. 32 (2000) 12–42.

[63] P. M. Merlin, A study of the recoverability of computing systems., Ph.D. thesis, University of California, Irvine, 1974. AAI7511026.

[64] B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using time petri nets, Software Engineering, IEEE Transactions on 17 (1991) 259 –273.

[65] L. Popova-Zeugmann, On time petri nets, Elektronische Informationsverarbeitung und Kybernetik 27 (1991) 227–244.

[66] L. Popova-Zeugmann, D. Schlatter, Analyzing paths in time petri nets, Fundam. Inform. 37 (1999) 311–327.

[67] F. Cassez, O. H. Roux, Structural translation from time petri nets to timed automata, Electr. Notes Theor. Comput. Sci. 128 (2005) 145–160.

[68] S. Ling, H. Schmidt, Time petri nets for workflow modelling and analysis, in: Systems, Man, and Cybernetics, 2000 IEEE International Conference on, volume 4, pp. 3039 –3044 vol.4.

[69] C. Ramchandani, Analysis of asynchronous concurrent systems by timed petri nets, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[70] W. M. Zuberek, Timed petri nets and preliminary performance evaluation, in: Proceedings of the 7th annual symposium on Computer Architecture, ISCA '80, ACM, New York, NY, USA, 1980, pp. 88–96.

[71] B. Walter, Timed petri-nets for modelling and alalyzing protocols with real-time characteristics, in: Protocol Specification, Testing, and Verification, pp. 149–159.

[72] H.-M. Hanisch, Analysis of place/transition nets with timed arcs and its application to batch process control, in: M. Ajmone Marsan (Ed.), Application and Theory of Petri Nets 1993, volume 691 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1993, pp. 282–299.

A. Proofs to propositions in Sec. 5

Lemma A.1. $e \bullet\!\!\rightarrow_{|\delta} = \{e' \mid e \bullet\!\!\rightarrow e' \wedge e' \in (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\, \delta_{\mathsf{E}})\}$

Proof *According to def 5.1-vi, the response relation in projected graph is*
$\bullet\!\!\rightarrow_{|\delta} = \bullet\!\!\rightarrow \cap((\bullet\!\!\rightarrow \rightarrow\!\!\diamond\, \delta_{\mathsf{E}}) \times (\rightarrow\!\!\diamond\, \delta_{\mathsf{E}})) \cup ((\bullet\!\!\rightarrow \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}}))$.
Informally it contains relations which can cause a response on an event which is either included in the set of events in the project parameter (δ_{E}) *or in a set of events which are milestones to events in project parameter* $(\rightarrow\!\!\diamond\, \delta_{\mathsf{E}})$.
$\bullet\!\!\rightarrow_{|\delta} = \{(e'', e') \mid e'' \bullet\!\!\rightarrow e' \wedge e' \in (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\, \delta_{\mathsf{E}})\}$ *and hence*
$e \bullet\!\!\rightarrow_{|\delta} = \{e' \mid e \bullet\!\!\rightarrow e' \wedge e' \in (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\, \delta_{\mathsf{E}})\}$

Lemma A.2. $e{\rightarrow}{+}_{|\delta}{=}e{\rightarrow}{+}\cap(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})$

Proof *According to def 5.1-ix, the include relation in projected graph is*
$\rightarrow{+}_{|\delta}{=}{\rightarrow}{+}\cap\Big(\big(({\rightarrow}{+}\,\delta_{\mathsf{E}})\times\delta_{\mathsf{E}}\big)\cup\big(({\rightarrow}{+}{\rightarrow}{\bullet}\,\delta_{\mathsf{E}})\times({\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\big)\cup\big(({\rightarrow}{+}{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\times({\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\big)\Big)$
$\rightarrow{+}_{|\delta}{=}\ \{(e'',e')\mid e''\rightarrow{+}\,e'\wedge e'\in(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\}$
$e{\rightarrow}{+}_{|\delta}{=}\ \{e'\mid e\rightarrow{+}\,e'\wedge e'\in(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\}$
$e{\rightarrow}{+}_{|\delta}{=}e{\rightarrow}{+}\cap(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})$

Lemma A.3. $e{\rightarrow}\%_{|\delta}{=}e{\rightarrow}\%\cap(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})$

Proof *According to def 5.1-x, the exclude relation in projected graph is*
$\rightarrow\%_{|\delta}{=}{\rightarrow}\%\cap\Big(\big(({\rightarrow}\%\,\delta_{\mathsf{E}})\times\delta_{\mathsf{E}}\big)\cup\big(({\rightarrow}\%{\rightarrow}{\bullet}\,\delta_{\mathsf{E}})\times({\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\big)\cup\big(({\rightarrow}\%{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\times({\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\big)\Big)$
$\rightarrow\%_{|\delta}{=}\ \{(e'',e')\mid e''\rightarrow\%\,e'\wedge e'\in(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\}$
$e{\rightarrow}\%_{|\delta}{=}\ \{e'\mid e\rightarrow\%\,e'\wedge e'\in(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})\}$
$e{\rightarrow}\%_{|\delta}{=}e{\rightarrow}\%\cap(\delta_{\mathsf{E}}\cup{\rightarrow}{\bullet}\,\delta_{\mathsf{E}}\cup{\rightarrow}{\diamond}\,\delta_{\mathsf{E}})$

Proposition A.1. *Let* $G=(\mathsf{E},\mathsf{M},{\rightarrow}{\bullet},{\bullet}{\rightarrow},{\rightarrow}{\diamond},{\rightarrow}{+},{\rightarrow}\%,\mathsf{L},l)$ *be a DCR Graph and* $G_{|\delta}$ *its projection with respect to a projection parameter* $\delta=(\delta_{\mathsf{E}},\delta_{\mathsf{L}})$. *Then, for* $e\in\delta_{\mathsf{E}}$ *and* $a\in\delta_{\mathsf{L}}$ *it holds that* $\mathsf{M}\vdash_G e\wedge\mathsf{M}\oplus_G e=\mathsf{M}'\wedge\mathsf{M}'_{|\delta}=\mathsf{M}''$ *if and only if* $\mathsf{M}_{|\delta}\vdash_{G_{|\delta}} e\wedge\mathsf{M}_{|\delta}\oplus_{G_{|\delta}} e=\mathsf{M}''$.

Proof *In order to prove the proposition, we have to show that the proposition in both directions.*

(G→P) *for* $e\in\delta_{\mathsf{E}}$ *and* $a\in\delta_{\mathsf{L}}$. $\mathsf{M}\vdash_G e\wedge\mathsf{M}\oplus_G e=\mathsf{M}'\wedge\mathsf{M}'_{|\delta}=\mathsf{M}''\implies\mathsf{M}_{|\delta}\vdash_{G_{|\delta}} e\wedge\mathsf{M}_{|\delta}\oplus_{G_{|\delta}} e=\mathsf{M}''$.
We will split the proof into 2 steps:

(A) $\mathsf{M}\vdash_G e\implies\mathsf{M}_{|\delta}\vdash_{G_{|\delta}} e$
From def 3.1, we have $\mathsf{M}\vdash_G e\implies e\in\mathsf{In}\wedge(\mathsf{In}\cap{\rightarrow}{\bullet}\,e)\subseteq\mathsf{Ex}$ *and* $(\mathsf{In}\cap{\rightarrow}{\diamond}\,e)\subseteq\mathsf{E}\backslash\mathsf{Re}$.
In order to prove that $\mathsf{M}_{|\delta}\vdash_{G_{|\delta}} e$, *we have to show that*
$e\in\mathsf{In}_{|\delta}\wedge(\mathsf{In}_{|\delta}\cap{\rightarrow}{\bullet}_{|\delta}\,e)\subseteq\mathsf{Ex}_{|\delta}\wedge(\mathsf{In}_{|\delta}\cap{\rightarrow}{\diamond}_{|\delta}\,e)\subseteq\mathsf{E}_{|\delta}\backslash\mathsf{Re}_{|\delta}$. *We will prove each part individually as follows,*

37

(i) *To prove:* $e \in \mathsf{In}_{|\delta}$.
From def 5.1-iiic we have,
$\mathsf{In}_{|\delta} = \mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})$ *therefore*
$e \in \mathsf{In} \wedge e \in \delta_{\mathsf{E}} \implies e \in \mathsf{In}_{|\delta}$.

(ii) *To prove:* $(\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \subseteq \mathsf{Ex}_{|\delta}$.
$\forall e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e)$,

(a) $e' \in \mathsf{In}_{|\delta} \implies e' \in \mathsf{In}$

(b) $e' \in \rightarrow\!\bullet_{|\delta} e \implies e' \in \rightarrow\!\bullet\, e$ *from def 5.1-iv*

Using above 2 statements and from $\mathsf{M} \vdash_G e$
$\forall e'.e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \implies e' \in (\mathsf{In} \cap \rightarrow\!\bullet\, e) \implies e' \in \mathsf{Ex}$,
Further, $(\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \subseteq \mathsf{Ex}_{|\delta}$, *from def 5.1-iiia*

(iii) *To prove:* $(\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta}$
$\forall e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e)$

(a) $e' \in \mathsf{In}_{|\delta} \implies e' \in \mathsf{In}$

(b) $e' \in \rightarrow\!\diamond_{|\delta} e \implies e' \in \rightarrow\!\diamond\, e$, *from def 5.1-viii*

Using above 2 statements and from $\mathsf{M} \vdash_G e$
$e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \implies e' \in (\mathsf{In} \cap \rightarrow\!\diamond\, e) \implies e' \in \mathsf{E} \backslash \mathsf{Re} \implies e' \notin \mathsf{Re}$,
According to def 5.1 iiib, we have $\mathsf{Re}_{|\delta} = \mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})$. *and so* $e' \notin \mathsf{Re} \implies e' \notin \mathsf{Re}_{|\delta}$.
Further, $e' \in \mathsf{E}_{|\delta} \wedge e' \notin \mathsf{Re}_{|\delta} \implies e' \in \mathsf{E}_{|\delta} \setminus \mathsf{Re}_{|\delta}$.
Hence we can conclude that $(\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta}$

From (G→P)-A-i, (G→P)-A-ii and (G→P)-A-iii, we have proved that $e \in \mathsf{In}_{|\delta} \wedge (\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \subseteq \mathsf{Ex}_{|\delta} \wedge (\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta}$ *is valid.*
Therefore we can conclude that $\mathsf{M} \vdash_G e \implies \mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e$.

(B) *To prove:* $\mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}'' \implies \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}''$.
We have $\mathsf{M} \oplus_G e = \mathsf{M}'$ *where* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *and* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *and from the def 3.1, we can infer* $\mathsf{Ex}' = \mathsf{Ex} \cup \{e\}, \mathsf{Re}' = (\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\rightarrow$, *and* $\mathsf{In}' = (\mathsf{In} \cup e\rightarrow\!+) \setminus e\rightarrow\!\%$.
In projected graph, we have $\mathsf{M}_{|\delta} = (\mathsf{Ex}_{|\delta}, \mathsf{Re}_{|\delta}, \mathsf{In}_{|\delta})$, $\mathsf{M}'' = (\mathsf{Ex}''_{|\delta}, \mathsf{Re}''_{|\delta}, \mathsf{In}''_{|\delta})$ *and from above result we know that* $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e$. *Hence we can infer that* $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}_{|\delta} \cup \{e\}, \mathsf{Re}''_{|\delta} = (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e\bullet\!\rightarrow_{|\delta}$, *and* $\mathsf{In}''_{|\delta} = (\mathsf{In}_{|\delta} \cup e\rightarrow\!+_{|\delta}) \setminus e\rightarrow\!\%_{|\delta}$.
We have to prove that $\mathsf{M}'_{|\delta} = \mathsf{M}''$. *In order to prove this equivalence,*

we will show that $\mathsf{Ex}'_{|\delta} = \mathsf{Ex}''_{|\delta}$, $\mathsf{Re}'_{|\delta} = \mathsf{Re}''_{|\delta}$ *and* $\mathsf{In}'_{|\delta} = \mathsf{In}''_{|\delta}$ *individually as follows,*

(i) *To prove:* $\mathsf{Ex}'_{|\delta} = \mathsf{Ex}''_{|\delta}$.
$\mathsf{Ex}'_{|\delta} = (\mathsf{Ex} \cup \{e\}) \cap \mathsf{E}_{|\delta}$ *from def 5.1-iiia*
$= (\mathsf{Ex} \cap \mathsf{E}_{|\delta}) \cup (\{e\} \cap \mathsf{E}_{|\delta})$ *distributive law of sets*
$= \mathsf{Ex}_{|\delta} \cup \{e\}$ *according to def 5.1-iiia and* $e \in \delta_{\mathsf{E}} \subseteq \mathsf{E}_{|\delta}$.
$= \mathsf{Ex}''_{|\delta}$.
Hence we can conclude that $\mathsf{Ex}'_{|\delta} = \mathsf{Ex}''_{|\delta}$.

(ii) *To prove:* $\mathsf{Re}'_{|\delta} = \mathsf{Re}''_{|\delta}$.
$\mathsf{Re}'_{|\delta} = \big((\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\!\rightarrow\ \big) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})$ *from def 5.1-iiib*
$= \big((\mathsf{Re} \setminus \{e\}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \cup \big(\ e\bullet\!\!\rightarrow\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big)$ *distributive law*
$= \big(\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}}) \setminus (\{e\} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \cup \big(\ e\bullet\!\!\rightarrow\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big)$ *set intersection distributes over set difference*
$= (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup \big(\ e\bullet\!\!\rightarrow\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big)$
$= (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup \{e' \mid e \bullet\!\!\rightarrow e' \wedge e' \in (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\}$
$= (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e\bullet\!\!\rightarrow_{|\delta}$ *using lemma A.1*
$= \mathsf{Re}''_{|\delta}$
Hence we can conclude that $\mathsf{Re}'_{|\delta} = \mathsf{Re}''_{|\delta}$.

(iii) *To prove:* $\mathsf{In}'_{|\delta} = \mathsf{In}''_{|\delta}$
$\mathsf{In}'_{|\delta} = \mathsf{In}' \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})$, *from def 5.1-iiic*
But we know that $\mathsf{In}' = (\mathsf{In} \cup e\rightarrow\!+) \setminus e\rightarrow\!\%$
$\mathsf{In}'_{|\delta} = \big((\mathsf{In} \cup e\rightarrow\!+) \setminus e\rightarrow\!\%\ \big) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})$
$\mathsf{In}'_{|\delta} = \big((\mathsf{In} \cup e\rightarrow\!+) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \setminus \big(\ e\rightarrow\!\%\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big)$ *set intersection distributes over set difference*
$\mathsf{In}'_{|\delta} = \big((\mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \cup \big(\ e\rightarrow\!+\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \setminus \big(\ e\rightarrow\!\%\ \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big)$ *distributive law*
$\mathsf{In}'_{|\delta} = \Big(\big(\mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})\big) \cup e\rightarrow\!+_{|\delta}\ \Big) \setminus e\rightarrow\!\%_{|\delta}$ *using lemmas A.2 and A.3*
But we know that the marking in projected graph before executing event e *is* $\mathsf{In}_{|\delta} = \mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})$. *Using this fact, we can rewrite the above statement as follows,*
$\mathsf{In}'_{|\delta} = (\mathsf{In}_{|\delta} \cup e\rightarrow\!+_{|\delta}) \setminus e\rightarrow\!\%_{|\delta}$
$\mathsf{In}'_{|\delta} = \mathsf{In}''_{|\delta}$

Hence we can conclude that $\mathsf{In}'_{|\delta} = \mathsf{In}''_{|\delta}$.

From (G→P)-B-i, (G→P)-B-ii and (G→P)-B-iii, we have proved that $\mathsf{Ex}'_{|\delta} = \mathsf{Ex}''_{|\delta}$, $\mathsf{Re}'_{|\delta} = \mathsf{Re}''_{|\delta}$ *and* $\mathsf{In}'_{|\delta} = \mathsf{In}''_{|\delta}$. *and there by we can conclude that* $\mathsf{M}'_{|\delta} = \mathsf{M}''$.

Since we have proved both parts: ((G→P)-A and (G→P)-B), the proposition $\mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}'' \implies \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}''$ *holds.*

(P→G) for $e \in \delta_{\mathsf{E}}$ *and* $a \in \delta_{\mathsf{L}}$. $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}'' \implies \mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}''$
Again, we will split the proof into 2 parts.

(A) $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \implies \mathsf{M} \vdash_G e$
From def 3.1, we have $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \implies e \in \mathsf{In}_{|\delta} \wedge (\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \subseteq \mathsf{Ex}_{|\delta} \wedge (\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta}$
In order to prove that $\mathsf{M} \vdash_G e$, *we have to show that*
$e \in \mathsf{In} \wedge (\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$ *and* $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$

(i) To prove: $e \in \mathsf{In}$
From def 5.1-iiic we have: $\mathsf{In}_{|\delta} = \mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\, \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$
$e \in \mathsf{In}_{|\delta} \wedge \big(\mathsf{In}_{|\delta} = \mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\, \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})\big) \implies e \in \mathsf{In}$.

(ii) To prove: $(\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$
From def 5.1-iv, we have $\rightarrow\!\bullet_{|\delta} = \rightarrow\!\bullet \cap ((\rightarrow\!\bullet\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$
$\forall e'.e' \in \rightarrow\!\bullet_{|\delta} e \implies (e', e) \in \rightarrow\!\bullet_{|\delta} \implies (e', e) \in \rightarrow\!\bullet \implies e' \in \rightarrow\!\bullet e$ *and therefore* $\rightarrow\!\bullet_{|\delta} e = \rightarrow\!\bullet e$.
$\forall e'.e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \implies (e' \in \mathsf{In}_{|\delta}) \cap (e' \in \rightarrow\!\bullet_{|\delta} e) \implies (e' \in \mathsf{In}) \cap (e' \in \rightarrow\!\bullet e) \implies e' \in (\mathsf{In} \cap \rightarrow\!\bullet e)$, *and hence*
$(\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) = (\mathsf{In} \cap \rightarrow\!\bullet e)$.
$(\mathsf{In}_{|\delta} \cap \rightarrow\!\bullet_{|\delta} e) \subseteq \mathsf{Ex}_{|\delta} \implies (\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}_{|\delta}$.
according to def 5.1-iiia : $\mathsf{Ex}_{|\delta} = \mathsf{Ex} \cap \mathsf{E}_{|\delta}$.
Hence $(\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}_{|\delta} \implies (\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$

(iii) To prove: $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$
From def 5.1-viii, we have $\rightarrow\!\diamond_{|\delta} = \rightarrow\!\diamond \cap ((\rightarrow\!\diamond\, \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$,
$\forall e'.e' \in \rightarrow\!\diamond_{|\delta} e \implies (e', e) \in \rightarrow\!\diamond_{|\delta} \implies (e', e) \in \rightarrow\!\diamond \implies e' \in \rightarrow\!\diamond e$ *and therefore* $\rightarrow\!\diamond_{|\delta} e = \rightarrow\!\diamond e$.
$\forall e'.e' \in (\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) \implies (e' \in \mathsf{In}_{|\delta}) \cap (e' \in \rightarrow\!\diamond_{|\delta} e) \implies (e' \in \mathsf{In}) \cap (e' \in \rightarrow\!\diamond e) \implies e' \in (\mathsf{In} \cap \rightarrow\!\diamond e)$, *and hence*
$(\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta} e) = (\mathsf{In} \cap \rightarrow\!\diamond e)$.

$(\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta}\, e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta} \implies (\mathsf{In} \cap \rightarrow\!\diamond\, e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta} \implies$
$\forall e' \in (\mathsf{In} \cap \rightarrow\!\diamond e).e' \notin \mathsf{Re}_{|\delta}$.
according to def 5.1-iiib : $\mathsf{Re}_{|\delta} = \mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$,
$\forall e' \in (\mathsf{In} \cap \rightarrow\!\diamond\, e).e' \notin \mathsf{Re}_{|\delta} \implies e' \notin (\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}))$.
Further, as $e' \rightarrow\!\diamond\, e$*, we know that* $e' \in (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$*. The only way* $e' \notin (\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}))$ *becomes true is when* $e' \notin \mathsf{Re}$.
Hence $(\mathsf{In}_{|\delta} \cap \rightarrow\!\diamond_{|\delta}\, e) \subseteq \mathsf{E}_{|\delta} \backslash \mathsf{Re}_{|\delta} \implies (\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

Form (P→G)-A-(i), (P→G)-A-(ii) and (P→G)-A-(iii), we can conclude that $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \implies \mathsf{M} \vdash_G e$.

(B) $\mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}'' \implies \mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}''$
We have $\mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}''$ *in the local graph where* $\mathsf{M}_{|\delta} = (\mathsf{Ex}_{|\delta}, \mathsf{Re}_{|\delta}, \mathsf{In}_{|\delta})$, $\mathsf{M}'' = (\mathsf{Ex}''_{|\delta}, \mathsf{Re}''_{|\delta}, \mathsf{In}''_{|\delta})$ *and from the def 3.1, we can infer*
$\mathsf{Ex}''_{|\delta} = \mathsf{Ex}_{|\delta} \cup \{e\}, \mathsf{Re}''_{|\delta} = (\mathsf{Re}_{|\delta} \backslash \{e\}) \cup e\!\bullet\!\!\rightarrow_{|\delta}$, *and* $\mathsf{In}''_{|\delta} = (\mathsf{In}_{|\delta} \cup e\!\rightarrow\!+_{|\delta}) \backslash\, e\!\rightarrow\!\%_{|\delta}$.
In main graph, we know $\mathsf{M} \vdash_G e$ *where* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *and hence we can workout the new marking as* $\mathsf{M} \oplus_G e = \mathsf{M}'$ *where* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *with* $\mathsf{Ex}' = \mathsf{Ex} \cup \{e\}, \mathsf{Re}' = (\mathsf{Re} \setminus \{e\}) \cup e\!\bullet\!\!\rightarrow$, *and* $\mathsf{In}' = (\mathsf{In} \cup e\!\rightarrow\!+) \backslash\, e\!\rightarrow\!\%$.
We have to prove that $\mathsf{M}'' = \mathsf{M}'_{|\delta}$.

(i) To prove: $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$
Let us start with $\mathsf{Ex}''_{|\delta}$
$\mathsf{Ex}''_{|\delta} = \mathsf{Ex}_{|\delta} \cup \{e\}$
$= (\mathsf{Ex} \cap \mathsf{E}_{|\delta}) \cup \{e\}$ *from def 5.1-iiia*
$= (\mathsf{Ex} \cup \{e\}) \cap (\mathsf{E}_{|\delta} \cup \{e\})$
$= \mathsf{Ex}' \cap \mathsf{E}_{|\delta}$
$= \mathsf{Ex}'_{|\delta}$
Hence we can conclude that $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$.

(ii) To prove: $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$
Let us start with $\mathsf{Re}''_{|\delta}$
$\mathsf{Re}''_{|\delta} = (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e\!\bullet\!\!\rightarrow_{|\delta}$
$= \big((\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})) \setminus \{e\}\big) \cup e\!\bullet\!\!\rightarrow_{|\delta}$ *from def 5.1-iiib*
$= \big((\mathsf{Re} \setminus \{e\}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})\big) \cup e\!\bullet\!\!\rightarrow_{|\delta}$ *(set relative complements)*
$= \big((\mathsf{Re} \setminus \{e\}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})\big) \cup (e\!\bullet\!\!\rightarrow \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}))$ *using lemma A.1*
$= \big((\mathsf{Re} \setminus \{e\}) \cup e\!\bullet\!\!\rightarrow\big) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$
$= \mathsf{Re}' \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$

$= \mathsf{Re}'_{|\delta}$ *according to def 5.1-iiib.*
Hence we can conclude that $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$.

(iii) To prove: $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$

Let us starts with $\mathsf{In}''_{|\delta}$ *and show that it will be equal to the projection over included set from global graph* $(\mathsf{In}'_{|\delta})$.
$\mathsf{In}''_{|\delta} = (\mathsf{In}_{|\delta} \cup\ e{\rightarrow}{+}_{|\delta}) \setminus\ e{\rightarrow}\%_{|\delta}$
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})) \cup\ e{\rightarrow}{+}_{|\delta}\ \big) \setminus\ e{\rightarrow}\%_{|\delta}$, *from def 5.1-iiic.*
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})) \cup (e{\rightarrow}{+} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}}))\big) \setminus (e{\rightarrow}\% \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}}))$ *using lemmas A.2 and A.3*
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cup\ e{\rightarrow}{+}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})\big) \setminus (e{\rightarrow}\% \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}}))$.
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cup\ e{\rightarrow}{+}) \setminus\ e{\rightarrow}\%\ \big) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})$.
$\mathsf{In}''_{|\delta} = (\mathsf{In}') \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\bullet\ \delta_{\mathsf{E}} \cup \rightarrow\!\diamond\ \delta_{\mathsf{E}})$.
$\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$.
Hence we can conclude that $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$.

From (P→G)-B-i, (P→G)-B-ii and (P→G)-B-iii, we have proved that $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$, $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$ *and* $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$ *and there by we can conclude that* $\mathsf{M}'' = \mathsf{M}'_{|\delta}$.

Since we have proved both parts: ((P→G)-A and (P→G)-B), the proposition for $e \in \delta_{\mathsf{E}}$ *and* $a \in \delta_{\mathsf{L}}$. $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}'' \implies \mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}''$ *holds.*

Finally, we have proved the proposition in both ways $\big($*(G→P) and (P→G)*$\big)$, *therefore the proposition: for* $e \in \delta_{\mathsf{E}}$ *and* $a \in \delta_{\mathsf{L}}$ *it holds that* $\mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}' \wedge \mathsf{M}'_{|\delta} = \mathsf{M}''$ *if and only if* $\mathsf{M}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}''$ *holds.*

Proposition A.2. *Let* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *be a DCR Graph and* $G_{|\delta}$ *its projection with respect to a projection parameter* $\delta = (\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$. *Then, for* $e \notin \mathsf{E}_{|\delta}$ *it holds that* $\mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}'$ *implies* $\mathsf{M}_{|\delta} = \mathsf{M}'_{|\delta}$.

Proof *According to projection definition 5.1,* $e \notin \mathsf{E}_{|\delta} \implies e \notin G_{|\delta}$, *therefore there will not be any change in the marking. Hence* $\mathsf{M}_{|\delta} = \mathsf{M}'_{|\delta}$.

Proposition A.3. *Let* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *be a DCR Graph and* $G_{|\delta}$ *its projection with respect to a projection parameter* $\delta = (\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$*. Then for* $e \in \mathsf{E}_{|\delta}$ *(and* $a \notin \delta_{\mathsf{L}}$*) it holds that* $\mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}'$ *implies* $\mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}'_{|\delta}$.

Proof *The proof for this proposition is more or less similar to proof in the part (P→G)-(B) of proposition A.1 with minor changes.*

We have $\mathsf{M} \oplus_G e = \mathsf{M}'$ *where* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *and* $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ *and from the def 3.1, we can infer* $\mathsf{Ex}' = \mathsf{Ex} \cup \{e\}, \mathsf{Re}' = (\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\rightarrow$*, and* $\mathsf{In}' = (\mathsf{In} \cup e\rightarrow\!+) \setminus e\rightarrow\!\%$.
In projected graph, we have marking projected from global graph, according to def 5.1 as $\mathsf{M}_{|\delta} = (\mathsf{Ex}_{|\delta}, \mathsf{Re}_{|\delta}, \mathsf{In}_{|\delta})$*. The result of executing event e in projected graph will be a marking, let us say* $\mathsf{M}''_{|\delta} = \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e$*, then we have to prove that* $\mathsf{M}''_{|\delta} = \mathsf{M}'_{|\delta}$.
Let us say that $\mathsf{M}''_{|\delta} = (\mathsf{Ex}''_{|\delta}, \mathsf{Re}''_{|\delta}, \mathsf{In}''_{|\delta})$*, and since in the projected graph we have* $\mathsf{M}''_{|\delta} = \mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e$*, we can infer from the def 3.1,* $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}_{|\delta} \cup \{e\}, \mathsf{Re}''_{|\delta} = (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e\bullet\!\rightarrow_{|\delta}$*, and* $\mathsf{In}''_{|\delta} = (\mathsf{In}_{|\delta} \cup e\rightarrow\!+_{|\delta}) \setminus e\rightarrow\!\%_{|\delta}$.
In order to prove this equivalence of $\mathsf{M}''_{|\delta} = \mathsf{M}'_{|\delta}$*, we will show that* $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$, $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$ *and* $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$ *individually as follows,*

(i) *To prove:* $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$
Let us start with $\mathsf{Ex}''_{|\delta}$
$\mathsf{Ex}''_{|\delta} = \mathsf{Ex}_{|\delta} \cup \{e\}$
$= (\mathsf{Ex} \cap \mathsf{E}_{|\delta}) \cup \{e\}$ *from def 5.1-iiia*
$= (\mathsf{Ex} \cup \{e\}) \cap (\mathsf{E}_{|\delta} \cup \{e\})$
$= \mathsf{Ex}' \cap \mathsf{E}_{|\delta}$
$= \mathsf{Ex}'_{|\delta}$
Hence we can conclude that $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$.

(ii) *To prove:* $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$
Let us start with $\mathsf{Re}''_{|\delta}$
$\mathsf{Re}''_{|\delta} = (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e\bullet\!\rightarrow_{|\delta}$
$= \big((\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})) \setminus \{e\}\big) \cup e\bullet\!\rightarrow_{|\delta}$ *from def 5.1-iiib*
$= \big((\mathsf{Re} \setminus \{e\}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})\big) \cup e\bullet\!\rightarrow_{|\delta}$ *(set relative complements)*
$= \big((\mathsf{Re} \setminus \{e\}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})\big) \cup (e\bullet\!\rightarrow \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}))$ *using lemma A.1*
$= \big((\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\rightarrow\big) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$
$= \mathsf{Re}' \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}})$

$= \mathsf{Re}'_{|\delta}$ *according to def 5.1-iiib.*
Hence we can conclude that $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$.

(iii) To prove: $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$
Let us starts with $\mathsf{In}''_{|\delta}$ *and show that it will be equal to the projection over included set from global graph* $(\mathsf{In}'_{|\delta})$.
$\mathsf{In}''_{|\delta} = (\mathsf{In}_{|\delta} \cup e \rightarrow+_{|\delta}) \setminus e \rightarrow\%_{|\delta}$
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E})) \cup e \rightarrow+_{|\delta}\ \big) \setminus e \rightarrow\%_{|\delta}$ *from def 5.1-iiic.*
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E})) \cup (e \rightarrow+ \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E}))\big) \setminus (e \rightarrow\% \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E}))$ *using lemmas A.2 and A.3*
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cup e \rightarrow+) \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E})\big) \setminus (e \rightarrow\% \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E}))$.
$\mathsf{In}''_{|\delta} = \big((\mathsf{In} \cup e \rightarrow+) \setminus e \rightarrow\%\ \big) \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E})$.
$\mathsf{In}''_{|\delta} = (\mathsf{In}') \cap (\delta_\mathsf{E} \cup \rightarrow\bullet\ \delta_\mathsf{E} \cup \rightarrow\diamond\ \delta_\mathsf{E})$.
$\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$.
Hence we can conclude that $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$.

From (i), (ii) and (iii), we have proved that $\mathsf{Ex}''_{|\delta} = \mathsf{Ex}'_{|\delta}$, $\mathsf{Re}''_{|\delta} = \mathsf{Re}'_{|\delta}$ *and* $\mathsf{In}''_{|\delta} = \mathsf{In}'_{|\delta}$ *and there by we can conclude that* $\mathsf{M}'' = \mathsf{M}'_{|\delta}$.
Therefore the proposition: for $e \in \mathsf{E}_{|\delta}$ *(and* $a \notin \delta_\mathsf{L}$*) it holds that* $\mathsf{M} \vdash_G e \wedge \mathsf{M} \oplus_G e = \mathsf{M}'$ *implies* $\mathsf{M}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M}'_{|\delta}$ *is proved.*

Proof of Proposition 5.1.1. Let $G = (\mathsf{E}, \mathsf{M_t}, \rightarrow\bullet, t_c, \bullet\rightarrow, t_r, \rightarrow\diamond, \rightarrow+, \rightarrow\%, \mathsf{L}, l)$ be a Timed DCR Graph and $G_{|\delta}$ its projection with respect to a projection parameter $\delta = (\delta_\mathsf{E}, \delta_\mathsf{L})$. Then, for $e \in \delta_\mathsf{E}$ and $a \in \delta_\mathsf{L}$ it holds that $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}''$ if and only if $\mathsf{M_t}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M_t}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$.
In order to prove the proposition, we have to show that the proposition in both directions.

(G→P) for $e \in \delta_\mathsf{E}$ and $a \in \delta_\mathsf{L}$. $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}'' \implies \mathsf{M_t}_{|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M_t}_{|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$.
We will split the proof into 2 steps:

(A) $\mathsf{M_t} \vdash_G e \implies \mathsf{M_t}_{|\delta} \vdash_{G_{|\delta}} e$
From def 3.3, we have
$\mathsf{M_t} \vdash_G e \implies \mathsf{M} \vdash_{G'} e \wedge \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex}(e')$.
$\mathsf{M} \vdash_{G'} e$ has been proved in the (G→P)(A) part of the proof of proposition A.1, therefore we have to show that $\forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq$

$t_{ex}(e')$ implies that $\forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex|\delta}(e')$, where $e' \overset{k}{\rightarrow\bullet}_{|\delta} e =_{def} e' \rightarrow\bullet_{|\delta} e$ and $t_{c|\delta}(e, e') = k$.

$\forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex}(e')$
$= \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex}(e')$, from def 5.1-iv
$= \forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex}(e')$, from def 5.1-iiic
$= \forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq (t_{ex}(e')$ if $e' \in \mathsf{Ex})$, by definition of t_{ex}
$= \forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq (t_{ex}(e')$ if $e' \in \mathsf{Ex} \cap \delta_{\mathsf{E}})$, since $\mathsf{M}_{|\delta} \vdash_{G'} e \wedge e' \in \mathsf{In}_{|\delta}$, it preserves the t_{ex} behavior
$= \forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq (t_{ex}(e')$ if $e' \in \mathsf{Ex}_{|\delta})$, from def 5.1-iiia
$= \forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex|\delta}(e')$ from def 5.1-iiid.
Hence proved.

(B) To prove: $\mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}'' \implies \mathsf{M}_{\mathsf{t}|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$
From def. 3.3 we have $\mathsf{M_t} \oplus_G e =_{def} \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e, t'_{ex}, t'_{re}\big)$, where

(i) $t'_{ex}(e') = \begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{otherwise} \end{cases}$

(ii) $t'_{re}(e') = \begin{cases} k & \text{if } e \overset{k}{\bullet\rightarrow} e' \\ t_{re}(e') & \text{otherwise} \end{cases}$

We proved the $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e$ part in the (G→P)(B) part of the proof of proposition A.1. We then need to show that $(t_{ex}')_{|\delta} \implies t_{ex}''$ and that $(t_{re}')_{|\delta} \implies t_{re}''$

$(t_{ex}')_{|\delta}(e')$
$= t'_{ex}(e')$ if $e' \in \mathsf{Ex}'_{|\delta}$, according to def 5.1-iiid
$= \left(\begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{otherwise} \end{cases}\right)$ if $e' \in (\mathsf{Ex} \cup \{e\}) \cap \mathsf{E}_{|\delta}$, according to def 3.3, and def 5.1-iiia
$= \begin{cases} 0 & \text{if } e' = e \wedge e' \in (\mathsf{Ex} \cup \{e\}) \cap \mathsf{E}_{|\delta} \\ t_{ex}(e') & \text{if } e' \in (\mathsf{Ex} \cup \{e\}) \cap \mathsf{E}_{|\delta} \end{cases}$, by moving the projection inward

But we know that if $e' = e$, trivially $e \in \mathsf{Ex} \cup \{e\}$ and $e \in \mathsf{E}_{|\delta}$, by def 5.1-i. Moreover, the second branch will be used only if $e' \neq e$, therefore
$= \begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{if } e' \in (\mathsf{Ex} \cap \mathsf{E}_{|\delta}) \end{cases}$
$= {t_{ex}}''(e')$, according to def 5.1-iiia, and def 3.3

$({t_{re}}')_{|\delta}(e')$
$= t'_{re}(e')$ if $e' \in \mathsf{Re}'_{|\delta}$, according to def 5.1-iiie
$= \left(\begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow} e' \\ t_{re}(e') & \text{otherwise} \end{cases} \right)$ if $e' \in \mathsf{Re}'_{|\delta}$, according to def 3.3
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \wedge e' \in \mathsf{Re}'_{|\delta} \\ t_{re}(e') & \text{if } e' \in \mathsf{Re}'_{|\delta} \end{cases}$, according to def 5.1-vi and moving the projection inward
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \wedge e' \in ((\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\!\rightarrow) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}) \\ t_{re}(e') & \text{if } e' \in ((\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\!\rightarrow) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}) \end{cases}$,
according to def 5.1-iiib and def 3.1
But we know that if $e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e'$, trivially $e' \in e \bullet\!\!\rightarrow$, and also that the second branch will be used only if $(e, e') \notin \bullet\!\!\rightarrow_{|\delta}$, therefore
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \\ t_{re}(e') & \text{if } e' \in \mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond\, \delta_{\mathsf{E}}) \end{cases}$
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \\ t_{re|\delta}(e') & \text{otherwise} \end{cases}$, by def 5.1-vi, and def 5.1-iiie
$= {t_{re}}''(e')$, according to def 3.3

Since we have proved both parts: ((G→P)-A and (G→P)-B), the proposition $\mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M}'_{\mathsf{t}|\delta} = \mathsf{M_t}'' \implies \mathsf{M}_{\mathsf{t}|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$ holds.

(P→G) for $e \in \delta_{\mathsf{E}}$ and $a \in \delta_{\mathsf{L}}$. $\mathsf{M}_{\mathsf{t}|\delta} \vdash_{G_{|\delta}} e \wedge \mathsf{M}_{\mathsf{t}|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}'' \implies \mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M}'_{\mathsf{t}|\delta} = \mathsf{M_t}''$
Again, we will split the proof into 2 parts.

(A) $\mathsf{M}_{\mathsf{t}|\delta} \vdash_{G_{|\delta}} e \implies \mathsf{M_t} \vdash_G e$

From def 3.3, we have
$\mathsf{M_t} \vdash_G e \implies \mathsf{M} \vdash_{G'} e \wedge \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex}(e')$.
$\mathsf{M}_{|\delta} \vdash_{G'} e$ has been proved in the (P→G)(A) part of the proof of proposition A.1, therefore we have to show that $\forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex|\delta}(e')$ implies that $\forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex}(e')$, where $e' \overset{k}{\rightarrow\bullet} e =_{def} e' \rightarrow\bullet e$ and $t_c(e, e') = k$.

$\forall e' \in \mathsf{In}_{|\delta}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex|\delta}(e')$
$= \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet}_{|\delta} e \implies k \leq t_{ex|\delta}(e')$ from A.1-i
$= \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex|\delta}(e')$ from A.1-ii
$= \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq (t_{ex}(e') \text{ if } e' \in \mathsf{Ex}_{|\delta})$, according to def 5.1-iiid
$= \forall e' \in \mathsf{In}.e' \overset{k}{\rightarrow\bullet} e \implies k \leq t_{ex}(e')$ since $\mathsf{M} \vdash_{G'} e \wedge e' \in \rightarrow\bullet e \wedge e' \in \mathsf{In} \implies e' \in \mathsf{Ex}_{|\delta}$.
Hence proved.

(B) $\mathsf{M}_{\mathsf{t}|\delta} \oplus_{G_{|\delta}} e = \mathsf{M_t}'' \implies \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M}'_{\mathsf{t}|\delta} = \mathsf{M_t}''$
From def. 3.3 we have
$\mathsf{M_t} \oplus_G e =_{def} \big((\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e, t'_{ex}, t'_{re}\big)$, where

(i) $t'_{ex}(e') = \begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{otherwise} \end{cases}$

(ii) $t'_{re}(e') = \begin{cases} k & \text{if } e \overset{k}{\bullet\rightarrow} e' \\ t_{re}(e') & \text{otherwise} \end{cases}$

We proved the $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e$ part in the (P→G)(B) part of the proof of proposition A.1. We then need to show that $t_{ex}'' \implies (t_{ex}')_{|\delta}$ and that $t_{re}'' \implies (t_{re}')_{|\delta}$

$t_{ex}''(e')$
$= \begin{cases} 0 & \text{if } e' = e \\ t_{ex|\delta}(e') & \text{otherwise} \end{cases}$, according to def 3.3 and considering that if $e' \neq e$, the underlying t_{ex} behaves exactly like $t_{ex|\delta}$
$= \begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{if } e' \in \mathsf{Ex}_{|\delta} \end{cases}$, according to def 5.1-iiid

$= \left(\begin{cases} 0 & \text{if } e' = e \\ t_{ex}(e') & \text{otherwise} \end{cases} \right)$ if $e' \in (\mathsf{Ex}_{|\delta}) \cup \{e\}$, since the overall function will continue to behave the same
$= t'_{ex}(e')$ if $e' \in (\mathsf{Ex} \cap \delta_{\mathsf{E}}) \cup \{e\}$, according to def 3.1, and def 5.1-iiia
$= t'_{ex}(e')$ if $e' \in (\mathsf{Ex} \cup \{e\}) \cap (\delta_{\mathsf{E}} \cup \{e\})$
$= t'_{ex}(e')$ if $e' \in \mathsf{Ex}'_{|\delta}$, according to def 3.1, and def 5.1-iiia
$= (t_{ex}{}')_{|\delta}(e')$, according to def 3.3

$t_{re}{}''(e')$
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \\ t_{re|\delta}(e') & \text{otherwise} \end{cases}$, according to def 3.3 and considering that if $(e, e') \not\in \bullet\!\!\rightarrow_{|\delta}$, the underlying t_{re} behaves exactly like $t_{re|\delta}$
$= \begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow}_{|\delta} e' \\ t_{re}(e') & \text{if } e' \in \mathsf{Re}_{|\delta} \end{cases}$, according to def 5.1-iiie
$\left(\begin{cases} k & \text{if } e \overset{k}{\bullet\!\!\rightarrow} e' \\ t_{re}(e') & \text{otherwise} \end{cases} \right)$ if $e' \in (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e \bullet\!\!\rightarrow_{|\delta}$, according to def 5.1-vi, and since the overall function will continue to behave the same
$= t_{re}{}'(e')$ if $e' \in (\mathsf{Re}_{|\delta} \setminus \{e\}) \cup e \bullet\!\!\rightarrow_{|\delta}$, according to def 3.3
$= t_{re}{}'(e')$ if $e' \in ((\mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}}) \setminus \{e\}) \cup e \bullet\!\!\rightarrow_{|\delta})$, according to def 5.1-iiib
$= t_{re}{}'(e')$ if $e' \in ((\mathsf{Re} \setminus \{e\}) \cup e \bullet\!\!\rightarrow_{|\delta}) \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\!\diamond\ \delta_{\mathsf{E}})$
$= t_{re}{}'(e')$ if $e' \in \mathsf{Re}'_{|\delta}$, according to def 3.1, and def 5.1-vi
$= (t_{re}{}')_{|\delta}(e')$, according to def 5.1-iiie

Since we have proved both parts: ((P$\rightarrow$G)-A and (P$\rightarrow$G)-B), the proposition for $e \in \delta_{\mathsf{E}}$ and $a \in \delta_{\mathsf{L}}$. $\mathsf{M_{t|\delta}} \vdash_{G_{|\delta}} e \wedge \mathsf{M_{t|\delta}} \oplus_{G_{|\delta}} e = \mathsf{M_t}'' \implies \mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}''$ holds.

Finally, we have proved the proposition in both ways $\big($(G$\rightarrow$P) and (P$\rightarrow$G)$\big)$, therefore the proposition: for $e \in \delta_{\mathsf{E}}$ and $a \in \delta_{\mathsf{L}}$ it holds that $\mathsf{M_t} \vdash_G e \wedge \mathsf{M_t} \oplus_G e = \mathsf{M_t}' \wedge \mathsf{M_t}'_{|\delta} = \mathsf{M_t}''$ if and only if $\mathsf{M_{t|\delta}} \vdash_{G_{|\delta}} e \wedge \mathsf{M_{t|\delta}} \oplus_{G_{|\delta}} e = \mathsf{M_t}''$ holds.

Proof of Proposition 5.1.2. Follows trivially from result of proposition A.2.

Proof of Proposition 5.1.3. Considering the result of proposition A.3, since the marking of the underlying DCR Graph is the same, this proposition is proved by the result from the part (P$\rightarrow$G)-(B) of proposition 1.

Exformatics Declarative Case Management Workflows as DCR Graphs

Tijs Slaats^{1,2}, Raghava Rao Mukkamala^{1}, Thomas Hildebrandt^{1}, and Morten Marquard^{2} *

^{1} IT University of Copenhagen
Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
{hilde, rao, tslaats}@itu.dk, http://www.itu.dk
^{2} Exformatics A/S
Lautrupsgade 13, 2100 Copenhagen, Denmark
{mmq, ts}@exformatics.com, http://www.exformatics.com

Abstract. Declarative workflow languages have been a growing research subject over the past ten years, but applications of the declarative approach in industry are still uncommon. Over the past two years Exformatics A/S, a Danish provider of Electronic Case Management systems, has been cooperating with researchers at IT University of Copenhagen (ITU) to create tools for the declarative workflow language Dynamic Condition Response Graphs (DCR Graphs) and incorporate them into their products and in teaching at ITU. In this paper we give a status report over the work. We start with an informal introduction to DCR Graphs. We then show how DCR Graphs are being used by Exformatics to model workflows through a case study of an invoice workflow. Finally we give an overview of the tools that have been developed by Exformatics to support working with DCR Graphs and evaluate their use in capturing requirements of workflows and in a bachelor level course at ITU.

Keywords: workflows, declarative specifications, tools, teaching, case study

1 Introduction

Declarative workflow modelling [8, 9, 16] is an emerging field in both academia and industry which offers a new paradigm that supports flexibility and adaptability in business processes. Traditional imperative workflow languages describe *how* a process is carried out as a procedure with explicit control flow. This often leads to rigid and overspecified process descriptions, that fails to capture *why* the activities must be done in the given order. Declarative workflow languages on the other hand specify processes by the constraints describing *why* activities can or must be executed in a particular order, and not how the the process is to be executed, i.e. activities can be executed in any order and any number of times, as long as not prohibited by a constraint [15, 19]. This may lead to under specified process descriptions and make it difficult to perceive the path from

* This research is supported by the Danish Research Agency through an industrial PhD Grant.

start to end, but captures the reason for the ordering of activities and leaves flexibility in execution.

An example a constraint between activities is the response constraint [4, 16] (e.g. $A \bullet\!\!\rightarrow B$), which requires that an execution of one task (A) is eventually followed by an execution of another task (B), but it does not put any further limits on the number of times and order in which the tasks are executed. For example, it would be perfectly valid if the second task occurs first, as long as it also occurs after the first task. In other words, $B, AB, BAB, AAB, \ldots$ are all valid runs, where as $A, BA, BBA, \ldots$ are not valid runs, as they fail to satisfy the constraint by having an occurence of A that is not followed by an occurence of B.

Examples of processes that require more flexibility are commonly found in the healthcare [5] and case management [1] domains. In those processes, the work is being carried out by knowledge workers who typically have the experience and expertise needed to deal with the complexity of a process whose requirements may vary from case to case. For this reason, knowledge-intensive processes require flexible workflow systems that support the users in their work (instead of dictating them what to do) and allow them to make their own choices as long as they do not break those constraints that do need to be strictly followed in all cases (e.g. laws or organizational policies).

Over the last decade, several declarative languages for business processes have been proposed in academic literature. The first of these languages is Declare [15, 19] which gave a number of common workflow constraints formalized in Linear-time Temporal Logic (LTL). More recently, DCR Graphs [4] have been developed as a generalization of event structures [21], where processes are described as a graph of events related by only 4 basic constraints. A simple operational semantics based on markings of the process graph makes it possible to clearly visualize the runtime state. Furthermore, the Guard-Stage-Milestone [10] has been developed, which is a *data-centric* workflow model with declarative elements for modeling life cycles of business artifacts.

Even though by now these techniques have become well known in academia, their application in the industry is relatively uncommon. Over the last two years Exformatics A/S, a Danish provider of Electronic Case Management(ECM) systems, has been collaborating with researchers of IT University of Copenhagen (ITU), to develop tools for the declarative workflow language DCR Graphs with the aim to apply and evaluate the use of DCR Graphs on real world scenarios in the case management domain and in teaching at ITU.

The goal of the present paper is to give a status report, presenting and evaluating the tools developed so far. As the first step, the core DCR Graphs model were used by Exformatics A/S in a case study to capture some of the requirements in the design phase of a cross-organizational case management system [1]. The case study led to the further development of the DCR Graphs model by adding support for *hierarchical* modelling using nested events and a (*milestone*) constraint [6], making it possible to concisely specify that some event(s) must not be pending in order for some event to happen. It also encouraged developing a graphical design, simulation and verification tool [18] which is being used successfully in further case studies with industry and in teaching at ITU.

In the remainder of this paper we will first introduce DCR Graphs informally in Sec. 2, in Sec. 3 we will explain how they are used as the underlying formalism for workflows within the Exformatics ECM system and in Sec. 4 we will give an overview of the tools for managing DCR Graphs that have been developed by Exformatics. We evaluate and describe related work in Sec. 5 and conclusions and future work in Sec. 7.

2 DCR Graphs by Example

This section describes DCR Graphs informally by giving an overview of the declarative nature of the language and its graphical modeling notation. (All figures shown are produced in the developed graphical editor and simulation tool [18]). The formal semantics of DCR Graphs are given in [4, 6, 12].

A DCR Graph specifies a process as a set of *events*, typically representing the (possibly repeated) execution of activities in the workflow process, changes to a dataset or timer events. The events are represented graphically as rectangular boxes with zero or more *roles* in a small box on top of the event as depicted in Fig. 1, showing an excerpt of an invoice workflow with three events: Recieve Invoice, Enter Invoice Data and Responsible Approval and two roles: Administration (Adm), representing the administration office of a company and Responsible (Res), the person responsible for the invoice. The administation office has access to the tasks Recieve Invoice and Enter Invoice Data and the responsible has access to the task Responsible Approval.

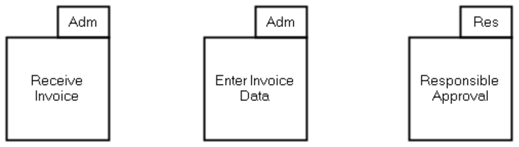
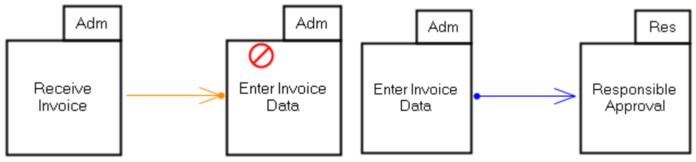


Fig. 1. DCR Graphs: Tasks and Roles

The concrete principals/actors (either human or automated) are typically not shown in the graphical notation, but will at runtime be assigned one or more of the roles and can then execute any of the events that are assigned to one of these roles.

The events in a DCR Graph can happen any number of times and in any order, unless prevented by a constraint relation. The graph in Fig. 1 has no constraints, so it would be valid to e.g. just receive an invoice and do nothing else, or to receive an invoice and then approve the invoice twice. Constraints are defined using five different kinds of relations between the events, named the *condition*, *response*, *milestone*, *inclusion* and *exclusion* relation respectively.

Fig. 2(a) gives an example of a condition relation (depicted graphically as →•) between Recieve Invoice and Enter Invoice Data, which states that before Enter Invoice Data can happen, the event Recieve Invoice must first have happened. In other words, we have to receive an invoice before we can enter the details of the invoice into the system. The DCR Graph shown in Fig. 2(a) allows possible runs such as Recieve Invoice.Enter Invoice Data or Recieve Invoice.Enter Invoice Data.Recieve Invoice or Recieve Invoice.Recieve Invoice.Enter Invoice Data, but it does not allow e.g. Enter Invoice Data. Recieve Invoice as it invalidates the condition constraint. As a help for the user, the graphical editor shows a ”no entry” sign at the event Enter Invoice Data to indicate that it is not enabled.



(a) The Condition Relation (b) The Response Relation

Fig. 2. The Condition and Response relations

In Fig. 2(b) is given an example of the response relation (depicted graphically as $\bullet\!\!\rightarrow$), which states that if Enter Invoice Data happens, Responsible Approval eventually has to happen in order for the workflow to be completed. Note that this relation is not counting, i.e., it is not required to execute Responsible Approval once for each execution of Enter Invoice Data. In other words, the response relation offers the flexibility of approving one to many invoices just by executing Responsible Approval once. Examples of completed runs in the process represented by the graph in Fig. 2(b) are: Enter Invoice Data.Responsible Approval, and Enter Invoice Data.Enter Invoice Data.Responsible Approval. An example of a run which is possible, but not completed is Enter Invoice Data.Responsible Approval.Enter Invoice Data as the last Enter Invoice Data is not (yet) followed by Responsible Approval.

In [6] we extended DCR Graphs to allow *nested* events as shown in Fig. 3. Nesting both acts as a logical grouping and as a shorthand notation for having the same relation between many events. For instance, the response relation from Enter Invoice Data in Fig. 3 represents a response relation from Enter Invoice Data to all three sub events of the super event Approval.

Adding nesting to the model, made it apparant, that it is useful to be able of express, that an event can not happen when a nested subgraph is not in an accepting state. We call this relation the milestone relation (depicted graphically as $\rightarrow\!\!\diamond$), and is exemplified shown in Fig. 3 from the Approval super event to Pay Invoice. The meaning is, that after doing Enter Invoice Data, we will have a pending response on each approval task and therefore we can't execute Pay Invoice until each of these tasks has been done. Note that in contrast to the condition relation, by using a combination of the response and milestone relations we can require approval again after it was already given.

Finally, the exclude relation (depicted graphically as $\rightarrow\!\!\%$) and its dual the include relation (depicted graphically as $\rightarrow\!\!+$) allows for dynamically respectively exclude and include events from the workflow. Fig. 4(a) shows a typical example of the use of the

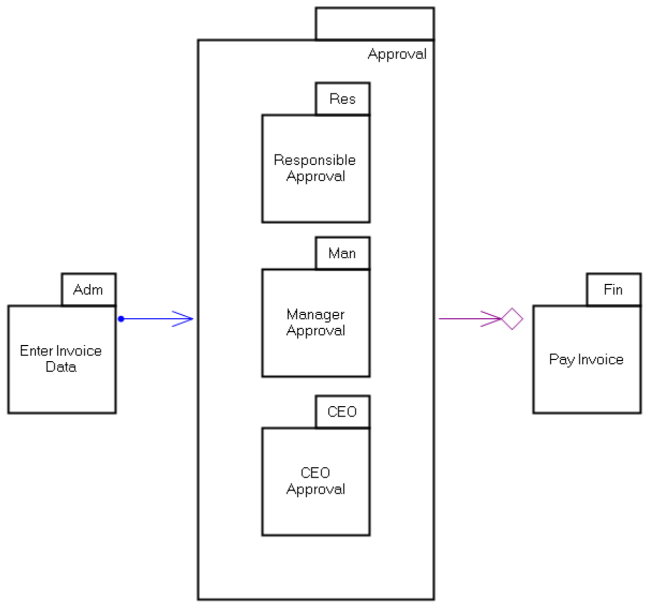
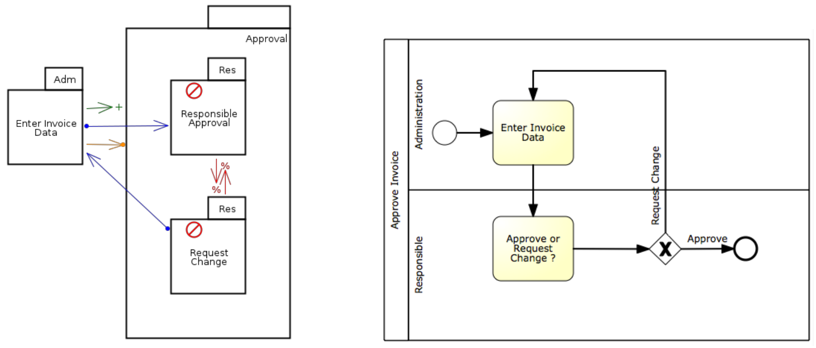


Fig. 3. Example of Nesting and the Milestone Relation

dynamic include and exclude relations to model *exclusive choice* between events: The responsible may choose between approving or request a change to the invoice. The choice is modelled by letting the two events mutually exclude each other. If a change is requested, the administration is required to enter data again (because of the response relation from Request Change to Enter Invoice Data), and when data is entered again, the two events nested under the *Approval* super event is included again because of the include relation from Enter Invoice Data to Approval. This example illustrates the flexible nature of DCR Graphs in process modeling, as compared to the typical BPMN procedural model in Fig. 4(b). In the DCR Graph, invoice data can be entered any number of times before approval, and changes can also be requested any number of times before data is entered again, while the BPMN process only allows every task to be executed once for each cycle in the loop. It is of course possible to model the more flexible execution in BPMN, but not in a natural way.

2.1 Execution Semantics

The runtime state of a DCR Graph is defined by a *marking* of the graph, formally given by 3 finite sets of events representing respectivly which events are *executed* (at least once), *pending responses* and *included*. By keeping track of which events have been executed at least once in the *executed* set, we can determine which conditions have been satisfied. The *pending responses* set keeps track of which events need to be executed before the workflow is in a completed state. Finally, the *included* set keeps track of the



(a) Modeling choice with include and exclude (b) Imperative BPMN model

Fig. 4. Declarative DCR Graph and imperative BPMN model of invoice approval

currently included events. An event is enabled for execution if it is currently included (i.e. part of the included set in the current marking) and all of its conditions are either executed or excluded (i.e all condition events that are currently included should be part of the executed events set) and no event related to it by the milestone relation is included and a pending response. A (finite or infinite) execution is defined to be accepting, when no event from some point stays included and as a pending response forever without eventually being executed.

The excluded events are graphically depicted by a dashed border, the executed events by a green checkmark at the event, and pending response events by a red exclamation mark. This is shown in Fig. 5, where Enter Invoice Data and Request Change are executed, and thereby Responsible Approval is a pending response, but it is also excluded and Enter Invoice Data is a pending response too.

A DCR Graph contains an initial marking defined as part of the graph. For example, a graph may have a number of initial pending responses (representing tasks that are required to be executed mandatorily for the workflow to be considered finished), or initially excluded events.

2.2 DCR Graphs with Global Data

In one of the more recent extensions to DCR Graphs [12], we have introduced the concept of global data. In DCR Graphs, data is modelled as a global store that contains a number of named variables. The variables are mapped to events so that we can specify which events can read/write to specific variables. Furthermore, *guards* are defined as boolean expressions over the values of variables. Guards can be placed on both events and relations. If a guard is assigned to an event, then as long as the guard does not evaluate to true, the event is blocked from execution. On the other hand, having a guard on a relation means that the relation is only evaluated when the guard evaluates to true, in other words the condition constraint only needs to hold and an event is only recorded as a response while the guard holds.

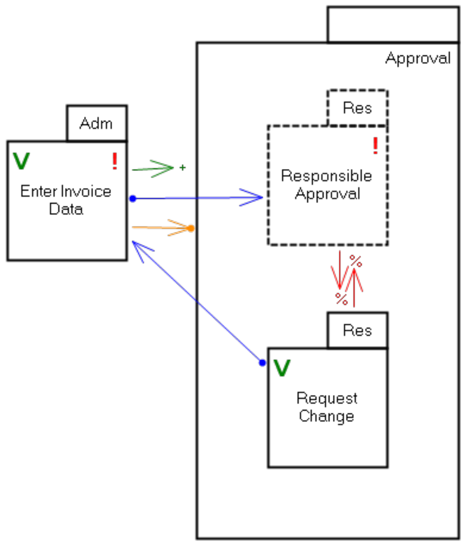


Fig. 5. Example marking after executing Enter Invoice Data followed by Request Change

For example, the response between Enter Invoice Data and Manager Approval in Fig 6 is only recorded when the amount of the invoice is equal or larger than 1000 euro, if the amount is lower than 1000 euros, executing Enter Invoice Data will not make Manager Approval a pending response.

3 Exformatics Workflows as DCR Graphs

Before the introduction of (Nested) DCR Graphs, the Exformatics workflow model consisted of tasks grouped under phases. There was always one active phase, which could be changed manually by the user, tasks belonging to that phase were then enabled. When introducing DCR Graphs we chose to map tasks to events and to maintain the phase model, mapping it to a single-level nesting structure. We removed the practice that tasks were enabled when their phase was active and allowed the active phase to be changed automatically through the execution of certain tasks. In the new model, the active phase no longer controls the workflow but instead just gives a general indication of the state that the case is in. We introduced all five relations of DCR Graphs as ways of constraining the flow of tasks. One distinction from the traditional DCR Graph approach is that tasks in the Exformatics system are normally only done once. As a result, when a task is executed, it is not shown in the list of tasks that need to be done anymore. However, unless it is exlicitely excluded through the exclude relation it remains possible to open the task again manually and do it again, so the execution semantics remains faithful to the DCR Graphs semantics.

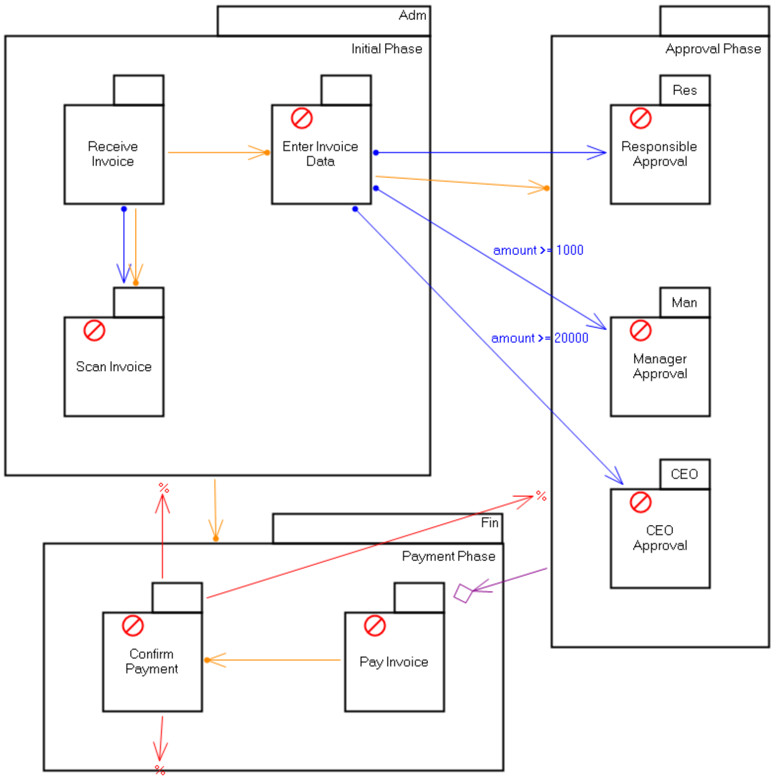


Fig. 6. Exformatics Invoice Workflow as a DCR Graph

Fig. 6 shows a workflow that is being used internally by Exformatics and has been modelled using DCR Graphs. It describes how to handle the process of receiving invoices.

The workflow contains five roles: 1) the administration department (Adm), which is responsible for receiving the invoice, scanning it and creating an invoice case . 2) The invoice responsible (Res), which is responsible for the invoice, usually because they are the person that bought the items that the invoice concerns, they are expected to check and approve the invoice. 3) The manager of the responsible (Man), whose approval may be needed in certain circumstances. 4) The CEO (CEO) who may also need to give approval in certain exceptional cases. And finally 5) the finance department (Fin), which takes care of paying the invoice and confirming that payment has succeeded. The tasks are divided into three phases, the Initial Phase which contains the tasks of the administration department, the Approval Phase which consists of the approval tasks and the Payment Phase which contains the tasks that handle the payment of the invoice.

The process starts when an invoice is received by the administration department, because Exformatics wants to keep all their documents in an electronic format it is required (through the response relation from Receive Invoice to Scan Invoice) that the invoice is scanned. The administration department is also required to decide if the invoice should be entered into the system (sometimes fake or wrong invoices are received which can be easily filtered out at first sight, for example because they are addressed to a non-existent employee). If they decide that the invoice appears legit then they enter all relevant data into the system, in particular the amount the invoice is for, which is used by the workflow system to determine whose approval is needed for the invoice. The responsible for the invoice should always approve the invoice (modelled by an unguarded response relation), if the amount of the invoice is higher then 1000 euros, approval from the responsible's manager is required as well (modelled by a response relation with the guard $\mathsf{amount} \geq 1000$). In special cases where the amount is higher then 20000 euros, approval from the CEO of the company is required as well.

It is possible that data is entered again, for example because a mistake was made by the administration department, or because a correction on the invoice was received, in this case new approvals will be required. When all necessary approvals have been received the invoice can be paid, this is modelled through the milestone relation from the Approval Phase to the task Pay Invoice, which means that Pay Invoice can not be done while there are pending responses in the Approval Phase. Once payment is confirmed, the invoice case should be closed, modelled through an exclusion relation from Confirm Payment to all three phases. There are five conditions in the workflow: first of all, Receive Invoice is required before the administration department can execute Enter Invoice Data or Scan Invoice. Enter Invoice Data is required before any approval can be given and all of the tasks in the Initial Phase should be done before any of the tasks in the Payment Phase can be done. Finaly, we have to pay the invoice before we can confirm payment.

4 Tool support

Several tools have been developed at Exformatics to design and execute DCR Graphs internally or externally when presenting DCR Graphs at seminars or when interacting with customers. First of all, to facilitate the exchange of process descriptions between the tools developed by Exformatics and the tools being developed at IT University of Copenhagen, we defined a common XML format, which we will show in the first subsection. Secondly we developed a set of webservices that provide functionality for the execution, verification, storage and visualization of DCR Graphs, we named this set of services the *Process Engine*. Finally, as already mentioned above, we developed a standalone graphical editor to support the visual modelling and simulation of DCR Graphs, called the *DCR Graphs Editor*, which has also been used for teaching at a bachelor level course on Business Processes and IT at the IT University of Copenhagen.

Fig. 7 gives an overview of these tools and how they interact with eachother and the Exformatics ECM. The Process Engine is central to our tools and is used by the ECM to execute, verify and visualize workflows. The DCR Graphs Editor allows for execution of single steps by itself, but also uses the Process Engine for verification

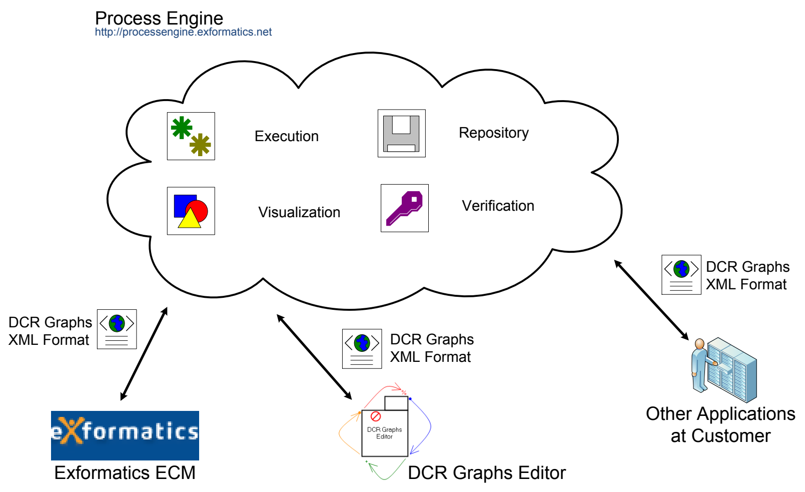


Fig. 7. Overview of the Exformatics DCR Graphs Tools

of DCR Graphs. Finally the purpose of the Process Engine is to be easily plugged in to other case management solutions as well, so that we may provide only workflow functionalities such as execution, verification, visualization and storage to customers without them being required to adopt the full Exformatics ECM package.

4.1 DCR Graphs XML Format

In listing 1 we give an example of the XML format for describing DCR Graphs.

The xml file consists of two main parts: the specification of the DCR Graph and the runtime state of the DCR Graph. The specification is split up into a section decribing resources and section describing constraints. The resource section contains subsections for events (possibly nested), labels, a mapping from labels to events, variables, expressions and variable acccess rights. The constraint section contains five subsections for the DCR Graph relations. The runtime section contains a subsection for the marking, containing the set of executed events, pending responses and included events, and a subsection for the state of the globalstore, which contains the values assigned to the variables in the current state.

Listing 1. Overview of DCR Graph XML Format

```
<?xml version = "1.0" encoding = "utf 8"?>
<dcrgraph>
  <specification>
    <resources>
      <events>
        <event id="Initial Phase">
          <event id="Enter Invoice Data"/>
          ...
        </event>
        ...
      </events>
      <labels>
        <label id="CEO Approval"/>
        ...
      </labels>
      <labelMappings>
        <labelMapping eventId="CEO Approval" labelId="CEO Approval"/>
        ...
      </labelMappings>
      <variables>
        <variable id="amount" value="0"/>
      </variables>
      <expressions>
        <expression id="gte1000" value="amount >= 1000"/>
        ...
      </expressions>
      <variableAccesses>
        <readAccesses>
          <readAccess eventId="Enter Invoice Data" variableId="amount"/>
          ...
        </readAccesses>
        <writeAccesses>
          <writeAccess eventId="Enter Invoice Data" variableId="amount"/>
        </writeAccesses>
      </variableAccesses>
    </resources>
    <constraints>
      <conditions>
        <condition sourceId="Receive Invoice" targetId="Scan Invoice"/>
        ...
      </conditions>
      <responses>
        <response sourceId="Enter Invoice Data" targetId="Manager Approval"
             expressionId="gte1000"/>
        ...
      </responses>
      <excludes>
        <exclude sourceId="Confirm Payment" targetId="Approval Phase"/>
        ...
      </excludes>
      <includes/>
      <milestones>
        <milestone sourceId="Approval Phase" targetId="Pay Invoice"/>
      </milestones>
    </constraints>
  </specification>
  <runtime>
    <marking>
      <executed/>
      <included>
        <event id="Approval Phase"/>
        ...
      </included>
      <pendingResponses/>
    </marking>
    <globalStore>
      <variable id="amount" value="0"/>
    </globalStore>
  </runtime>
</dcrgraph>
```

Next to the standard elements described above, it is possible to insert *custom* elements at all nodes of the XML tree. This allows one to add additional data for specific tools that is not required for the formal definition of a DCR Graph. Examples of these are the roles (they are not a part of the formal model as they are not necesairily interesting for applications in other domains than BPM) and the location of events when drawn in the visual editor as shown in listing 2.

Listing 2. Example of how custom data can be insterted into the XML format.

```
<?xml version="1.0" encoding="utf 8"?>
<event id="CEO Approval">
  <custom>
    <visualization>
      <location xLoc="449" yLoc="123"/>
    </visualization>
      <roles>
        <role>CEO</role>
    </roles>
  </custom>
</event>
```

4.2 Process Engine

Currently the Process Engine consists of three main webservices: the first for execution, the second for storage of DCR Graphs and the third for visualization of DCR Graphs. The execution service contains methods for executing and verifying DCR Graphs. The execution methods support the global data model, verification consists of checking for deadlock and livelock, but only for standard DCR Graphs without data. In the future we plan to extend the verification aspect and move it to its own service. The repository service for storage of DCR Graphs is currently very limited and mainly a proof of concept, it is planned to extend this in the future so it can be used to support sharing of workflows between cooperating organizations. The visualization service can be used to automatically layout and draw DCR Graphs, currently limited to the basic model without guards on data. All of these services are used by the Exformatics ECM for modelling and executing workflows.

4.3 DCR Graphs Editor

The DCR Graphs Editor is a graphical editor for modelling and simulating DCR Graphs. There are two main screens in the tool: in the Process Model screen one can design DCR Graphs by drawing events, changing the name, label and initial marking, adding roles and adding relations between events. In he Process Simulation Screen one can simulate DCR Graphs by clicking on the events that one wants to execute, the tool will give feedback on the current trace of executed events, which events can be executed and if the DCR Graph is in an accepting state. The tool can also interact with the verification methods of the Process Engine to check DCR Graphs for deadlock and livelock. It currently supports nested DCR Graphs including the milestone relation and work is underway to also add support for the global data model. All the images of DCR Graphs in this paper come directly from the editor.

5 Related Work

As mentioned in the introduction Declare [15, 19] was the first serious attempt at creating a declarative notation for describing business processes. Tool support for Declare consists of a design tool, a server and corresponding user client for executing Declare processes. The designer is similar to the DCR Graphs Editor, allowing modellers to draw and verify Declare models (including a notion of data) by using a graphical user interface. The server is similar to the execution webservices contained in the Process Engine, allowing execution of Declare models by client programs. Finally the user client is somewhat comparable to the simulation part of the DCR Graphs Editor, although it offers more features to support the user in the execution of the process. These tools have been in development since the inception of the Declare language and therefor have seen a fair amount of iterations and reached a high level of maturity. The DCR Graphs tools on the other hand can be seen as being an advanced prototype version (with the most mature parts, such as the execution engine, currently being brought into production), where new features are still frequently being added. Both Declare and DCR Graphs are being included as extensions to the newest version of CPN Tools [20], for Declare it is the intention that this will become the main vehicle for further developments on the language and that no further features will be added to the previously mentioned tools. Declare also offers extensive support for analysis of Declare logs through ProM and support for process mining through the Declare Miner [11]. At the moment nothing comparable exists for DCR Graphs, however there is an interest in investigating process mining on running instances of DCR Graphs, particularly in the context of adaptive processes, with the goal of identifying common adaptation patterns. DCR Graphs also offer extended tool support for verification, allowing users to specify properties to be verified as a DCR Graph and then verifying processes modelled as DCR Graphs against these properties [13]. These tools are being developed at the IT University of Copenhagen and are therefor not described in detail this paper, however since these tools use the common XML format described in sec. 4.1, the Exformatics tools can easily interact with them.

The business artifacts [14] model developed by IBM Research combines both data and process aspects in a holistic manner. An artifact type contains both an information model (data for business objects) and a lifecycle model, which describes the possible ways a business entity might progress through and responds to events and external activities. A declarative approach using *Guard-Stage-Milestone* (GSM model) [9] based on ECA(Event Condition Action)-like rules for specification of life cycles on business artifacts has been developed in the recent years. Compared to DCR Graphs, the GSM-model has a richer support for data, but also a more complex semantics that does not capture acceptance criteria for infinite executions.

6 Evaluation

This work provides an initial report on tools being developed at Exformatics A/S examplified by a use-case being used internally within the company itself. As such no concrete quantitative evaluation of the usefulness and commercial viabilty of the tools exists yet. However, DCR Graphs as a modelling paradigm and the Exformatics tools themselves have already seen both commercial and academic use. As a modelling paradigm, DCR Graphs were applied in a commercial project involving Exformatics and *Landsorganisationen i Danmark (LO)*, the umbrella organisation for Danish unions. During this project DCR Graphs were used to model the IT system that Exformatics developed for LO [1], but the lack of tool support for design and simulation limited its use. In [5] we showed how DCR Graphs can be used to model a distributed healthcare process encountered in a Danish hospital. DCR Graphs and the tools are currently employed in a project jointly with a danish research foundation for modelling the case management process for handling funding applications from submission to decision. All of these cases have been demonstrated for industry at seminars with positive feedback resulting in several requests for follow up meetings. Finally, Exformatics has recently started a commercial project for the Danish Cancer Society, including the development of an invoice approval solution based on the example used in this paper and using the Process Engine for execution of the workflows in the solution.

In the recent paper [17] we give the first empirical evaluation on what practitioners think of declarative modelling based on a study performed at a Dutch provider of ECM software. During the study some of those participating were presented Declare, while others were presented DCR Graphs. While the overall results of the study point in the direction of a hybrid model combining the imperative and declarative paradigms, it was also clear that the declarative paradigm by itself was percieved as useful for the right application domains.

In the Spring 2012 and 2013, the DCR Graphs model has been introduced in a bachelor course in IT and Business Process Modelling at the IT University of Copenhagen [2]. Each year, the course was followed by about 40 students, and the DCR Graph model was introduced for capturing process requirements, along with BPMN 2.0 for modelling processes imperatively. The students worked in groups, modelling their own processes identified in a field study performed in a previous course. They first modelled the process in BPMN and subsequently were asked to model the requirements in DCR Graphs and compare the models. They all experienced that the initial BPMN was good at describing a procedure of *how* to carry out the process. However, when turning to the DCR Graph model, they also realized that in most cases their BPMN model only described a fairly rigid, happy path through the process. In most cases it took the group two iterations to change their mindset to model requirements instead of the procedure. This may however be influenced by the fact, that they did no longer have access to the company in which they had performed the field study. Only in 2013, the DCR Graphs editor was available, and we experienced that it made it much easier for the students to learn the notation and semantics, and to appreciate its use for modelling process requirements. However, it was also clear that it still could be difficult for some of the students to visualize the possible paths of the process specified as DCR Graphs.

7 Conclusion

In this paper, we have given an informal introduction to DCR Graphs and briefly described current tool support, and how DCR Graphs and the tools are being used by Exformatics and in teaching at ITU university to model workflows.

Even though the uses in practice and teaching so far is limited, it has been very encouraging. At presentations for industry the models have generally been appreciated and easily understood. At the course the students were able to apply DCR Graphs to model processes obtained from their own field studies in a previous course. They reported back that using the simulation facility in the tool was a great help to understand both the constraints of their own process and DCR Graphs as a model language.

As part of the future work, we plan to further develop the tools, making them more easily accessible and user-friendly to process modelers, based on the usability studies and feedback from students and clients of Exformatics. Furthermore, we also intend to upgrade the tools to support some of the latest extensions on DCR Graphs such as time [7], a distributed data model and more advanced verification techniques. Similarly, we are also working on extending the theory of DCR Graphs to provide a behavioral type system for cross-organizational workflows as initiated in [3]. In the future we also want to research the challenge of developing business processes for knowledge-intensive and adaptive case management processes as initiated in [13], which require more focus on evolutionary process data and adaptability of the process during execution.

References

1. T. Hildebrandt, R.R. Mukkamala, and T. Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 161 –170, 29 2011-sept. 2 2011. 2, 14
2. Thomas Hildebrandt. It and business process modelling course. *IT University of Copenhagen*, 2013. `https://blog.itu.dk/BIMF-F2013/`. 14
3. Thomas Hildebrandt, Marco Carbone, and Tijs Slaats. Rsvp: Live sessions with responses. In *Proceedings of BEAT'13, 1st International Workshop on Behavioural Types*, 2013. 15
4. Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Post-proceedings of PLACES 2010*, 2010. 2, 3
5. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Declarative modelling and safe distribution of healthcare workflows. In *International Symposium on Foundations of Health Information Engineering and Systems*, Johannesburg, South Africa, August 2011. 2, 14
6. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. 2, 3, 4
7. Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Contracts for cross-organizational workflows as timed dynamic condition response graphs. *Journal of Logic and Algebraic Programming (JLAP)*, may 2013. `http://dx.doi.org/10.1016/j.jlap.2013.05.005`. 15
8. Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In Kohei Honda and Alan Mycroft, editors, *PLACES*, volume 69 of *EPTCS*, pages 59–73, 2010. 1
9. Richard Hull. Formal study of business entities with lifecycles: Use cases, abstract models, and results. In Tevfik Bravetti, Mario; Bultan, editor, *7th International Workshop on Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, 2010. 1, 13
10. Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and

Roman Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of WS-FM'10*, pages 1–24, Berlin, Heidelberg, 2011. Springer-Verlag. 2

11. F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2011. 13
12. Raghava Rao Mukkamala. *A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, March 2012. Forthcomming. 3, 6
13. Raghava Rao Mukkamala, Thomas Hildebrandt, and Tijs Slaats. Towards trustworthy adaptive case management with dynamic condition response graphs. In *Proceedings of the 17th IEEE International EDOC Conference, EDOC 2013*, 2013. 13, 15
14. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42:428–445, July 2003. 13
15. M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. Van Der Aalst. Constraint-based workflow models: change made easy. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07, pages 77–94, Berlin, Heidelberg, 2007. Springer-Verlag. 1, 2, 12
16. M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proc. of the 2006 international conference on Business Process Management Workshops*, BPM'06, pages 169–180. Springer-Verlag, 2006. 1, 2
17. Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative Modeling—An Academic Dream or the Future for BPM? . Accepted for BPM 2013. 14
18. Tijs Slaats. Dcr graphs wiki. *IT University of Copenhagen*, 2013. `http://www.itu.dk/research/models/wiki/index.php/DCR_Graphs_Editor`. 2, 3
19. W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. 1, 2, 12
20. M. Westergaard and T. Slaats. Mixing Paradigms for More Comprehensible Models. Accepted for BPM 2013. 13
21. Glynn Winskel. *Events in Computation*. PhD thesis, Edinburgh University, 1980. 2

6 Safe Cross-Organizational Flexible Processes

Safe Distribution of Declarative Processes

Thomas Hildebrandt[1], Raghava Rao Mukkamala[1], and Tijs Slaats[1,2] ⋆

[1] IT University of Copenhagen
Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
{hilde, rao, tslaats}@itu.dk, `http://www.itu.dk`
[2] Exformatics A/S, 2100 Copenhagen, Denmark

Abstract. We give a general technique for safe distribution of a declarative (global) process as a network of (local) synchronously communicating declarative processes. Both the global and local processes are given as Dynamic Condition Response (DCR) Graphs. DCR Graphs is a recently introduced declarative process model generalizing labelled prime event structures to a systems model able to finitely represent ω-regular languages. An operational semantics given as a transition semantics between markings of the graph allows DCR Graphs to be conveniently used as both specification and execution model. The technique for distribution is based on a new general notion of projection of DCR Graphs relative to a subset of labels and events identifying the set of external events that must be communicated from the other processes in the network in order for the distribution to be safe. We prove that for any vector of projections that covers a DCR Graph that the network of synchronously communicating DCR Graphs given by the projections is bisimilar to the original global process graph. We exemplify the distribution technique on a process identified in a case study of an cross-organizational case management system carried out jointly with Exformatics A/S.

1 Introduction

A model-driven software engineering approach to distributed information systems typically include both *global* models describing the collective behavior of the system being developed and *local* models describing the behavior of the individual peers or components.

The global and local descriptions should be consistent. If the modeling languages have formal semantics and the local model language support composition of individual processes, the consistency can be formally established, which we will refer to as the *consistency problem*: Given a global model and a set of local models, is the behavior of the composition of the local models consistent with the global model? In order to support ”top-down” model-driven engineering starting from the global model, one should address the more challenging *distributed synthesis problem*: Given a global model and

⋆ Authors listed alphabetically. This research is supported by the Danish Research Agency through a Knowledge Voucher granted to Exformatics (grant #10-087067, www.exformatics.com), the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu) and the Computer Supported Mobile Adaptive Business Processes project (grant #274-06-0415, www.cosmobiz.dk).

some formal description of how the model should be distributed, can we synthesize a set of local processes with respect to this distribution which are consistent to the the global model?

In past work, briefly surveyed below, the result of the distributed synthesis have been a network of local processes described in an imperative process model, e.g. as a network of typed pi-calculus processes or a product automaton. The global process description has either been given declaratively, e.g. in some temporal logic, or imperatively, e.g. as a choreography or more generally a transition system.

In the present paper we address the distributed synthesis problem in a setting where both the global and the local processes are described *declaratively* as Dynamic Condition Response Graphs (DCR Graphs). DCR Graphs is a declarative workflow model introduced previously in [14, 15] as a generalization of the classical event structure model [47] allowing finite specification of infinite behavior (by allowing events to be executed more than once and replacing the symmetric conflict relation by asymmetric exclusion and (re-)inclusion relations) and specification of progress conditions (by replacing the causal order relation of event structures with two relations, respectively defining the conditions for and required responses to the execution of an event).

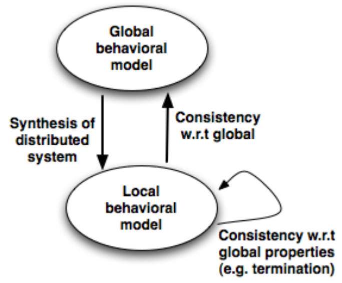
The motivation for introducing the DCR Graph model is to give, as part of the Trustworthy Pervasive Healthcare Services [13] project, a declarative model that can be used both as specification language and execution language for flexible workflow and business process. Indeed, the DCR Graphs model is inspired by and formalizes the core primitives of the process model employed by the industrial partner (Resultmaker) in the TrustCare project and is now being implemented in the workflow engine developed at Exformatics. As identified in e.g. [6,43] declarative process languages make it easier to specify loosely constrained systems. Also, we believe the declarative approach is more promising when it comes to composition, and (dynamic) changes of processes which is one of the main objectives of the TrustCare project.

To safely distribute a DCR Graph we first define (Def. 3, Sec. 3.1) a new general notion of *projection* of DCR Graphs relative to a subset of labels and events. The key point is to identify the set of events that must be communicated from other processes in the network in order for the state of the local process to stay consistent with the global specification (Prop. 1, Sec. 3). To also enable the reverse operation, building global graphs from local graphs, we then define the composition of two DCR Graphs, essentially by gluing joint events. As a sanity check we prove (Prop. 2, Sec. 3.2) that if we have a collection of projections of a DCR Graph that cover the original graph (Def. 7, Sec. 3.2) then the composition yields back the same graph. We then finally proceed to the main technical result, defining networks of synchronously communicating DCR Graphs and stating (in Thm. 1, Sec. 3.3) the correspondence between a global process and a network of communicating DCR Graphs obtained from a covering projection (relying on Prop. 1). Throughout the paper we exemplify the distribution technique on a simple cross-organizational process identified within a case study carried out jointly with Exformatics A/S using DCR Graphs for model-driven design and engineering of an inter-organizational case management system. We conclude in Sec. 4 and provide pointers to future work.

1.1 Related Work

There are many researchers [1, 20, 21, 40–42, 46] who have explicitly focussed on the problem of verifying the correctness of inter-organizational workflows in the domain of petri nets. In [41], message sequence charts are used to model the interaction between the participant workflows that are modeled using petri nets and the overall workflow is checked for consistency against an interaction structure specified in message sequence charts. In [20] Kindler et. al. followed a similar but more formal and concrete approach, where the interaction of different workflows is specified using a set of scenarios given as sequence diagrams and using criteria of local soundness and composition theorem, guaranteed the global soundness of an inter-organizational workflow. The authors in [40] proposed *Query Nets* based on predicate/transition petri nets to guarantee global termination, without the need for having the global specification. The work on workflow nets [1, 46] use a P2P (Public-To-Private) approach to partition a shared public view of an inter-organizational workflow over its participating entities and projection inheritance is used to generate a private view that is a subclass to the relevant public view, to guarantee the deadlock and livelock freedom. Further a more liberal and a weaker notion than projection inheritance, *accordance* has been used in [42] to guarantee the weak termination in the multiparty contracts based on open nets.

Modeling global behavior as a set of conversations among participating services has been studied by many researchers [2, 3, 11, 35, 48, 49] in the area business processes. An approach based on guarded automata studied in [11], for the realizability analysis of conversation protocols, whereas the authors in [49] used colored petri nets to capture the complex conversations. A framework for calculating and controlled propagation of changes to the process choreographies based on the modifications to partner's private processes has been studied in [35]. Similarly, but using process calculus to model service contracts, Bravetti-Zavattaro proposed conformance notion for service composition in [2] and further enhanced their correctness criteria in [3] by the notion of strong service compliance.


Global behavioral model
Synthesis of distributed system
Consistency w.r.t global
Local behavioral model
Consistency w.r.t global properties (e.g. termination)
Fig. 1. Key problems studied in related work

Researchers [9,19,23,29] in the web services community have been working on web service composition and decentralized process execution using BPEL [30] and other related technologies to model the web services. A technique to partition a composite web service using program analysis was studied in [29] and on the similar approach, [19] explored decomposition of a business process modeled in BPEL, primarily focussing on P2P interactions . Using a formal approach based on I/O automata representing the services, the authors in [23] have studied the problem of synthesizing a decentralized choreography strategy, that will have optimal overhead of service composition in terms of costs associated with each interaction.

The derivation of descriptions of local components from a global model has been researched for the imperative choreography language WS-CDL in the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [4]. To put it briefly, the work formalizes the core of WS-CDL as the global process calculus and defines a formal theory of end-point projections projecting the global process calculus to abstract descriptions of the behavior of each of the local ”end-points” given as pi-calculus processes typed with session types.

A methodology for deriving process descriptions from a business contract formalized in a formal contract language was studied in [22], while [36] proposes an approach to extract a distributed process model from collaborative business process. In [9, 10], the authors have proposed a technique for the flexible decentralization of a process specification with necessary synchronization between the processing entities using dependency tables.

In [5, 12, 27] foundational work has been made on synthesizing distributed transition systems from global specification for the models of synchronous product and asynchronous automata [50]. In [27] Mukund categorized structural and behavioral characterizations of the synthesis problem for synchronous and loosely cooperating communication systems based on three different notions of equivalence: state space, language and bisimulation equivalence. Further Castellani et. al. [5] characterized when an an arbitrary transition system is isomorphic to its product transition systems with a specified distribution of actions and they have shown that for finite state specifications, a finite state distributed implementation can be synthesized. Complexity results for distributed synthesis problems for the three notions of equivalences were studied in [12].

Many commercial and research workflow management systems also support distributed workflow execution and some of them even support ad-hoc changes as well. ADEPT [34], Exotica [24], ORBWork [7], Rainman [31] and Newcastle-Nortel [39] are some of the distributed workflow management systems. A good overview and discussion about distributed workflow management systems can be found in [32, 33].

So far the formalisms discussed above are more or less confined to imperative modeling languages such as Petri nets, workflow/open nets and automata based languages. To the best of our knowledge, there exists very few works [8, 25] that have studied the synthesis problem in declarative modeling languages and none where both the global and local processes are given declaratively. In [8], Fahland has studied synthesizing declarative workflows expressed in DecSerFlow [45] by translating to Petri nets. Only a predefined set of DecSerFlow constraints are used in the mapping to the Petri nets patterns, so this approach has a limitation with regards to the extensibility of the DecSerFlow language. On the other hand, in [25] Montali has studied the composition of ConDec [44] models with respect to conformance with a given choreography, based on the compatibility of the local ConDec models. But his study was limited to only composition, whereas the problem of synthesizing local models from a global model has not been studied.

2 Dynamic Condition Response Graphs

Dynamic Condition Response (DCR) Graphs has recently been introduced [15] as a declarative process model generalizing labelled event structures [47] to allow finite representations of infinite behavior (i.e. a systems model [37, 38]) and representation of progress properties.

A DCR Graph consists of a set of *labelled events*, a *marking* defining the *executed* events, *pending response* events and *included* events, and four binary relations between the events, defining the temporal constraints between events and dynamic inclusion and exclusion of events.

We employ the following notations in the rest of the paper.

Notation: For a set A we write $\mathcal{P}(A)$ for the power set of A. For a binary relation $\rightarrow \subseteq A \times A$ and a subset $\xi \subseteq A$ of A we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{a \in A \mid (\exists a' \in \xi \mid a \rightarrow a')\}$ and the set $\{a \in A \mid (\exists a' \in \xi \mid a' \rightarrow a)\}$ respectively. Also, we write $\rightarrow^{-1}$ for the inverse relation. Finally, for a natural number k we write $[k]$ for the set $\{1, 2, \ldots, k\}$.

We then formally define a DCR Graph as follows.

Definition 1. *A Dynamic Condition Response Graph G is a tuple* $(\mathsf{E_G}, \mathsf{M_G}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L_G}, l_\mathsf{G})$*, where*

(i) $\mathsf{E_G}$ *is the set of* events

(ii) $\mathsf{M_G} = (\mathsf{Ex_G}, \mathsf{Re_G}, \mathsf{In_G}) \in \mathcal{P}(\mathsf{E_G}) \times \mathcal{P}(\mathsf{E_G}) \times \mathcal{P}(\mathsf{E_G})$ *is the* marking*,*

(iii) $\rightarrow\!\bullet \subseteq \mathsf{E_G} \times \mathsf{E_G}$ *is the* condition *relation*

(iv) $\bullet\!\rightarrow \subseteq \mathsf{E_G} \times \mathsf{E_G}$ *is the* response *relation*

(v) $\pm : \mathsf{E_G} \times \mathsf{E_G} \rightharpoonup \{+, \%\}$ *is a partial function defining the* dynamic inclusion and exclusion *relations by* $e \rightarrow\!+\, e'$ *if* $\pm(e, e') = +$ *and* $e \rightarrow\!\%\, e'$ *if* $\pm(e, e') = \%$

(vi) $\mathsf{L_G}$ *is the set of* labels

(vii) $l_\mathsf{G} : \mathsf{E_G} \rightarrow \mathcal{P}(\mathsf{L_G})$ *is a labeling function mapping events to sets of labels.*

We write $\mathcal{M}(G)$ *for the set* $\mathcal{P}(\mathsf{E_G}) \times \mathcal{P}(\mathsf{E_G}) \times \mathcal{P}(\mathsf{E_G})$ *of markings of G.*

The marking $\mathsf{M_G} = (\mathsf{Ex_G}, \mathsf{Re_G}, \mathsf{In_G})$ is a tuple of three sets defining respectively the previously executed events ($\mathsf{Ex_G}$), the set of required responses ($\mathsf{Re_G}$), and the currently included events ($\mathsf{In_G}$). The set of required responses are the events that must eventually be executed (or excluded) in order to accept the execution, also referred to as the pending responses. The set of included events are the events that currently are relevant for conditions and may be executed (if their conditions are met). The condition relation $\rightarrow\!\bullet$ defines which (of the currently included) events must have been executed before an event can be executed. That is, for an event e to be executed, it must be included, i.e. $e \in \mathsf{In_G}$ and the included conditions must be executed: $(\rightarrow\!\bullet\ e) \cap \mathsf{In_G} \subseteq \mathsf{Ex_G}$. The response relation $\bullet\!\rightarrow$ defines which responses are required after executing an event. That is, if the event e is executed, the events $e\ \bullet\!\rightarrow$ are added to the set of required responses in the marking. The dynamic inclusion and exclusion relations define how the set of included events changes by executing an event: If the event e is executed, the events $e\ \rightarrow\!+$ are added to the set of included events in the marking and the events $e\ \rightarrow\!\%$ are removed. Finally, an event is labelled by zero or more labels. (This is slightly more general than previous work, where labels of events were sets of triples consisting of an action, a role and a principal.)

Fig. 2 below shows an example DCR Graph identified during the development by Exformatics of a cross-organizational case management system for the umbrella organization of unions in Denmark, named LO.

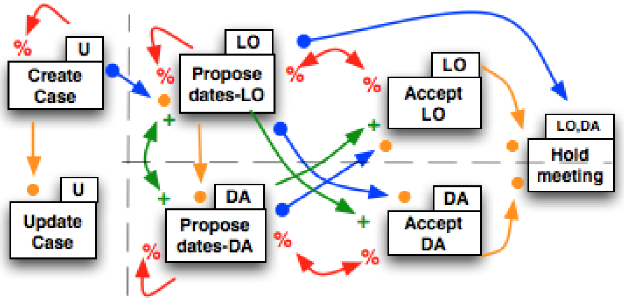


Fig. 2. Cross-organizational case management example

The graph has 7 events, drawn as boxes with ”ears”, and captures a process of creating a case, agreeing on meeting dates and holding meetings. The names of the events are written inside the box and the set of actions for each event, representing the roles that can execute the event, is written inside the ”ear”. That is, the event Create Case in the upper left has label U and represents the creation of a case by a case manager at a union (role U). The rightmost event, Hold meeting has two different labels, LO and DA, representing a meeting held by LO and DA (the umbrella organization of employers) respectively.

The semantics for DCR Graphs has been given in [14, 15] as a labelled transition system with acceptance condition for infinite computations. The set of accepted runs of DCR Graphs was characterized by a mapping to Büchi-automata in [26].

Definition 2. *For a DCR Graph* $G = (\mathsf{E_G}, \mathsf{M_G}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L_G}, l_\mathsf{G})$*, we define the corresponding labelled transition system* $TS(G)$ *to be the tuple* $(\mathcal{M}(G), \mathsf{M_G}, \mathcal{EL}(G), \rightarrow)$ *where* $\mathcal{EL}(G) = \mathsf{E_G} \times \mathsf{L_G}$ *is the set of labels of the transition system,* $\mathsf{M_G} = (\mathsf{Ex_G}, \mathsf{In_G}, \mathsf{Re_G}) \in \mathcal{M}(G)$ *is the initial marking, and* $\rightarrow \subseteq \mathcal{M}(G) \times \mathcal{EL}(G) \times \mathcal{M}(G)$ *is the transition relation defined by* $\mathsf{M_G}' \xrightarrow{(e,a)} \mathsf{M_G}''$ *if*

(i) $\mathsf{M_G}' = (\mathsf{Ex_G}', \mathsf{In_G}', \mathsf{Re_G}')$ *is the marking before transition*
(ii) $\mathsf{M_G}'' = (\mathsf{Ex_G}'', \mathsf{In_G}'', \mathsf{Re_G}'')$ *is the marking after transition*
(iii) $e \in \mathsf{In_G}'$, $a \in l_\mathsf{G}(e)$
(iv) $\rightarrow\!\bullet e \cap \mathsf{In_G}' \subseteq \mathsf{Ex_G}'$,
(v) $\mathsf{Ex_G}'' = \mathsf{Ex_G}' \cup \{e\}$
(vi) $\mathsf{In_G}'' = (\mathsf{In_G}' \cup e\!\rightarrow\!+) \setminus e\!\rightarrow\!\%$,

(vii) $\mathsf{Re}_{\mathsf{G}}'' = (\mathsf{Re}_{\mathsf{G}}' \setminus \{e\}) \cup e{\bullet}\!\!\rightarrow$,

We define a run $a_0, a_1, \ldots$ *of the transition system to be a sequence of labels of a sequence of transitions* $\mathsf{M}_{\mathsf{G}i} \xrightarrow{(e_i,a_i)} \mathsf{M}_{\mathsf{G}i+1}$ *starting from the initial marking. We define a run to be accepting if for the underlying sequence of transitions it holds that* $\forall i \geq 0, e \in \mathsf{Re}_{\mathsf{G}i}.\exists j \geq i.(e = e_j \vee e \notin \mathsf{In}_{\mathsf{G}j+1})$. *In words, a run is accepting if every response event either happen at some later state or become excluded.*

Condition *(iii)* in the above definition expresses that, only events that are currently included and mapped to the labels in L_{G} can be executed, Condition *(iv)* requires that all condition events to e which are currently included should have been executed previously. Condition *(v)*, *(vi)* and *(vii)* are the updates to the sets of executed, included events and required responses respectively. Note that an event e' can not be both included and excluded by the same event e, but an event may trigger itself as a response.

To ease keeping track of transition systems of different DCR Graphs we extend the transition system to transitions between graphs in the obvious way, writing $G \xrightarrow{(e,a)} G'$ if $G = (\mathsf{E}_{\mathsf{G}}, \mathsf{M}_{\mathsf{G}}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L}_{\mathsf{G}}, l_{\mathsf{G}})$, $\mathsf{M}_{\mathsf{G}} \xrightarrow{(e,a)} \mathsf{M}_{\mathsf{G}}'$ in $TS(G)$ and $G' = (\mathsf{E}_{\mathsf{G}}, \mathsf{M}_{\mathsf{G}}', \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L}_{\mathsf{G}}, l_{\mathsf{G}})$.

3 Projection and Composition

In this section we define projections and compositions of dynamic condition response graphs.

3.1 Projection

First we define how to project a DCR Graph G with respect to a *projection parameter* $\delta = (\mathsf{E}_\delta, \mathsf{L}_\delta)$, where $\mathsf{E}_\delta \subseteq \mathsf{E}_{\mathsf{G}}$ is a subset of the events of G and $\mathsf{L}_\delta \subseteq \mathsf{L}_{\mathsf{G}}$ is a subset of the labels.

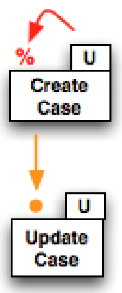
Intuitivly, the projection $G_{|\delta}$ contains only those events and relations that are relevant for the execution of events in E_δ and the labeling is restricted to the set L_δ. This includes both the events in E_δ and any other event that can affect the marking, or ability to execute of an event in E_δ through one or more relations.

Definition 3. *If* $G = (\mathsf{E}_{\mathsf{G}}, \mathsf{M}_{\mathsf{G}}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L}_{\mathsf{G}}, l)$ *then* $G_{|\delta} = (\mathsf{E}_{\mathsf{G}|\delta}, \mathsf{M}_{\mathsf{G}|\delta}, \rightarrow\!\bullet_{|\delta}, \bullet\!\rightarrow_{|\delta}, \pm_{|\delta}, \mathsf{L}_\delta, l_{|\delta})$ *is the projection of G with respect to* $\delta \subseteq \mathsf{E}_{\mathsf{G}}$ *where:*

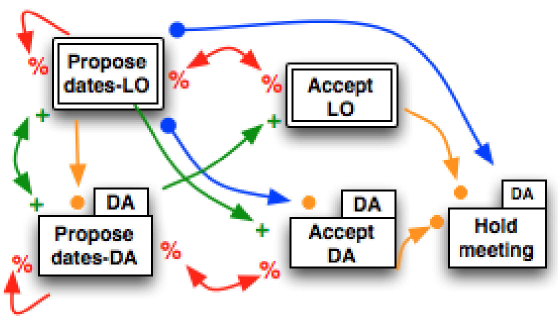
(i) $\mathsf{E}_{\mathsf{G}|\delta} = \rightarrow\mathsf{E}_\delta$, *for* $\rightarrow = \bigcup_{c \in C} c$, *and* $C = \{\mathsf{id}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!+, \rightarrow\!\%, \rightarrow\!+\!\rightarrow\!\bullet, \rightarrow\!\%\!\rightarrow\!\bullet\}$

(ii) $l_{|\delta}(e) = \begin{cases} l_{\mathsf{G}}(e) \cap \mathsf{L}_\delta & \text{if } e \in \mathsf{E}_\delta \\ \emptyset & \text{if } e \in \mathsf{E}_{\mathsf{G}|\delta} \backslash \mathsf{E}_\delta \end{cases}$

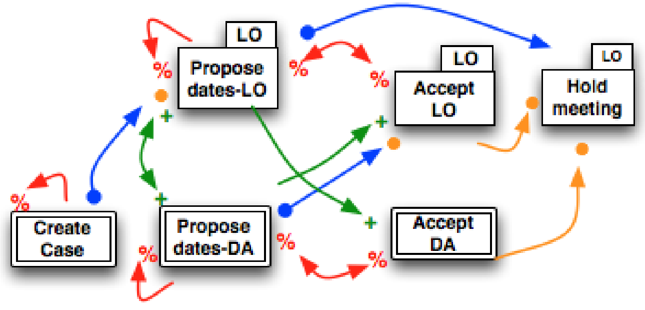
(iii) $\mathsf{M}_{\mathsf{G}|\delta} = (\mathsf{Ex}_{\mathsf{G}|\delta}, \mathsf{Re}_{\mathsf{G}|\delta}, \mathsf{In}_{\mathsf{G}|\delta})$ *where:*
- *(a)* $\mathsf{Ex}_{\mathsf{G}|\delta} = \mathsf{Ex}_{\mathsf{G}} \cap \mathsf{E}_{\mathsf{G}|\delta}$
- *(b)* $\mathsf{Re}_{\mathsf{G}|\delta} = \mathsf{Re}_{\mathsf{G}} \cap \mathsf{E}_\delta$
- *(c)* $\mathsf{In}_{\mathsf{G}|\delta} = (\mathsf{In}_{\mathsf{G}} \cap ((\mathsf{id} \cup \rightarrow\!\bullet)\mathsf{E}_\delta)) \cup (\mathsf{E}_{\mathsf{G}|\delta} \setminus ((\mathsf{id} \cup \rightarrow\!\bullet)\mathsf{E}_\delta))$



(a) Projection Over Role U



(b) Projection Over Role DA



(c) Projection Over Role LO

Fig. 3. Projecting of Arrange Meeting Example Over Roles

(iv) $\rightarrow\!\bullet_{|\delta} = \rightarrow\!\bullet \cap ((\rightarrow\!\bullet\ \mathsf{E}_\delta) \times \mathsf{E}_\delta)$
(v) $\bullet\!\!\rightarrow_{|\delta} = \bullet\!\!\rightarrow \cap ((\bullet\!\!\rightarrow\ \mathsf{E}_\delta) \times \mathsf{E}_\delta)$
(vi) $\rightarrow\!+_{|\delta} = \rightarrow\!+ \cap (((\rightarrow\!+\!\rightarrow\!\bullet\ \mathsf{E}_\delta) \times (\rightarrow\!\bullet\ \mathsf{E}_\delta)) \cup ((\rightarrow\!+\ \mathsf{E}_\delta) \times \mathsf{E}_\delta))$
(vii) $\rightarrow\!\%_{|\delta} = \rightarrow\!\% \cap (((\rightarrow\!\%\!\rightarrow\!\bullet\ \mathsf{E}_\delta) \times (\rightarrow\!\bullet\ \mathsf{E}_\delta)) \cup ((\rightarrow\!\%\ \mathsf{E}_\delta) \times \mathsf{E}_\delta))$

(i) defines the set of events as the union of the set E_δ of events that we project over, any event that has a direct relation towards an event in E_δ and events that exclude or include an event which is a condition for an event in E_δ. The additional events will be included in the projection without labels, as can be seen from the definition of the labeling function in *(ii)*. This means that the events can not be executed locally. However, when composed in a network containing other processes that can execute these events, their execution will be communicated to the process. For this reason we refer to these events as the (additional) external events of the projection. As proven in Prop. 1 the communication of the execution of this set of external events in addition to the local events shared by others ensure that the local state of the projection stay consistent with the global state. *(iii)* defines the projection of the marking: The executed events remain

the same, but are limited to the events in $\mathsf{E}_{\mathsf{G}|\delta}$. The responses are limited to events in E_δ because these are the only responses that will affect the local execution of the projected graph. The set of included events remains the same for events in E_δ or $\mathsf{E}_\delta{}^{\rightarrow\bullet}$, because these can affect which events are enabled in the projected graph. All other external events of the projected graph are included regardless of their state in the marking of the global graph. This is because in the local process is only notified of the execution of these events, not their in- or exclusion. Finally, *(iv)*, *(v)*, *(vi)* and *(vii)* state which relations should be included in the projection. For the events in E_δ all incoming relations should be included. Additionally inclusion and exclusion relations to events that are a condition for an event in E_δ are included as well.

To define networks of communicating DCR Graphs and their semantics we use the following extension of a DCR Graph allowing any event to be executed with a special input label. These transitions will only be used for the communication in a network and thus not be visible as user events.

Definition 4. *For a DCR Graph* $G = (\mathsf{E}_\mathsf{G}, \mathsf{M}_\mathsf{G}, \rightarrow\bullet, \bullet\rightarrow, \pm, \mathsf{L}_\mathsf{G}, l)$ *define* $G^\flat = (\mathsf{E}_\mathsf{G}, \mathsf{M}_\mathsf{G}, \rightarrow\bullet, \bullet\rightarrow, \pm, \mathsf{L}_\mathsf{G} \cup \{\flat\}, l^\flat)$*, where* $l^\flat = l(e) \cup \{\flat\}$ *(assuming that* $\flat \notin \mathsf{L}_\mathsf{G}$*).*

We are now ready to state the key correspondence between global execution of events and the local execution of events in a projection.

Proposition 1. *Let* $G = (\mathsf{E}_\mathsf{G}, \mathsf{M}_\mathsf{G}, \rightarrow\bullet, \bullet\rightarrow, \pm, \mathsf{L}_\mathsf{G}, l)$ *be a DCR Graph and* $G_{|\delta}$ *its projection with respect to a projection parameter* $\delta = (\mathsf{E}_\delta, \mathsf{L}_\delta)$*. Then*

1. *for* $e \in \mathsf{E}_\delta$ *it holds that* $G \xrightarrow{(e,a)} G'$ *if and only if* $G_{|\delta} \xrightarrow{(e,a)} G'_{|\delta}$,
2. *for* $e \notin \mathsf{E}_{\mathsf{G}|\delta}$ *it holds that* $G \xrightarrow{(e,a)} G'$ *implies* $G_{|\delta} = G'_{|\delta}$,
3. *for* $e \in \mathsf{E}_{\mathsf{G}|\delta}$ *it holds that* $G \xrightarrow{(e,a)} G'$ *implies* $(G_{|\delta})^\flat \xrightarrow{(e,\flat)} (G'_{|\delta})^\flat$,

3.2 Composition

Now we define the binary composition of two DCR Graphs. Intuitively, the *composition* of G_1 and G_2 glues together the events that are both in G_1 and G_2.

Definition 5. *Formally, the composite* $G_1 \oplus G_2 = (\mathsf{E}_\mathsf{G}, \mathsf{M}_\mathsf{G}, \rightarrow\bullet, \bullet\rightarrow, \pm, \mathsf{L}_\mathsf{G}, l)$*, where* $G_i = (\mathsf{E}_{\mathsf{G}i}, \mathsf{M}_{\mathsf{G}i}, \rightarrow\bullet_i, \bullet\rightarrow_i, \pm_i, \mathsf{L}_{\mathsf{G}i}, l_i)$, $\mathsf{M}_{\mathsf{G}i} = (\mathsf{Ex}_{\mathsf{G}i}, \mathsf{Re}_{\mathsf{G}i}, \mathsf{In}_{\mathsf{G}i})$ *and:*

(i) $\mathsf{E}_\mathsf{G} = (\mathsf{E}_{\mathsf{G}1} \cup \mathsf{E}_{\mathsf{G}2})$
(ii) $\mathsf{M}_\mathsf{G} = (\mathsf{Ex}_\mathsf{G}, \mathsf{Re}_\mathsf{G}, \mathsf{In}_\mathsf{G})$*, where:*
  (a) $\mathsf{Ex}_\mathsf{G} = \mathsf{Ex}_{\mathsf{G}1} \cup \mathsf{Ex}_{\mathsf{G}2}$
  (b) $\mathsf{In}_\mathsf{G} = (\mathsf{In}_{\mathsf{G}1} \cup \mathsf{In}_{\mathsf{G}2}) \setminus (((\mathsf{E}_{\mathsf{G}_1}^{i} \cup \rightarrow\bullet\mathsf{E}_{\mathsf{G}_1}^{i}) \setminus \mathsf{In}_{\mathsf{G}1}) \cup ((\mathsf{E}_{\mathsf{G}_2}^{i} \cup \rightarrow\bullet\mathsf{E}_{\mathsf{G}_2}^{i}) \setminus \mathsf{In}_{\mathsf{G}2}))$
  (c) $\mathsf{Re}_\mathsf{G} = \mathsf{Re}_{\mathsf{G}1} \cup \mathsf{Re}_{\mathsf{G}2}$
  for $\mathsf{E}_{\mathsf{G}_j}^{i} = \{e \in \mathsf{E}_{\mathsf{G}j} \mid l_j(e) \neq \emptyset\}$
(iii) $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ *for each* $\rightarrow \in \{\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\}$
(iv) $l(e) = l_1(e) \cup l_2(e)$
(v) $\mathsf{L}_\mathsf{G} = \mathsf{L}_{\mathsf{G}1} \cup \mathsf{L}_{\mathsf{G}2}$

(iib) states that events are included, if they're either included in G_1 or G_2, unless they are events that are either internal or have a condition towards an internal event and are excluded in G_1 or G_2. The intuition here is that if an event is internal or has a condition towards an internal event, then it affects the enabled events of the graph, so it's inclusion status should be the same in the composed graph. The inclusion/exclusion status of other external events however may simply not have been updated because the graph is not aware of all relations towards these events. This is not unsafe because the inclusion of these events does not affect the execution of the graph. Therefor the definition states that if an event is internal or has a condition towards an internal event in G_1 or G_2, then it's inclusion status should be the same in the composed graph, and in any other case the event is included if it was included in G_1 or G_2. *(iic)* states that the events with pending responses are those events that have a pending response in G_1 or G_2.

Definition 6. *The composition $G_1 \oplus G_2$ is* well-defined *when:*

(i) $\forall(e \in \mathsf{E}_{\mathsf{G}1} \cap \mathsf{E}_{\mathsf{G}2} \mid (e \in \mathsf{Ex}_{\mathsf{G}1} \Leftrightarrow e \in \mathsf{Ex}_{\mathsf{G}2})$
(ii) $\forall(e \in (\mathsf{E}^{i}_{\mathsf{G}1} \cup \rightarrow\!\bullet \mathsf{E}^{i}_{\mathsf{G}1}) \cap (\mathsf{E}^{i}_{\mathsf{G}2} \cup \rightarrow\!\bullet \mathsf{E}^{i}_{\mathsf{G}2}) \mid (e \in \mathsf{In}_{\mathsf{G}1} \Leftrightarrow e \in \mathsf{In}_{\mathsf{G}2})$
(iii) $\forall(e \in \mathsf{E}^{i}_{\mathsf{G}1} \cap \mathsf{E}^{i}_{\mathsf{G}2} \mid (e \in \mathsf{Re}_{\mathsf{G}1} \Leftrightarrow e \in \mathsf{Re}_{\mathsf{G}2})$
(iv) $\forall(e, e' \in \mathsf{E}_{\mathsf{G}1} \cap \mathsf{E}_{\mathsf{G}2} \mid \neg((e \rightarrow\!+_1 e' \wedge e \rightarrow\!\%_2 e') \vee (e \rightarrow\!\%_1 e' \wedge e \rightarrow\!+_2 e')))$

(i) ensures that those events that will be glued together have the same execution marking. *(ii)* ensures that events that will be glued together and in both DCR Graphs belong to either the set of internal events or the set of events that have a conditional relation towards an internal event, have the same inclusion marking. *(iii)* ensures that events that will be glued together and in both DCR Graphs belong to the set of internal events have the same pending response marking. *(iv)* ensures that by composing the two DCR Graphs no event both includes and excludes the same event. If $G_1 \oplus G_2$ is well-defined, then we also say that G_1 and G_2 are *composable* with respect to eachother.

Lemma 1. *The composition operator $\oplus$ is commutative and associative.*

Definition 7. *We call a vector $\Delta = \delta_1 \ldots \delta_k$ of projection parameters* covering *for some DCR Graph G* $= (\mathsf{E}_\mathsf{G}, \mathsf{M}_\mathsf{G}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, \mathsf{L}_\mathsf{G}, l_\mathsf{G})$ *if:*

1. $\bigcup_{i \in [k]} \mathsf{E}_{\delta i} = \mathsf{E}_\mathsf{G}$ *and*
2. $(\forall a \in \mathsf{L}_\mathsf{G}.\forall e \in \mathsf{E}_\mathsf{G}.a \in l_\mathsf{G}(e) \Rightarrow (\exists i \in [k].e \in \mathsf{E}_{\delta i} \wedge a \in \mathsf{L}_{\delta i})$

Proposition 2. *If some vector $\Delta = \delta_1 \ldots \delta_k$ of projection parameters is covering for some DCR Graph G then:* $\bigoplus_{i \in [k]} G_{|\delta_i} \; = \; G$

3.3 Safe Distributed Synchronous Execution

In this section we define networks of synchronously communicating DCR Graphs and prove the main technical theorem of the paper stating that a network of synchronously communicating DCR Graphs obtained by projecting a DCR Graph G with respect to a covering set of projection parameters has the same behavior as the original graph G.

Definition 8. *We define a* network of synchronously communicating DCR Graphs N *by the grammar*

$$N := G \mid N \| N$$

and let $\mathcal{N}_{\mathsf{E}\times\mathsf{L}}$ *be the set of all networks with events in* E *and labels in* L.

We write $\Pi_{i\in[n]}G_i$ *for* $G_1\|G_2\|\ldots\|G_n$. *We define the set of events of a network of graphs inductively by* $\mathcal{E}(G) = \mathsf{E}_\mathsf{G}$ *and* $\mathcal{E}(N_1\|N_2) = \mathcal{E}(N_1) \cup \mathcal{E}(N_2)$. *Similarly, we define the set of labels of a network of graphs inductively by* $\mathcal{L}(G) = \mathsf{L}_\mathsf{G}$ *and* $\mathcal{L}(N_1\|N_2) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2)$.

Definition 9. *The semantics of networks of buffered DCR Graphs are given by the following inference rules:*

input $$\frac{G_1^\flat \xrightarrow{(e,\flat)} G_2^\flat}{G_1 \overset{\triangleright e}{\rightsquigarrow} G_2}$$

sync input $$\frac{N_1 \overset{\triangleright e}{\rightsquigarrow} N_1' \quad N_2 \overset{\triangleright e}{\rightsquigarrow} N_2'}{N_1\|N_2 \overset{\triangleright e}{\rightsquigarrow} N_1'\|N_2'}$$

local input $$\frac{N_i \overset{\triangleright e}{\rightsquigarrow} N_i' \quad e \notin \mathcal{E}(N_{1-i})}{N_0\|N_1 \overset{\triangleright e}{\rightsquigarrow} N_0'\|N_1} \quad N_{1-i} = N_{1-i}', i \in \{0,1\}$$

sync step $$\frac{N_i \xrightarrow{(e,a)} N_i' \quad N_{1-i} \overset{\triangleright e}{\rightsquigarrow} N_{1-i}'}{N_0\|N_1 \xrightarrow{(e,a)} N_0'\|N_1'} \quad i \in \{0,1\}$$

local step $$\frac{N_i \xrightarrow{(e,a)} N_i' \quad e \notin \mathcal{E}(N_{i-1})}{N_0\|N_1 \xrightarrow{(e,a)} N_0'\|N_1} \quad N_{1-i} = N_{1-i}', i \in \{0,1\}$$

For a network of synchronously communicating DCR Graphs N we define the corresponding transition system $TS(N)$ *by* $(\mathcal{N}_{\mathcal{EL}(N)}, N, \mathcal{EL}(N), \rightarrow \subseteq \mathcal{N}_{\mathcal{EL}(N)} \times \mathcal{EL}(N) \times \mathcal{N}_{\mathcal{EL}(N)})$ *where* $\mathcal{EL}(N) = \mathcal{E}(N) \times \mathcal{L}(N)$ *and the transition relation* $\rightarrow \subseteq \mathcal{N}_{\mathcal{EL}(N)} \times \mathcal{EL}(N) \times \mathcal{N}_{\mathcal{EL}(N)}$ *is defined by the inference rules above.*

We define a run $a_0, a_1, \ldots$ *of the transition system to be a sequence of labels of a sequence of transitions* $N_i \xrightarrow{(e_i,a_i)} N_{i+1}$ *starting from the initial network. We define a run for a network* $N = \Pi_{i\in[n]}G_i$ *to be accepting if for the underlying sequence of transitions it holds that* $\forall j \in [n], \forall i \geq 0, e \in \mathsf{Re}_{\mathsf{G}j,i}.\exists k \geq i.(e = e_k \vee e \notin \mathsf{In}_{\mathsf{G}j,k+1})$, *where* $\mathsf{Re}_{\mathsf{G}j,i}$ *is the set of required responses in the* j*th DCR Graph in the network in the* i*th step of the run. In words, a run is accepting if every response event in a local DCR Graph in the network either happen at some later state or become excluded.*

We are now ready to give the main theorem of the paper, stating the correspondence between a global DCR Graph and the network of synchronously communicating DCR Graph obtained from a covering projection.

Theorem 1. *For a Dynamic Condition Response Graph G and a covering vector of projection parameters $\Delta = \delta_1 \ldots \delta_n$ it holds that $TS(G)$ is bisimilar to $TS(G_\Delta)$, where $G_\Delta = \Pi_{i \in [n]} G_{|\delta_i}$. Moreover, a run is accepting in $TS(G)$ if and only if the bisimilar run is accepting in $TS(G_\Delta)$.*

3.4 Example

In this section, we will use the arrange meeting example from Sec. 1 and show how events are executed in distributed setting. We assume the arrange meeting example is projected to a network $G^1_u \mid\mid G^1_{da} \mid\mid G^1_{lo}$ of three DCR Graphs as shown in the Fig. 3 and described in Sec. 3 and abbreviate the names for the events.

1. Using *sync step*, *local input*, and *input* we get the transition $G^1_u \mid\mid G^1_{da} \mid\mid G^1_{lo} \xrightarrow{(\texttt{Cc,U})} G^2_u \mid\mid G^1_{da} \mid\mid G^2_{lo}$ capturing the local execution of the event `Cc` labelled with `U` in G^1_u which is communicated synchronously to G^1_{lo}. This updates the markings by adding the event `Cc` to the set of executed events in both G^1_u and G^1_{lo}. But since `Cc` has an exclude relation to itself in both G^1_u and G^1_{lo} (see Fig. 3(a) and 3(c)), the event is also excluded from the set of included events in both markings. Finally, because of the response relation to the event `PdLO` in G^1_{lo} (see Fig. 3(c)), the event `PdLO` is added to the set of required responses in the resulting marking G^2_{lo}.
2. We can now execute the event `PdLO` in the DCR graph G^2_{lo} concurrently with the event `Uc` in DCR graph G^2_u.
   As the event `Uc` is only local to G^2_u we get by using *local step* the transition $G^2_u \mid\mid G^1_{da} \mid\mid G^2_{lo} \xrightarrow{(\texttt{Uc,U})} G^3_u \mid\mid G^1_{da} \mid\mid G^2_{lo}$ that only updates the marking of G^2_u.
   In addition to being local to G^2_{lo}, the event `PdLO` is also external event in graph G^1_{da}, so as in the first step by using *sync step local input*, and *input* we get the transition $G^3_u \mid\mid G^1_{da} \mid\mid G^2_{lo} \xrightarrow{(\texttt{PdLO,LO})} G^3_u \mid\mid G^2_{da} \mid\mid G^3_{lo}$, where the event `PdLO` has been added to the executed event set of both the marking of G^1_{da} and G^2_{lo}. Again, because of the self-exclusion relations, the event `PdLO` is also excluded from the sets of included events in the two markings, and because of the response relations, the events `ADA` and `Hm` are added to the set of pending responses in G^1_{da} and the event `Hm` is added to the set of pending responses in G^2_{lo}.
3. In response to the dates proposed by `LO`, the `DA` may choose to propose new dates by executing the event `PdDA` in the graph graph G^2_{da}.
   $G^3_u \mid\mid G^2_{da} \mid\mid G^3_{lo} \xrightarrow{(\texttt{PdDA,DA})} G^3_u \mid\mid G^3_{da} \mid\mid G^4_{lo}$ This triggers the exclusion of the events `PdDA` and `ADA` and the inclusion of the events `PdLO` and `ALO` in the markings of both G^2_{da} and G^3_{lo}. It will also include the event `ALO` in the required response set in the resulting marking G^4_{lo}.
4. Now `LO` may choose to accept the new dates proposed by `DA` by executing the event `ALO` in the graph graph G^4_{lo}, giving the transition
   $G^3_u \mid\mid G^3_{da} \mid\mid G^4_{lo} \xrightarrow{(\texttt{ALO,LO}))} G^3_u \mid\mid G^4_{da} \mid\mid G^5_{lo}$. This records the event `ALO` as executed in markings of both G^4_{da} and G^5_{lo} and excludes `PdLO` in both markings (i.e. it is not possible to propose new dates after acceptance).

5. Since the event ALO is recorded as executed in markings of both G^4_{da} and G^5_{lo} and the event ADA is excluded, the hold meeting event Hm will be enabled in both graphs G^5_{lo} and G^4_{da}. The LO may choose to hold the meeting, giving the transition $G^3_u \mid \rhd G^4_{da} \mid\mid G^5_{lo} \xrightarrow{(\texttt{Hm},\texttt{LO})} G^3_u \mid\mid G^5_{da} \mid\mid G^6_{lo}$
Note that this event is also communicated to DA, added to the set of executed events and removed from the set of pending responses. Since there are no pending responses in any of the local graphs the finite run is in an accepting state.

4 Conclusion

We have given a general technique for distributing a declarative (global) process as a network of synchronously communicating (local) declarative processes and proven the global and distributed execution to be equivalent.

The global and local processes are given as Dynamic Condition Response (DCR) Graphs, a recently introduced declarative process model generalizing labelled prime event structures to a systems model able to finitely represent ω-regular languages. The DCR Graph model has the advantage that it is on the one hand declarative and compositional, and on the other hand it has a simple and intuitive operational semantics given as a transition semantics between markings of the graph. This allows the model to be used both as specification and execution model.

As briefly surveyed in Sec. 1.1 there have been a lot of related work on synthesis of distributed systems and proving consistency with respect to a global model or property. We believe this is the first treatment where both the local and global models are given declaratively in the same model. This maintains the flexibility of a declarative model for the local processes, and allows local processes to be further distributed if necessary.

We exemplified the safe distribution technique on a process identified in a case study of an inter-organizational case management system carried out jointly with Exformatics A/S.

We leave for future work to study the harder problem of asynchronously communicating distributed processes. This may benefit from researching the true concurrency semantics inherent in the model and extend the transition semantics to include concurrency, e.g. like in [18, 28]. We also plan to study behavioral types describing the interfaces between communicating DCR Graphs, extending the work on session types in [4] to a declarative setting. Moreover, we intend to address extension of the DCR Graph model with time, data and dynamic instantiation of sub processes (also referred to multiple instances) to be able to model more realistic workflow processes. A first step is taken in [17] extending DCR Graphs to allow nested sub graphs. This extension introduced an additional relation between events, the milestone relation, making it possible to express the acceptance of a sub graph succinctly. We believe the results in the present paper can be extended to nested DCR Graphs and the milestone relation, although it will complicate the definition of projections.

Finally, we plan to continue the ongoing implementation of tools for DCR Graphs, and in particular to implement the safe distribution technique in the current prototype design and simulation tools briefly described in [16].

References

1. Wil M. P. van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, CAiSE '01, pages 140–156, 2001.
2. Mario Bravetti and Gianluigi Zavattaro. Contract based multi-party service composition. In *International Symposium on Fundamentals of Software Engineering (FSEN)*, volume 4767, pages 207–222. Springer, 2007.
3. Mario Bravetti and Gianluigi Zavattaro. A theory of contracts for strong service compliance. *Mathematical. Structures in Comp. Sci.*, 19:601–638, June 2009.
4. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *16th European Symposium on Programming (ESOP'07)*, LNCS, pages 2–17. Springer, 2007.
5. Ilaria Castellani, Madhavan Mukund, and P. Thiagarajan. Synthesizing distributed transition systems from global specifications. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1738, pages 219–231. Springer Berlin / Heidelberg, 1999.
6. David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
7. S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah. Orbwork: A reliable distributed corba-based workflow enactment system for meteor2. Technical report, The University of Georgia, 1996.
8. Dirk Fahland. Towards analyzing declarative workflows. In *Autonomous and Adaptive Web Services*, 2007.
9. Walid Fdhila and Claude Godart. Toward synchronization between decentralized orchestrations of composite web services. In *CollaborateCom'09*, pages 1–10, 2009.
10. Walid Fdhila, Ustun Yildiz, and Claude Godart. A flexible approach for automatic process decentralization using dependency tables. *International Conference on Web Services*, 2009.
11. Xiang Fu, Tevfik Bultan, and Jianwen Su. Realizability of conversation protocols with message contents. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 96–, Washington, DC, USA, 2004. IEEE Computer Society.
12. Keijo Heljanko and Alin Stefanescu. Complexity results for checking distributed implementability. In *Proceedings of the Fifth International Conference on Application of Concurrency to System Design*, pages 78–87, 2005.
13. Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. `http://www.trustcare.dk/`.
14. Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Post-proceedings of PLACES 2010*, 2010.
15. Thomas Hildebrandt and Raghava Rao Mukkamala. Distributed dynamic condition response structures. In *Pre-proceedings of International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 10)*, March 2010.
16. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proceedings of IEEE International EDOC Conference*, 2011. (to appear) `http://www.itu.dk/people/rao/pubs_accepted/dcrscasestudy-edoc11.pdf`.
17. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. to appear.
18. Thomas Hildebrandt and Vladimiro Sassone. Comparing transition systems with independence and asynchronous transition systems. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 84–97. Springer Berlin / Heidelberg, 1996.

19. R. Khalaf and F. Leymann. Role-based decomposition of business processes using BPEL. In *Web Services, 2006. ICWS '06. International Conference on*, pages 770 –780, sept. 2006.
20. Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 235–253, London, UK, 2000. Springer-Verlag.
21. Axel Martens. Analyzing web service based business processes. In *Fundamental Approaches to Software Engineering*. Springer Berlin / Heidelberg, 2005.
22. Zoran Milosevic, Shazia Sadiq, and Maria Orlowska. Towards a methodology for deriving contract-compliant business processes. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 395–400. Springer Berlin / Heidelberg, 2006.
23. Saayan Mitra, Ratnesh Kumar, and Samik Basu. Optimum decentralized choreography for web services composition. In *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*, 2008.
24. C. Mohan, D. Agrawal, G. Alonso, A. El Abbadi, R. Guenthoer, and M. Kamath. Exotica: a project on advanced transaction management and workflow systems. *SIGOIS Bull.*, 16:45–50, August 1995.
25. Marco Montali. *Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.
26. Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010.
27. M. Mukund. From global specifications to distributed implementations. In *Synthesis and Control of Discrete Event Systems*. Springer, 2002.
28. Madhavan Mukund and Mogens Nielsen. Ccs, locations and asynchronous transition systems. In Rudrapatna Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 328–341. Springer Berlin / Heidelberg, 1992.
29. Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. *SIGPLAN Not.*, 39:170–187, October 2004.
30. OASIS WSBPEL Technical Committee. Web Services Business Process Execution Language, version 2.0, 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`.
31. Santanu Paul, Edwin Park, and Jarir Chaar. Rainman: a workflow system for the internet. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, 1997.
32. F. Ranno and S. K. Shrivastava. A review of distributed workflow management systems. In *Proceedings of the international joint conference on Work activities coordination and collaboration*, 1999.
33. M. U. Reichert, T. Bauer, and P. Dadam. Flexibility for distributed workflows. In *Handbook of Research on Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments*, pages 137–171. IGI Global, Hershey, PA, 2009.
34. Manfred Reichert and Thomas Bauer. Supporting ad-hoc changes in distributed workflow management systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 150–168. Springer Berlin / Heidelberg, 2007.
35. Stefanie Rinderle, Andreas Wombacher, and Manfred Reichert. Evolution of process choreographies in dychor. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *LNCS*, pages 273–290. Springer, 2006.

36. W. Sadiq, S. Sadiq, and K. Schulz. Model driven distribution of collaborative business processes. In *Services Computing, 2006. SCC '06. IEEE International Conference on*, pages 281 –284, sept. 2006.
37. V. Sassone, M. Nielsen, and G. Winskel. A classification of models for concurrency. In *4th International Conference on Concurrency Theory, CONCUR '93.*, volume Lectur of *LNCS*, pages 82–96. Springer, 1993.
38. Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170:297–348, 1996.
39. Wheater Shrivastava, S. M. Wheater, S. K. Shrivastava, and F. Ranno. A corba compliant transactional workflow system for internet applications. In *Proc. Of IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware 98*, pages 1–85233. Springer-Verlag, 1998.
40. Arthur ter Hofstede, Rob van Glabbeek, and David Stork. Query nets: Interacting workflow modules that ensure global termination. In *Business Process Management*. Springer Berlin / Heidelberg, 2003.
41. W. M. P. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
42. Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. *The Computer Journal*, 53(1):90–106, January 2010.
43. Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
44. Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006.
45. Wil M.P van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In M. Bravetti, M. Nunez, and Gianluigi Zavattaro, editors, *Proceedings of Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *LNCS*, pages 1–23. Springer Verlag, 2006.
46. W.M.P. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Information Technology and Management*, 4:345–389, 2003.
47. Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.
48. Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In *Proceedings of the 6th International Conference on Database Theory*, pages 230–246, London, UK, 1997. Springer-Verlag.
49. X. Yi and K.J Kochut. Process composition of web services with complex conversation protocols. In *Design, Analysis, and Simulation of Distributed Systems Symposium at Adavanced Simulation Technology*, 2004.
50. W. Zielonka. Notes on finite asynchronous automata. *Informatique Thorique et Applications*, 21(2):99–135, 1987.

Declarative Modelling and Safe Distribution of Healthcare Workflows

Thomas Hildebrandt[1], Raghava Rao Mukkamala[1], and Tijs Slaats[1,2] ⋆

[1] IT University of Copenhagen
Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
{hilde, rao, tslaats}@itu.dk, `http://www.itu.dk`
[2] Exformatics A/S, 2100 Copenhagen, Denmark

Abstract. We present a formal technique for safe distribution of workflow processes described declaratively as nested Dynamic Condition Response (DCR) Graphs and apply the technique to a distributed healthcare workflow. Concretely, we provide a method to synthesize from a nested DCR Graph and any distribution of its atomic events a set of local process graphs communicating by shared events, such that the distributed execution of the local processes is equivalent to executing the original process. The technique extends our recent work on safe distribution of non-nested DCR Graphs applied to cross-organizational case management. The main contributions of the present paper is to adapt the technique to allow for nested processes and milestones and to apply it to a healthcare workflow identified in a previous field study at danish hospitals. We also provide a new formalization of the semantics of DCR Graphs which highlights its declarative nature.

1 Introduction

The overall goal of the interdisciplinary Trustworthy Pervasive Healthcare Services (TrustCare) project [16] is to develop a foundation for trustworthy it-supported healthcare workflows. Healthcare workflows involve coordination of a heterogeneous set of professionals, patients, organizations and sectors, and must be able to adapt to inevitable changes of treatment processes and organization of the work [23, 40]. This challenges traditional workflow management systems using an imperative process modeling language such as Business Process Model and Notation (BPMN) [35] in which the control flow is modeled explicitly. Typically a flow diagram will only cover the normal flow and a few possible exceptionally flows, leading to rigid and inflexible, over-specified workflows. Declarative process languages, allowing any flow that fulfill the specified constraints, have been suggested by a number of researchers as being more appropriate for representing workflow processes requiring a high degree of flexibility [8–10, 33, 43].

The TrustCare project combines research in pervasive user-interfaces [2, 3] with research in formal logic and domain specific process languages, taking as starting point [31]

⋆ Authors listed alphabetically. This research is supported by the Danish Research Agency through the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu) and the Computer Supported Mobile Adaptive Business Processes project (grant #274-06-0415, www.cosmobiz.dk).

the declarative workflow process language developed and used by Resultmaker, the industrial partner of the project.

The present paper focus on and extends our work on formal process languages, in particular the development of a basic formal declarative workflow language called Dynamic Condition Response Graphs (DCR Graphs) introduced in [17] and extended to allow nested (i.e. hierarchical) process definitions in [19, 20] and a new milestone relation between activities. In [21] we have shown how, given a (non-nested) DCR Graph describing a global, collaborative process and any distribution of the activities, to derive a set of local DCR Graphs corresponding to the activity distribution and achieving the same global behavior by synchronously communicating the relevant events between the local processes. The main new contributions of the present paper is to adapt the distribution technique given in [21] to allow for nested processes and the new milestone relation between activities as introduced in [20] and demonstrate the use of the technique on an oncology healthcare workflow previously identified during a field study at danish hospitals [26]. The workflow was described loc. cit. using an early formalization of the Resultmaker workflow process language. Another contribution of the present paper is to formalize the workflow process as a nested DCR Graphs. Finally, we also provide a new presentation of the formal semantics of DCR Graphs that highlights the declarative nature. The present paper extends the results presented in the pre-proceedings of FHIES 2011 [22] which omitted the primitives of DCR Graphs for dynamically changing the set of included activities in the workflow.

The intention of the oncology healthcare workflow is to illustrate two important kinds of flexibility appearing in healthcare workflows: 1) The need for reconsidering a previous activity if its validity at a later stage is questioned by a co-worker and 2) The need for distribution of collaborative tasks and ability to tailor this distribution to local conditions (e.g. the size and organization of work within a hospital). These needs have also been identified during field studies of case management processes [19] and appears to be generally relevant for knowledge work processes and not only healthcare workflows.

The rest of the paper is structured as follows. In Sec. 1.1 ending the introduction we briefly discuss related work. We present the oncology workflow example in Sec. 2 as a nested Dynamic Condition Response (DCR) Graph, describing the semantics informally. In Sec. 3 we recall the formal definition of nested DCR Graphs and provide a new presentation of their semantics. We then in Sec. 4 formalize the distribution technique and exemplify it on the oncology workflow. Finally we conclude in Sec. 5.

1.1 Related Work

The problem of verifying the correctness of cross-organizational workflows described as variants of Petri nets has been studied in [1, 25, 27, 39, 41, 42, 46] and models of global behavior based on conversations among participating services have been studied in [4, 5, 14, 36, 48, 49]. A technique to partition a composite web service using program analysis was studied in [34] and using a similar approach, [24] explored decomposition of a business process modeled in BPEL, primarily focussing on P2P interactions . Using a formal approach based on I/O automata representing the services, the authors in [29] have studied the problem of synthesizing a decentralized choreography strategy, that will have optimal overhead of service composition in terms of costs associated with each interaction.

The derivation of descriptions of local components from a global model has also been researched in the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [6]. The work formalizes the core of WS-CDL as the global process calculus and defines a formal theory of end-point projections projecting the global process calculus to abstract descriptions of the behavior of each of the local ”end-points” given as pi-calculus processes typed with session types.

A methodology for deriving process descriptions from a business contract formalized in a formal contract language was studied in [28], while [38] proposes an approach to extract a distributed process model from collaborative business process. In [12, 13], the authors have proposed a technique for the flexible decentralization of a process specification with necessary synchronization between the processing entities using dependency tables. In [7,15,32] foundational work has been made on synthesizing distributed transition systems from global specification for the models of synchronous product and asynchronous automata [50].

The formalisms discussed above are all confined to imperative modeling languages such as Petri nets, workflow/open nets and automata based languages. To the best of our knowledge, there exists very few works on distributed cross-organizational workflows which consider declarative modeling languages and none where both the global and local processes are given declaratively using the same formalism. In [11], Fahland has studied synthesizing declarative workflows expressed in DecSerFlow [45] by translating to Petri nets. Only a predefined set of DecSerFlow constraints are used in the mapping to the Petri nets patterns, so this approach has a limitation with regards to the extensibility of the DecSerFlow language. On the other hand, in [30] Montali has studied the composition of ConDec [44] models with respect to conformance with a given choreography, based on the compatibility of the local ConDec models. But his study was limited to only composition of local models, whereas the problem of splitting a global model in local models has not been studied.

2 Distributed Declarative Healthcare Workflows by Example

In Fig. 1 below we show the graphical representation of the nested Dynamic Condition Response Graph formalizing a variant of the oncology workflow studied in [26]. In this section we informally describe the formalism and the distribution technique formalized in the rest of the paper using the example workflow. For details of the field study and the workflow we refer the reader to [26].

The boxes denote *activities* (also referred to as events in the following sections). Administer medicine is a *nested* activity having sub activities give medicine and trust. Give medicine is an *atomic* activity, i.e. it has no sub activities. Trust is again a nested activity having sub activities sign nurse 1 and sign nurse 2. Finally, medicine preparation is a nested activity having seven sub activities dealing with the preparation of medicine. An activity may be either included or excluded, the latter are drawn as a dashed box as e.g. the edit and cancel activities.

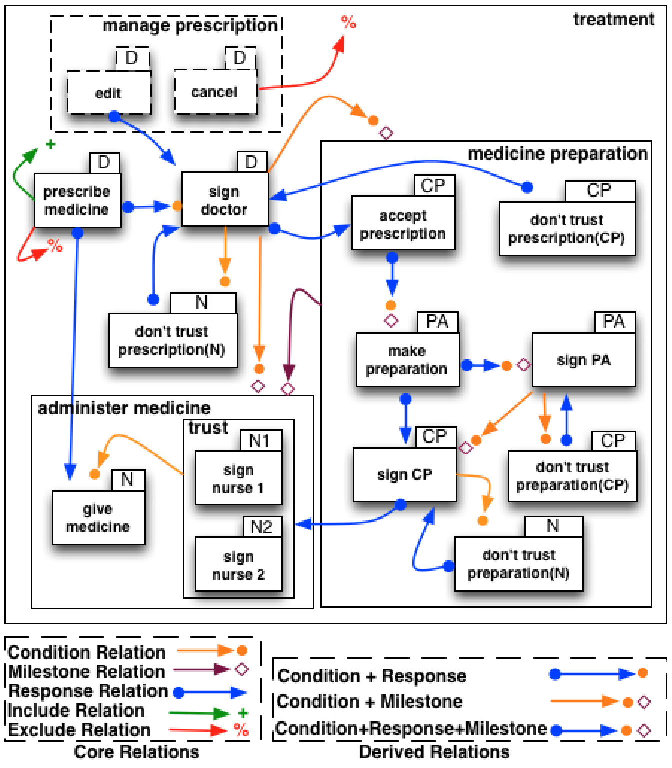


Fig. 1. Oncology Workflow as a nested DCR Graph

A *run* of the workflow consists of a (possibly infinite) sequence of executions of atomic activities. (A nested activity is considered executed when all its sub activities are executed). An activity can be executed any number of times during a run, as long as the activity is included and the constraints for executing it are satisfied, in which case we say the activity is *enabled*.

The constraints and dynamic exclusion and inclusion are expressed as five different core relations between activities represented as arrows in the figure above: The *condition relation*, the *response relation*, the *milestone relation*, the *include relation*, and the *exclude relation*.

The condition relation is represented by an orange arrow with a bullet at the arrow head. E.g. the condition relation from the activity sign doctor to the activity don't trust prescription(N) means that sign doctor must have been executed at least once before the activity don't trust prescription(N) can be executed.

The response relation is represented by a blue arrow with a bullet at its source. E.g. the response relation from the activity prescribe medicine to the activity give medicine means that the latter must be executed (at some point) after (any execution of) the activity prescribe medicine. We say that a workflow is in a *completed* state if all such response constraints have been fulfilled (or the required response activity is excluded). However, note that a workflow may be continued from a completed state and change to a non-completed state if an activity is executed that requires another response or includes an activity which has not been executed since it was last required as a response. Also note that the response constraint may cause some infinite runs to never pass through a complete state if the executed activities keep triggering new responses. In the following section we make precise when such infinite runs can be regarded as a complete execution.

The third core relation used in the example is the *milestone relation* represented as a dark red arrow with a diamond at the arrow head. The milestone relation was introduced in [20] jointly with the ability to nest activities. A relation to and/or from a nested activity simply unfolds to relations between all sub activities. A milestone relation from a nested activity to another activity then in particular means that the entire nested activity must be in a completed state before that activity can be executed. E.g. medicine preparation is a milestone for the activity administer medicine, which means that none of the sub activities of administer medicine can be carried out if any one of the sub activities of medicine preparation is included and has not been executed since it was required as a response.

Two activities can be related by any combination of these relations. In the graphical notation we have employed some shorthands, e.g. indicating the combination of a condition and a response relation by and arrow with a bullet in both ends.

Finally, DCR Graphs allow two relations for dynamic exclusion and dynamic inclusion of activities represented as a green arrow with a plus at the arrow head and a red arrow with a minus at the arrow head respectively. The exclusion relation is used in the example between the cancel activity and the treatment activity. Since all other activities in the workflow are sub activities of the treatment activity this means that all activities are excluded if the cancel activity is executed. The inclusion relation is used between the prescribe medicine activity and the manage prescription activity

The run-time state of a nested DCR Graph can be formally represented as a pair $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ of sets of atomic activities (referred to as the *marking* of the graph). The set Ex is the set of atomic activities that have been executed at least once during the run. The set Re is the set of atomic activities that, if included, are required to be executed at least one more time in the future as the result of a response constraint (i.e. they are pending responses). Finally, the set In denotes the currently included activities.

The set Ex thus may be regarded as a set of completed activities, the set Re as the set of activities on the to-do list and the set In as the activities that are currently relevant for the workflow.

Note that an activity may be completed once and still be on the to-do list, which simply means that it must be executed (completed) again. This makes it very simple to model the situation where an activity needs to be (re)considered as a response to the execution of an activity. In the oncology example this is e.g. the case for the response relation between the don't trust prescription(N) activity (representing that a nurse reports that he doesn't trust the prescription) and the sign doctor activity. The effect is that the doctor is asked to reconsider her signature on the prescription. In doing that she may or may not decide to change the prescription, i.e. execute prescribe medicine again.

We indicate the marking graphically by adding a check mark to every atomic activity that has been executed (i.e. is included in the set Ex of the marking), an exclamation mark to every atomic activity which, if included, is required to be executed at least once more in the future (i.e. is included in the set Re), and making a box dashed if the activity is not included (i.e. is not included in the set In of the marking). In Fig. 1 we have shown an example marking where prescribe medicine has been executed. This has caused manage prescription and its sub activities edit and cancel to be included, and sign doctor and give medicine to be required as responses, i.e the two activities are included in the set Re of the marking (on the to-do list).

As described above, an activity can be executed if it is enabled. Sign doctor is enabled for execution in the example marking, since its only condition (prescribe medicine) has been executed and it has no milestones. Give medicine on the other hand is not enabled since it has the (nested) activity trust as condition, which means that all sub activities of trust (sign nurse 1 and sign nurse 2) must be executed before give medicine is enabled. Also, both give medicine and trust are sub activities of administer medicine which further has sign doctor as condition and milestone, and medicine preparation as milestone. The condition relation from sign doctor means that the prescription must be signed before the medicine can be administered. The milestone relations means that the medicine can not be given as long as sign doctor or any of the sub activities of medicine preparation is on the to-do list (i.e. in the set Re of pending responses).

Most activities should only be available to a subset of the users of the workflow system. For this reason the commercial implementation of the workflow management system provided by Resultmaker employs a role based access control, assigning to every atomic activity a finite set of roles and assigning to every role a set of access rights controlling if the activity is invisible or visible to users fulfilling the role. If an activity is visible it is specified wether the role is allowed to execute the activity or not. Users are either statically (e.g. by login) or dynamically assigned to roles (e.g. by email invitation).

In the formalization presented in this paper, the assigned roles are given as part of the name of the activity. In the graphical representation we have shown the roles within small ”ears” on the boxes. In the example workflow we have the following different roles: Doctor (D), Controlling Pharmacist (CP), Pharmacist Assistant (PA) and Nurse (N). Hereto comes roles N1 and N2 which must dynamically be assigned to two different authorized persons (nurses or doctors). This is at present the only way to implement the constraint stating that two different authorized persons must sign the product prepared by the pharmacists before the medicine is administered to the patient. Future work will address less ad hoc ways to handle these kind of constraints between activities referring to the identify of users.

The commercial implementation is based on a centralized workflow manager controlling the execution of the entire, global workflow. However, workflows often span different units or departments within the organization, e.g. the pharmacy and the patient areas, or even cross boundaries of different organizations (e.g. different hospitals). In some situations it may be very relevant to execute the local parts of the workflow on a local (e.g. mobile) device without permanent access to a network, e.g. during preparation of the medicine in the pharmacy. Also, different organizations may want to keep control of their own parts of the workflow and not delegate the management to a central service. This motivates the ability to split the workflow in separate components, each only referring to the activities relevant for the local unit and being manageable independently of the other components.

The technique for distributing DCR Graphs introduced in [21] and extended in the present paper is a first step towards supporting this kind of splitting of workflow definitions. Given any division of activities on local units (assigning every activity to at least one unit) it describes how to derive a set of graphs, one for each unit, describing the local part of the workflow. Such a local process, referred to as a *projection* is again a DCR Graph. It includes the activities assigned to the unit but also the relevant *external* activities executed within other units for which an event must be send to the local process when they are executed. An example of a projection relative to the activities assigned the doctor role (D) is given in Fig. 2(a) in Sec. 4. The diagram shows that the projection also includes the two external activities (indicated as double line boxes) don't trust prescription (N) and don't trust prescription (CP). These two activities, representing respectively a nurse and a controlling pharmacist reporting that the prescription is not trusted, are the only external activities that may influence the workflow of the doctor by requiring sign doctor as a response. Similarly, Fig. 2(b),2(c), and 2(d) shows projections corresponding to the nurse, controlling pharmacist, and pharmacist assistant roles. However, if for instance the roles of the controlling pharmacist and the pharmacist assistant are always assigned to the same persons one may instead choose to keep all these activities together in a unit. This can be obtained by simply projecting on all activities assigned either the CP or the PA role.

3 Nested Dynamic Condition Response Graphs

Dynamic Condition Response Graphs (DCR Graphs) [17] is both a generalization of labelled event structures [47] and the Process Matrix workflow model developed by Resultmaker, our industrial partner in the TrustCare research project. The DCR Graphs were extended in [20] to Nested DCR Graphs, supporting sub graphs and a new milestone relation, motivated by a case study of cross-organizational case management [19]. Further in [21], we have considered safe distribution of DCR Graphs without milestone relation where as in [18], we have defined projection and distribution for a restricted nested model Condition Response Graphs, simplified by not allowing the dynamic inclusion and exclusion. In the present paper, we will consider full version of Nested DCR Graphs with both milestone and dynamic inclusion/exclusion relations and provide distribution of nested DCR Graphs. We employ the following notations in the paper.

Notation: For a set A we write $\mathcal{P}(A)$ for the power set of A. For a binary relation $\rightarrow \subseteq A \times A$ and a subset $\xi \subseteq A$ of A we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{a \in A \mid (\exists a' \in \xi \mid a \rightarrow a')\}$ and the set $\{a \in A \mid (\exists a' \in \xi \mid a' \rightarrow a)\}$ respectively. Also, we write $\rightarrow^{-1}$ for the inverse relation. Finally, for a natural number k we write $[k]$ for the set $\{1, 2, \ldots, k\}$.

We then formally define nested dynamic condition response graph as follows.

Definition 1. *A Dynamic Condition Response Graph (DCR Graph) G is a tuple* $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where*

(i) E *is the set of* events *(or activities),*
(ii) $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \in \mathcal{M}(G)$ *is the* marking, *for* $\mathcal{M}(G) =_{def} \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$,
(iii) $\rightarrow\!\bullet \subseteq \mathsf{E} \times \mathsf{E}$ *is the* condition *relation,*
(iv) $\bullet\!\rightarrow \subseteq \mathsf{E} \times \mathsf{E}$ *is the* response *relation,*
(v) $\rightarrow\!\diamond \subseteq \mathsf{E} \times \mathsf{E}$ *is the* milestone *relation,*
(vi) $\rightarrow\!+, \rightarrow\!\% \subseteq \mathsf{E} \times \mathsf{E}$ *is the dynamic* include *relation and* exclude *relation, satisfying that* $\forall e \in \mathsf{E}. e \rightarrow\!+ \cap e \rightarrow\!\% = \emptyset$,
(vii) L *is the set of* labels,
(viii) $l : \mathsf{E} \rightarrow \mathcal{P}(\mathsf{L})$ *is a labeling function mapping events to sets of labels.*

A Nested Dynamic Condition Response Graph (Nested DCR Graph) G is a tuple $(\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where*

(i) $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *is a DCR Graph,*
(ii) $\rhd : \mathsf{E} \rightharpoonup \mathsf{E}$ *is a partial function mapping an event to its super-event (if defined),*
(iii) $\mathsf{M} \in \mathcal{P}(\mathsf{atoms}(\mathsf{E})) \times \mathcal{P}(\mathsf{atoms}(\mathsf{E})) \times \mathcal{P}(\mathsf{atoms}(\mathsf{E}))$*, where* $\mathsf{atoms}(\mathsf{E}) = \mathsf{E} \backslash \{e \in \mathsf{E} \mid \exists e' \in \mathsf{E}. \rhd(e') = e\}$ *is the set of atomic events.*

We write $e \rhd e'$ *if* $e' = \rhd^k(e)$ *for* $0 < k$ *and write* $e \unrhd e'$ *if* $e \rhd e'$ *or* $e = e'$*, and* $e \unlhd e'$ *if* $e' \rhd e$ *or* $e = e'$*. We require that the resulting relation,* $\unrhd \subset E \times E$*, referred to as the nesting relation, is a well founded partial order. We also require that the nesting relation is consistent with respect to dynamic inclusion/exclusion in the following sense: If* $e \rhd e'$ *or* $e' \rhd e$ *then* $e \rightarrow\!+ \cap e' \rightarrow\!\% = \emptyset$.

We already introduced the graphical notation for Nested DCR Graphs by example in the previous section. We will not write out the complete formal specification. The events are all boxes, e.g. $\mathsf{E} = \{\mathsf{treatment}, \mathsf{manage\,prescription}, \mathsf{prescribe\,medicine}, ...\}$, the nesting relation captures the inclusion of boxes, e.g. $\rhd(e) = \mathsf{administer\,medicine}$, if $e \in \{\mathsf{give\,medicine}, \mathsf{trust}\}$ and $\rhd(e) = \mathsf{trust}$, if $e \in \{\mathsf{sign\,nurse1}, \mathsf{sign\,nurse2}\}$ and so forth. The inital marking is the triple $M = (\emptyset, \emptyset, \mathsf{E} \backslash \{\mathsf{manage\,prescription}, \mathsf{edit}, \mathsf{cancel}\})$, meaning no events have been executed, no events are initially required as responses and all events except the events $\{\mathsf{manage\,prescription}, \mathsf{edit}, \mathsf{cancel}\}$ are included. The condition relation includes e.g. the pairs $\mathsf{sign\,doctor} \rightarrow\!\bullet \mathsf{don't\,trust\,prescription(N)}$ and $\mathsf{trust} \rightarrow\!\bullet \mathsf{give\,medicine}$, the response relation includes e.g. the pairs $\mathsf{prescribe\,medicine} \bullet\!\rightarrow \mathsf{sign\,doctor}$ and $\mathsf{edit} \bullet\!\rightarrow \mathsf{sign\,doctor}$, the milestone relation includes e.g. the pairs $\mathsf{sign\,doctor} \rightarrow\!\diamond \mathsf{administer\,medicine}$ and $\mathsf{sign\,PA} \rightarrow\!\diamond \mathsf{sign\,CP}$, the dynamic inclusion relation includes the single pair $\mathsf{prescribe\,medicine} \rightarrow\!+ \mathsf{manage\,prescription}$ and the dynamic exclusion includes exactly the two pairs $\mathsf{prescribe\,medicine} \rightarrow\!\% \mathsf{prescribe\,medicine}$ and $\mathsf{cancel} \rightarrow\!\% \mathsf{treatment}$. We take as labels pairs of action names and roles, i.e. the set of labels L includes e.g. the pairs $(\mathsf{edit}, \mathsf{D})$, $(\mathsf{cancel}, \mathsf{D})$, $(\mathsf{give\,medicine}, \mathsf{N})$, and $(\mathsf{sign\,PA}, \mathsf{PA})$. Super events with no role assigned such as $\mathsf{manage\,prescription}$ are assigned the empty set of labels.

Note that the labels of events consist of the name of the event and a role which defines who can execute that event. In our implementation every event can be assigned any number of roles and every user of the system can have multiple roles. A user can then execute an event if she has at least one role that is assigned to the event.

To define the execution semantics for Nested DCR Graphs we first define how to flatten a nested graph to the simpler DCR Graph. Essentially, all relations to and/or from nested events are extended to sub events, and then only the atomic events are preserved.

Definition 2. *For a Nested DCR Graph* $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *define the underlying flat Dynamic Condition Response Graph as*

$$G^{\flat}_{l_f} = (\mathsf{atoms}(\mathsf{E}), \mathsf{M}, \rightarrow\!\bullet^{\flat}, \bullet\!\rightarrow^{\flat}, \rightarrow\!\diamond^{\flat}, \rightarrow\!+^{\flat}, \rightarrow\!\%^{\flat}, \mathsf{L}, l)$$

where $rel^{\flat} = \unrhd rel \unlhd$ *for some relation* $rel \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$.

It is easy to see from the definition that the underlying DCR Graph has at most as many events as the nested graph and that the size of the relations may increase by an order of n^2 where n is the number of atomic events.

Below we give a new presentation of the semantics of DCR Graphs stressing its declarative nature.

First we formalize in Def. 3 that an event e of a (flat) DCR Graph is enabled when it is included in current marking ($e \in \mathsf{In}$), all the included events that are conditions for it are in the set of executed events (i.e. $(\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$) and none of the included events that are milestones for it are in the set of pending response events (i.e. $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$).

Definition 3. *For a Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, and* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *we define that an event* $e \in \mathsf{E}$ *is* enabled*, written* $G \vdash e$*, if* $e \in \mathsf{In}$*,* $(\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$ *and* $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

Def. *4* below then defines the change of the marking when an enabled event is executed: First the event is added to the set of executed events and removed from the set of pending responses. Then all events that are a response to the event are added to the set of pending responses. Note that if an event is a response to itself, it will remain in the set of pending responses after execution. Similarly, the included events set will updated by adding all the events that are included by the event and by removing all the events that are excluded by the event.

Definition 4. *For a Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *and an enabled event* $G \vdash e$ *, the result of executing* e *is a Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M}', \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where* $\mathsf{M}' = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e =_{def} \big(\mathsf{Ex} \cup \{e\}, (\mathsf{Re} \backslash \{e\}) \cup e\bullet\!\rightarrow, (\mathsf{In} \cup e\rightarrow\!+) \setminus e\rightarrow\!\%\big)$.

We now define the semantics for Nested DCR Graph by using the corresponding flat graph.

Definition 5. *For a Nested Condition Response Graph* $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *we define that* $e \in \mathsf{atoms}(\mathsf{E})$ *is enabled, written* $G \vdash e$*, if* $G^{\flat} \vdash e$*. Similarly, the result of executing* $G \vdash e$ *is defined as:* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_{G^{\flat}} e$.

As an example, in the intial marking $M = (\emptyset, \emptyset, \mathsf{E}\backslash\{\mathsf{manage\ prescription}, \mathsf{edit}, \mathsf{cancel}\})$ we have that $G \vdash$ prescribe medicine, i.e. the event prescribe medicine is enabled.

After executing prescribe medicine the new marking $M' = M \oplus_G$prescribe medicine $=$ $(\{\mathsf{prescribe\ medicine}\}, \{\mathsf{sign\ doctor}, \mathsf{give\ medicine}\}, \mathsf{E}\backslash\{\mathsf{prescribe\ medicine}\})$. That is, prescribe medicine is added to the set of executed events, and sign doctor and give medicine are added to the set of pending responses, because prescribe medicine $\bullet\!\!\rightarrow$ sign doctor and prescribe medicine $\bullet\!\!\rightarrow$ give medicine}. The event prescribe medicine is removed from the set of included events because prescribe medicine $\rightarrow\!\%$ prescribe medicine. The events $\{\mathsf{manage\ prescription}, \mathsf{edit}, \mathsf{cancel}\}$ are included since prescribe medicine $\rightarrow\!+$ manage prescription, and the inclusion relation is "flattened" to include also prescribe medicine $\rightarrow\!+$ edit and prescribe medicine $\rightarrow\!+$ cancel.

From the definition of enabling and execution above we can construct a labelled transition semantics for a nested DCR Graphs, with acceptance conditions for finite and infinite computations.

Definition 6. *For a nested DCR Graph* $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *we write* $G \xrightarrow{(e,a)} G'$ *if* $G' = (\mathsf{E}, \rhd, \mathsf{M}', \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*,* $G \vdash e$*,* $a \in l(e)$*, and* $\mathsf{M}' = \mathsf{M} \oplus_{G^\flat} e$*. We then define the corresponding labelled transition system* $TS(G)$ *to be the tuple* $(\mathcal{G}_{\mathsf{E}\times\mathsf{L}}, G, \mathsf{E}\times\mathsf{L}, \rightarrow \subseteq \mathcal{G}_{\mathsf{E}\times\mathsf{L}} \times \mathsf{E} \times \mathsf{L} \times \mathcal{G}_{\mathsf{E}\times\mathsf{L}})$ *where* $\mathcal{G}_{\mathsf{E}\times\mathsf{L}}$ *is the set of all nested DCR Graphs with events in* E *and labels in* L*.*

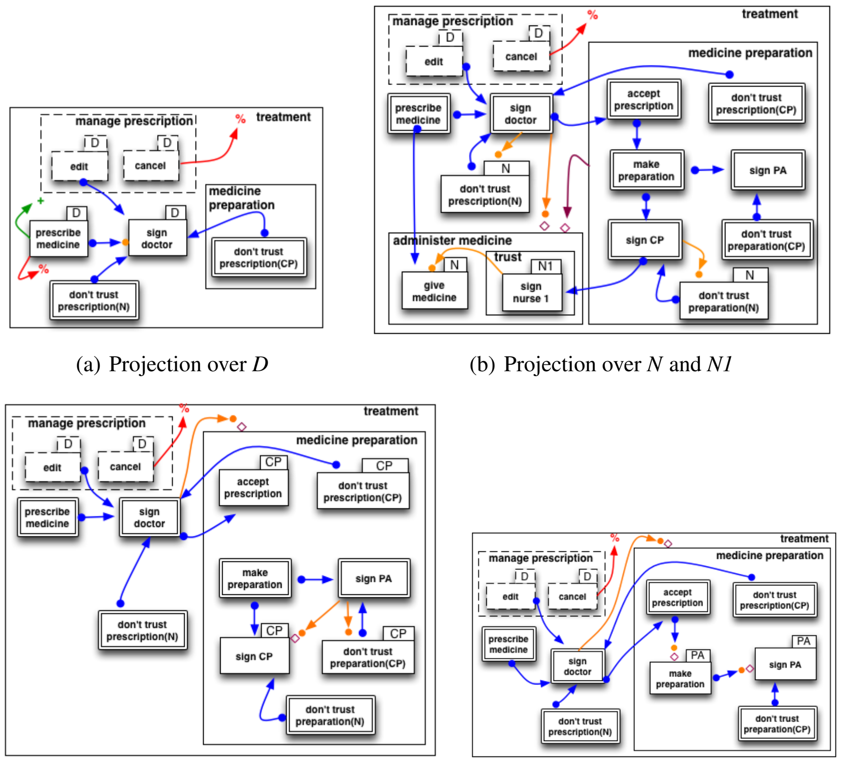
We define a run $a_0, a_1, \ldots$ *of a nested DCR Graph G to be a sequence of labels of a sequence of transitions* $G_i \xrightarrow{(e_i,a_i)} G_{i+1}$*. starting from* $G_0 = G$*. Assuming the marking of* G_i *is* $(\mathsf{Ex}_i, \mathsf{Re}_i, \mathsf{In}_i)$*, define a run to be accepting if for the underlying sequence of transitions it holds that* $\forall i \geq 0, e \in \mathsf{Re}_i.\exists j \geq i.(e = e_j \vee e \notin \mathsf{In}_j)$*. In words, a run is accepting if every required response event happens at some later stage or become excluded.*

4 Projections and Distributed Execution

In Sec. 4.1 below we define the notion of projection of a nested DCR Graphs, restricting the graph to a subset of the events, and in Sec. 4.2 we define the technique for distributing a nested DCR Graph as a set of local nested DCR Graphs obtained as projections and communicating by notifications of event executions.

4.1 Projections

A nested DCR Graph G is projected with respect to a *projection parameter* $\delta = (\delta_\mathsf{E}, \delta_\mathsf{L})$, where $\delta_\mathsf{E} \subseteq \mathsf{E}$ is a subset of the events of G satisfying that $\rhd(\delta_\mathsf{E}) \subseteq \delta_\mathsf{E}$, i.e. the subset is closed under the super event relation, and $\delta_\mathsf{L} \subseteq \mathsf{L}$ is a subset of the labels. The intuition is that the graph is restricted to only those events and relations that are relevant for the execution of events in δ_E and the labeling is restricted to the set δ_L. The technical difficulty is to infer the events and relations not in δ_E, referred to as *external events* below, that should be included in the projection because they influence the execution of the workflow restricted to the events in δ_E.



(a) Projection over *D*

(b) Projection over *N* and *N1*

(c) Projection over *CP*

(d) Projection over *PA*

Fig. 2. Oncology workflow projected with respect to different roles

Fig. 2 shows examples of projections of the oncology workflow with respect to different roles. For instance, Fig. 2(a) shows the projection with respect to the projection parameter $(\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$ where δ_{E}={manage prescription, edit, cancel, prescribe medicine, sign doctor} and δ_{L}={(edit, D), (cancel, D), (prescribe medicine, D), (sign doctor, D)}. The two events don't trust prescription (N) and don't trust prescription (CP) shown with double line borders are external events included in the projected graph even though they don't appear in the projection parameter. It is interesting to note that the doctor only needs to be aware of these two activities carried out by other participants. In comparison, the projection over the roles for nurses (*N* and *N1*) contains all the events since they may influence (because of the milestone relations) the execution of the events with roles *N* and *N1*. In other words, the doctors can carry out workflows highly independent of the other activities while the nurses are dependent on any event carried out by the other roles.

The formal definition of projection for nested DCR Graphs is given in 7 below. It generalizes the definition of projection introduced in [21] for DCR Graphs to support nesting and milestones.

Definition 7. *If* $G = (\mathsf{E}, \triangleright, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *then* $G_{|\delta} = (\mathsf{E}_{|\delta}, \triangleright_{|\delta}, \mathsf{M}_{|\delta}, \rightarrow\!\bullet_{|\delta}$ $, \bullet\!\!\rightarrow_{|\delta}, \rightarrow\!\diamond_{|\delta}, \rightarrow\!+_{|\delta}, \rightarrow\!\%_{|\delta}, \delta_{\mathsf{L}}, l_{|\delta})$ *is the projection of* G *with respect to* $\delta \subseteq \mathsf{E}$ *where:*

(i) $\mathsf{E}_{|\delta} = \rightarrow \delta_{\mathsf{E}}$, *for* $\rightarrow = \bigcup_{c \in C} c$, *and* $C = \{\mathsf{id}, \rightarrow\!\bullet^{\flat}, \bullet\!\!\rightarrow^{\flat}, \rightarrow\!\diamond^{\flat}, \rightarrow\!+^{\flat}, \rightarrow\!\%^{\flat}, \bullet\!\!\rightarrow^{\flat}\rightarrow\!\diamond^{\flat},$ $\rightarrow\!+^{\flat}\rightarrow\!\bullet^{\flat}, \rightarrow\!\%^{\flat}\rightarrow\!\bullet^{\flat}, \rightarrow\!+^{\flat}\rightarrow\!\diamond^{\flat}, \rightarrow\!\%^{\flat}\rightarrow\!\diamond^{\flat}\ \}$

(ii) $\triangleright_{|\delta}(e) = \triangleright(e)$, *if* $e \in \mathsf{E}_{|\delta}$

(iii) $l_{|\delta}(e) = \begin{cases} l(e) \cap \delta_{\mathsf{L}} & \textit{if } e \in \delta_{\mathsf{E}} \\ \emptyset & \textit{if } e \in \mathsf{E}_{|\delta} \backslash \delta_{\mathsf{E}} \end{cases}$

(iv) $\mathsf{M}_{|\delta} = (\mathsf{Ex}_{|\delta}, \mathsf{Re}_{|\delta}, \mathsf{In}_{|\delta})$ *where:*
   (a) $\mathsf{Ex}_{|\delta} = \mathsf{Ex} \cap \mathsf{E}_{|\delta}$
   (b) $\mathsf{Re}_{|\delta} = \mathsf{Re} \cap (\delta_{\mathsf{E}} \cup \rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}})$
   (c) $\mathsf{In}_{|\delta} = \mathsf{In} \cap \mathsf{E}_{|\delta}$

(v) $\rightarrow\!\bullet_{|\delta} = \rightarrow\!\bullet \cap ((\rightarrow\!\bullet^{\flat} \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$

(vi) $\bullet\!\!\rightarrow_{|\delta} = \bullet\!\!\rightarrow \cap ((\bullet\!\!\rightarrow^{\flat}\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}}) \times (\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}})) \cup ((\bullet\!\!\rightarrow^{\flat} \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}}))$

(vii) $\rightarrow\!\diamond_{|\delta} = \rightarrow\!\diamond \cap ((\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}})$

(viii) $\rightarrow\!+_{|\delta} = \rightarrow\!+ \cap \Big(((\rightarrow\!+^{\flat} \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}} \big) \cup \big((\rightarrow\!+^{\flat}\rightarrow\!\bullet^{\flat} \delta_{\mathsf{E}}) \times (\rightarrow\!\bullet^{\flat} \delta_{\mathsf{E}} \big) \cup \big((\rightarrow\!+^{\flat}\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}}) \times$ $(\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}} \big) \Big)$

(ix) $\rightarrow\!\%_{|\delta} = \rightarrow\!\% \cap \Big(((\rightarrow\!\%^{\flat} \delta_{\mathsf{E}}) \times \delta_{\mathsf{E}} \big) \cup \big((\rightarrow\!\%^{\flat}\rightarrow\!\bullet^{\flat} \delta_{\mathsf{E}}) \times (\rightarrow\!\bullet^{\flat} \delta_{\mathsf{E}} \big) \cup \big((\rightarrow\!\%^{\flat}\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}}) \times$ $(\rightarrow\!\diamond^{\flat} \delta_{\mathsf{E}} \big) \Big)$

(i) defines the set of events in the projection as all events that has a relation pointing to an event in the set δ_{E}, where the relation is either the identity relation (i.e. it is an event in δ_{E}), one of the core relations (flattened) or the relations such as $\bullet\!\!\rightarrow^{\flat}\rightarrow\!\diamond^{\flat}$ which includes all events that triggers as a response some event that is a milestone to an event in δ_{E} or the relations that include/exclude conditions and milestones to an event in the set δ_{E}.

Events in $\mathsf{E}_{|\delta} \backslash \delta_{\mathsf{E}}$ are referred to as external events and will be included in the projection without labels, as can be seen from the definition of the labeling function in *(iii)*. As we will formalize below, events without labels can not be executed by a user locally. However, when composed in a network containing other processes that can execute these events, their execution will be communicated to the process.

(iv) defines the projection of the marking: The executed and included event sets are simply restricted to the events in $\mathsf{E}_{|\delta}$. The responses are restricted to events in δ_{E} and events that have a milestone relation to an event in δ_{E} because these are the only responses that will affect the local execution of the projected graph. Note that these events will by definition be events in $\mathsf{E}_{|\delta}$ but may be external events.

Finally, *(v)* - *(ix)* state which relations should be included in the projection. For the events in δ_{E} all incoming relations should be included. Additionally response relations to events that are a milestone for an event in δ_{E} are included as well.

To define networks of communicating nested DCR Graphs and their semantics we use the following extension of a nested DCR Graph adding a new label to every event.

Definition 8. *For an DCR Graph $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ define $G^{\varepsilon} = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L} \cup \{\varepsilon\}, l^{\varepsilon})$, where $l^{\varepsilon} = l(e) \cup \{\varepsilon\}$ (assuming that $\varepsilon \notin \mathsf{L}$).*

We are now ready to state the key correspondence between global execution of events and the local execution of events in a projection.

Proposition 1. *Let $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ be a nested DCR Graph and $G_{|\delta}$ its projection with respect to a projection parameter $\delta = (\delta_{\mathsf{E}}, \delta_{\mathsf{L}})$. Then*

1. *for $e \in \delta_{\mathsf{E}}$ and $a \in \delta_{\mathsf{L}}$ it holds that $G \xrightarrow{(e,a)} G'$ if and only if $G_{|\delta} \xrightarrow{(e,a)} G'_{|\delta}$,*
2. *for $e \notin \mathsf{E}_{|\delta}$ it holds that $G \xrightarrow{(e,a)} G'$ implies $G_{|\delta} = G'_{|\delta}$,*
3. *for $e \in \mathsf{E}_{|\delta}$ it holds that $G \xrightarrow{(e,a)} G'$ implies $(G_{|\delta})^{\varepsilon} \xrightarrow{(e,\varepsilon)} (G'_{|\delta})^{\varepsilon}$,*

4.2 Distributed Execution

We are now ready to define networks of nested DCR Graphs and give the main technical theorem of the paper stating that a network of nested DCR Graphs obtained by projecting a nested DCR Graph G with respect to a covering vector of projection parameters has the same behavior as the original graph G.

Intuitively, a vector of projection parameters is covering if every event is included in at least one projection parameter and every label that is assigned to an event occurs at least once together with that event.

Definition 9. *We call a vector $\Delta = (\delta_1, \ldots, \delta_k)$ of projection parameters* covering *for some DCR Graph $G = (\mathsf{E}, \rhd, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ if:*

1. $\bigcup_{i \in [k]} \delta_{\mathsf{E}i} = \mathsf{E}$ *and*
2. $(\forall a \in \mathsf{L}. \forall e \in \mathsf{E}. a \in l(e) \Rightarrow (\exists i \in [k]. e \in \delta_{\mathsf{E}i} \wedge a \in \delta_{\mathsf{L}i})$

Definition 10. *We define a* network of DCR Graphs *N by the grammar*

$$N := G \mid N \| N$$

and let $\mathcal{N}_{\mathsf{E} \times \mathsf{L}}$ be the set of all networks with events in E and labels in L.

We write $\Pi_{i \in [n]} G_i$ for $G_1 \| G_2 \| \ldots \| G_n$. We define the set of events of a network of graphs inductively by $\mathcal{E}(G) = \mathsf{E}$ and $\mathcal{E}(N_1 \| N_2) = \mathcal{E}(N_1) \cup \mathcal{E}(N_2)$. Similarly, we define the set of labels of a network of graphs inductively by $\mathcal{L}(G) = \mathsf{L}$ and $\mathcal{L}(N_1 \| N_2) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2)$.

Definition 11. *The transition semantics of networks of DCR Graphs are given by the following inference rules:*

input $$\frac{G_1^\varepsilon \xrightarrow{(e,\varepsilon)} G_2^\varepsilon}{G_1 \overset{\triangleright e}{\rightsquigarrow} G_2}$$

sync input $$\frac{N_1 \overset{\triangleright e}{\rightsquigarrow} N_1' \quad N_2 \overset{\triangleright e}{\rightsquigarrow} N_2'}{N_1 \| N_2 \overset{\triangleright e}{\rightsquigarrow} N_1' \| N_2'}$$

local input $$\frac{N_i \overset{\triangleright e}{\rightsquigarrow} N_i' \quad e \notin \mathcal{E}(N_{1-i})}{N_0 \| N_1 \overset{\triangleright e}{\rightsquigarrow} N_0' \| N_1} \quad N_{1-i} = N_{1-i}', i \in \{0,1\}$$

sync step $$\frac{N_i \xrightarrow{(e,a)} N_i' \quad N_{1-i} \overset{\triangleright e}{\rightsquigarrow} N_{1-i}'}{N_0 \| N_1 \xrightarrow{(e,a)} N_0' \| N_1'} \ i \in \{0,1\}$$

local step $$\frac{N_i \xrightarrow{(e,a)} N_i' \quad e \notin \mathcal{E}(N_{i-1})}{N_0 \| N_1 \xrightarrow{(e,a)} N_0' \| N_1} \quad N_{1-i} = N_{1-i}', i \in \{0,1\}$$

For a network of nested DCR Graphs N we define the corresponding transition system $TS(N)$ by $(\mathcal{N}_{\mathcal{EL}(N)}, N, \mathcal{EL}(N), \rightarrow \subseteq \mathcal{N}_{\mathcal{EL}(N)} \times \mathcal{EL}(N) \times \mathcal{N}_{\mathcal{EL}(N)})$ where $\mathcal{EL}(N) = \mathcal{E}(N) \times \mathcal{L}(N)$ and the transition relation $\rightarrow \subseteq \mathcal{N}_{\mathcal{EL}(N)} \times \mathcal{EL}(N) \times \mathcal{N}_{\mathcal{EL}(N)}$ is defined by the inference rules above.

We define a run $a_0, a_1, \ldots$ of the transition system to be a sequence of labels of a sequence of transitions $N_i \xrightarrow{(e_i,a_i)} N_{i+1}$ starting from the initial network. We define a run for a network $N = \Pi_{i\in[n]} G_i$ to be accepting if for the underlying sequence of transitions it holds that $\forall j \in [n], \forall i \geq 0, e \in \mathsf{Re}_{j,i}.\exists k \geq i.(e = e_k \vee e \notin \mathsf{In}_k)$, where $\mathsf{Re}_{j,i}$ is the set of required responses in the jth nested DCR Graph in the network in the ith step of the run. In words, a run is accepting if every response event in a local nested DCR Graph in the network happens at some later state or become excluded.

Thm. 1 below now states the correspondence between a nested DCR Graph and the network of nested DCR Graphs obtained from a covering projection.

Theorem 1. *For a nested DCR Graph G and a covering vector of projection parameters $\Delta = (\delta_1, \ldots, \delta_n)$ it holds that $TS(G)$ is bisimilar to $TS(G_\Delta)$, where $G_\Delta = \Pi_{i\in[n]} G_{|\delta_i}$. Moreover, a run is accepting in $TS(G)$ if and only if the bisimilar run is accepting in $TS(G_\Delta)$.*

The generality of the distribution technique given above allows for fine tuned projections where we select only a few events for a specific role and actor, but in most cases the parameter is likely to be chosen so that the projected graph shows the full responsibilities of a specific role or actor. A set of nested DCR Graphs can be maintained and executed in a distributed fashion, meaning that there is a separate implementation for every graph and that the execution of shared events is communicated between them. Through the distributed execution of projected graphs, nested DCR Graphs can be used as a (declarative) choreography model to the line of work (on typed imperative process models) in [6]: The original graph can be seen as the choreography, describing how the system as a whole should function, from which we project multiple end-points for individual roles or actors that can be implemented independently.

5 Conclusion and Future Work

We have presented a formal technique for safe distribution of collaborative, cross-organizational workflows modeled declaratively in the model of Nested Dynamic Condition Response (Nested DCR) Graphs [17,20]. The key difference between the present work and the related work surveyed in Sec. 1.1 is that Nested DCR Graphs is a declarative model while most previous work has focussed on imperative models. We have argued for the use of a declarative approach for flexible workflows of knowledge workers and exemplified the techniques on a small workflow identified during a previous field study at danish hospitals [26]. The example is not supposed to demonstrate completeness of the technique but to capture two common examples of flexibility, namely the need to reconsider previous activities and the flexibility to distribute the execution of a workflow across different units within or across organizations.

The distribution technique presented is an extension of a method developed recently for flat DCR Graphs [21] to allow for nesting of events and the co-called milestone relations. This again allows us to apply the technique to the oncology workflow which we believe is an important new contribution in order to communicate the ideas better to people working within the healthcare domain.

A number of interesting questions are left for future work. We have implemented a prototype tool for design, simulation and verification (model checking via the SPIN and ZING model checkers) of DCR Graphs as reported on in [19]. These tools should be extended to nested graphs and the distribution technique should be implemented. This then leads to considering what can be achieved by performing verification of local components individually. We also aim to investigate how to support dynamic changes to local components, using the underlying idea of the distribution technique to determine what should be changed in other components when a local component is changed. Finally we are working on extending the model to allow for data and time to be represented and developing a prototype implementation integrated with the work on pervasive user interfaces carried out in the other track of the TrustCare project. This would allow us to carry out a larger demonstration project jointly with a hospital evaluating both the workflow modeling and the pervasive user interfaces. Along the same lines, it would be interesting to relate our work to the approach in the OpenKnowledge and Safe & Sound projects based on the Lightweight Coordination Calculus (LCC) [37].

Acknowledgements: We are grateful to the anonymous reviewers for valuable feedback.

References

1. Wil M. P. van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, CAiSE '01, pages 140–156, 2001.

2. Jakob E. Bardram, Jonathan Bunde-Pedersen, Afsaneh Doryab, and Steffen Sorensen. Clinical surfaces: Activity-based computing support for distributed multi-display environments inside hospitals. In *INTERACT 2009: 12th IFIP TC13 Conference in Human-Computer Interaction*, Uppsala, Sweden, 2009.
3. Jakob E. Bardram, Jonathan Bunde-Pedersen, and Mads Soegaard. Support for activity-based computing in a personal computing operating system. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 211–220, New York, NY, USA, 2006. ACM Press.
4. Mario Bravetti and Gianluigi Zavattaro. Contract based multi-party service composition. In *International Symposium on Fundamentals of Software Engineering (FSEN)*, volume 4767, pages 207–222. Springer, 2007.
5. Mario Bravetti and Gianluigi Zavattaro. A theory of contracts for strong service compliance. *Mathematical. Structures in Comp. Sci.*, 19:601–638, June 2009.
6. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *16th European Symposium on Programming (ESOP'07)*, LNCS, pages 2–17. Springer, 2007.
7. Ilaria Castellani, Madhavan Mukund, and P. Thiagarajan. Synthesizing distributed transition systems from global specifications. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1738, pages 219–231. Springer Berlin / Heidelberg, 1999.
8. Federico Chesani, Pietro De Matteis, Paola Mello, Marco Montali, and Sergio Storari. A framework for defining and verifying clinical guidelines: A case study on cancer screening. In Floriana Esposito, Zbigniew Ras, Donato Malerba, and Giovanni Semeraro, editors, *Foundations of Intelligent Systems*, volume 4203 of *Lecture Notes in Computer Science*, pages 338–343. Springer Berlin / Heidelberg, 2006.
9. Federico Chesani, E. Lamma, P. Mello, M. Montali, S. Storari, P. Baldazzi, and M. Manfredi. *Compliance Checking of Cancer-Screening Careflows: an Approach Based on Computational Logic*, chapter Computer- Based Medical Guidelines and Protocols: a Primer and Current Trends. IOS Press, 2008.
10. Federico Chesani, Paola Mello, Marco Montali, and Sergio Storari. Testing careflow process execution conformance by translating a graphical language to computational logic. In Riccardo Bellazzi, Ameen Abu-Hanna, and Jim Hunter, editors, *Artificial Intelligence in Medicine*, volume 4594 of *Lecture Notes in Computer Science*, pages 479–488. Springer Berlin / Heidelberg, 2007.
11. Dirk Fahland. Towards analyzing declarative workflows. In *Autonomous and Adaptive Web Services*, 2007.
12. Walid Fdhila and Claude Godart. Toward synchronization between decentralized orchestrations of composite web services. In *CollaborateCom'09*, pages 1–10, 2009.
13. Walid Fdhila, Ustun Yildiz, and Claude Godart. A flexible approach for automatic process decentralization using dependency tables. *International Conference on Web Services*, 2009.
14. Xiang Fu, Tevfik Bultan, and Jianwen Su. Realizability of conversation protocols with message contents. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 96–, Washington, DC, USA, 2004. IEEE Computer Society.
15. Keijo Heljanko and Alin Stefanescu. Complexity results for checking distributed implementability. In *Proceedings of the Fifth International Conference on Application of Concurrency to System Design*, pages 78–87, 2005.
16. Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. `http://www.trustcare.dk/`.
17. Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Post proceedings of International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 10)*, 2011.

18. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Declarative modelling and safe distribution of healthcare workflows. In *International Symposium on Foundations of Health Information Engineering and Systems*, Johannesburg, South Africa, August 2011.
19. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proceedings of IEEE International EDOC Conference*, 2011. to appear.
20. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. to appear.
21. Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Safe distribution of declarative processes. In *9th International Conference on Software Engineering and Formal Methods (SEFM) 2011*, 2011. to appear.
22. International symposium on foundations of health information engineering and systems, August 2011.
23. Ali Rahmanzadeh John Fox, Nicky Johns. Disseminating medical knowledge: the proforma approach. *Artificial intelligence in medicine*, 14:157–182, September 1998.
24. R. Khalaf and F. Leymann. Role-based decomposition of business processes using BPEL. In *Web Services, 2006. ICWS '06. International Conference on*, pages 770 –780, sept. 2006.
25. Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 235–253, London, UK, 2000. Springer-Verlag.
26. Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Proceedings of 2nd International Workshop on Process-oriented information systems in healthcare (ProHealth 08)*, pages 36–43, Milan, Italy, 2008. BPM 2008 Workshops.
27. Axel Martens. Analyzing web service based business processes. In *Fundamental Approaches to Software Engineering*. Springer Berlin / Heidelberg, 2005.
28. Zoran Milosevic, Shazia Sadiq, and Maria Orlowska. Towards a methodology for deriving contract-compliant business processes. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 395–400. Springer Berlin / Heidelberg, 2006.
29. Saayan Mitra, Ratnesh Kumar, and Samik Basu. Optimum decentralized choreography for web services composition. In *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*, 2008.
30. Marco Montali. *Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.
31. Raghava Rao Mukkamala, Thomas Hildebrandt, and Janus Boris Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008.
32. M. Mukund. From global specifications to distributed implementations. In *Synthesis and Control of Discrete Event Systems*. Springer, 2002.
33. Nataliya Mulyar, Maja Pesic, Wil M. P. Van Der Aalst, and Mor Peleg. Declarative and procedural approaches for modelling clinical guidelines: addressing flexibility issues. In *Proceedings of the 2007 international conference on Business process management*, BPM'07, pages 335–346, Berlin, Heidelberg, 2008. Springer-Verlag.
34. Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. *SIGPLAN Not.*, 39:170–187, October 2004.
35. Object Management Group BPMN Technical Committee. Business Process Model and Notation, version 2.0. Webpage, january 2011. `http://www.omg.org/spec/BPMN/2.0/PDF`.

36. Stefanie Rinderle, Andreas Wombacher, and Manfred Reichert. Evolution of process choreographies in dychor. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *LNCS*, pages 273–290. Springer, 2006.
37. David Robertson. A lightweight coordination calculus for agent systems. In *In Declarative Agent Languages and Technologies*, pages 183–197, 2004.
38. W. Sadiq, S. Sadiq, and K. Schulz. Model driven distribution of collaborative business processes. In *Services Computing, 2006. SCC '06. IEEE International Conference on*, pages 281 –284, sept. 2006.
39. Arthur ter Hofstede, Rob van Glabbeek, and David Stork. Query nets: Interacting workflow modules that ensure global termination. In *Business Process Management*. Springer Berlin / Heidelberg, 2003.
40. P Terenziani, S Montani, A Bottrighi, M Torchio, G Molino, and G. Correndo. The glare approach to clinical guideline: Main features. *Symposium on Computerized Guidelines and Protocols*, 101:62–6, April 2004.
41. W. M. P. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
42. Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. *The Computer Journal*, 53(1):90–106, January 2010.
43. Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
44. Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006.
45. Wil M.P van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In M. Bravetti, M. Nunez, and Gianluigi Zavattaro, editors, *Proceedings of Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *LNCS*, pages 1–23. Springer Verlag, 2006.
46. W.M.P. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Information Technology and Management*, 4:345–389, 2003.
47. Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz (Eds.) Rozenberg, editors, *Advances in Petri Nets*, volume Vol. 255 of *Lecture Notes in Computer Science,*, pages 325–392. Springer, 1986.
48. Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In *Proceedings of the 6th International Conference on Database Theory*, pages 230–246, London, UK, 1997. Springer-Verlag.
49. X. Yi and K.J Kochut. Process composition of web services with complex conversation protocols. In *Design, Analysis, and Simulation of Distributed Systems Symposium at Adavanced Simulation Technology*, 2004.
50. W. Zielonka. Notes on finite asynchronous automata. *Informatique Thorique et Applications*, 21(2):99–135, 1987.

Live Sessions with Responses

Marco Carbone
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
carbonem@itu.dk

Thomas Hildebrandt
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
hilde@itu.dk

Tijs Slaats
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark

Exformatics A/S
Lautrupsgade 13
2100 Copenhagen, Denmark
tslaats@itu.dk

Abstract

We discuss ongoing work on *sessions with responses* as an extension of standard session types allowing to specify liveness properties. Concretely, sessions with responses are specified by annotating branching and selection labels with a finite conjunction of disjunctive responses. A disjunctive response is a finite disjunction of labels, and the intended meaning is that whenever a label is selected, one of the labels in each of the disjunctive responses is eventually selected. We illustrate the use of live sessions by a buyer/seller example where the buyer and seller engage in a potential infinite number of deals, and for each deal negotiates about the price but must eventually either agree or stop the negotiation. We informally describe a typing system for guaranteeing that well typed processes are live in this sense and discuss ongoing and future work.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords Session types, liveness

1. Introducing Live Sessions

Session types first appeared in [7] as types for abstracting communication patterns within a session. As a benefit, besides facilitating the design of distributed protocols, they can be used for guaranteeing in-session properties such as *safety* (lack of communication errors) and *progress* (a session never gets stuck).

Several other works following [7] propose different extensions of session types, including extensions to multiparty sessions [8], exceptions [4], Object-Oriented Programming [5], and many more.

However, the current state of the art focuses on proving that well-typed systems enjoy, at least, progress in the above meaning that a session never gets stuck. In many places, the term progress is used more generally in the meaning of *liveness*, i.e. that something *good* will eventually happen, and not just *something* will eventually happen.

A fundamental and very common form of liveness property is the *request-response* property: "Whenever some event a occurs, some event b will eventually occur in the future" [2]. The request-response property may be specified by the LTL formula $\mathbf{G}(a \implies \mathbf{F}b)$, where $\mathbf{G}$ is read as *generally*, i.e. in all future steps, and $\mathbf{F}$ reads *future*, i.e. eventually in some future step.

In the present short paper, we consider the possibility of enhancing session types to capture a generalization of the request-response property that we will refer to as the *disjunctive* response property: "Whenever some event a occurs, one event out of a given finite set of response events $\{b_1, \ldots, b_n\}$ will eventually occur in the future". We will allow several such disjunctive responses for the same event a, which will then correspond to formulas $\mathbf{G}(a \implies \mathbf{F}\rho)$, where $\rho = \delta_0 \wedge \delta_1 \wedge \ldots \wedge \delta_{n-1}$ for $\delta_i = b_1 \vee \ldots \vee b_{m_i}$.

Instead of using the LTL notation, we will use the shorter notation $a \bullet\!\!\rightarrow \rho$ as also used in e.g. [6, 9, 10].

Before embarking, we want to remark that a liveness property is sometimes defined as a property that can not be violated in finitely many steps [1, 2]. This makes perfect sense for non-terminating processes or if termination can not be detected. However, as sessions may indeed terminate, we intend also to identify violations in finite executions. For instance, the request-response property can be violated if the process terminates with an "open" request, i.e. a request that has not yet been responded to.

We illustrate our idea by giving an informal motivating example of a session type that we extend with annotations expressing the request-response property.

First, recall that a session type is a sequence of i/o actions specifying the expected communication done by one of the parties engaged in a session. For example, consider a buyer-seller scenario where a buyer and seller engage in a potential infinite number of deals, and for each deal negotiates about the price. In a negotiation, the buyer can either give an offer or stop the negotiation. If the buyer gives an offer, the seller can either ask for more or say it is ok. In a such a scenario, the seller's behaviour could be specified by the following session type:

$$\mu t.\ \&\left\{\begin{array}{l} \texttt{offer} : ?(\mathsf{int}).\ \oplus\{\,\texttt{more} : t, \texttt{ok} : t\},\\ \texttt{stop} : t \end{array}\right\}$$

The type above describes a non terminating session, where the seller is offering the buyer two options, namely `offer` and `stop`. If the first option is selected by the buyer then the seller expects to receive an integer and then selects either `more` or `ok`. Anyhow, after any option is chosen, such a behaviour is reiterated again.

The protocol (session) described above could be implemented in several ways. For instance,

$$\mu X.\ k\&\left\{\begin{array}{ll} \texttt{offer} : & k?(x).\ \mathsf{if}\ (\mathsf{notEnough}(x))\ \mathsf{then}\\ & k \oplus \texttt{more}.\ \mathsf{X}\ \mathsf{else}\ k \oplus \texttt{ok}.\ \mathsf{X}\\ \texttt{stop} : & \mathsf{X} \end{array}\right\}$$

gives an implementation where, for some values of x, the seller could decide to infinitely ask for more, ending up in a somehow unwanted computation. On the other hand, the following process

$$\mu X.\ k\&\left\{\begin{array}{l} \texttt{offer} :\ k?(x).\ k \oplus \texttt{ok}.\ \mathsf{X}\\ \texttt{stop} :\ \mathsf{X} \end{array}\right\}$$

still implements the session described by our type, but satisfying the liveness property that if an offer is made, this deal is eventually ended.

The example illustrates, that it would be relevant to constrain the session protocol further to require request-response liveness properties, where the usual in-session selection of branches can be used in the specification as request and response events.

We suggest that such properties could be intuitively expressed by allowing branching and selection labels to be annotated with a (conjunction of) disjunctive response requirements.

For instance, if we wanted to require that the ok or stop branch must eventually be selected if an offer is made, we could add the following annotation to the type above.

$$\mu\mathtt{t}.\ \&\left\{ \begin{array}{l} \mathtt{offer}[\mathtt{ok} \vee \mathtt{stop}] :?(\mathtt{int}).\ \oplus \{ \mathtt{more} : t, \mathtt{ok} : t\}, \\ \mathtt{stop} : t \end{array} \right\}$$

The first process above should then *not* be well-typed, while the second should pass the type check.

2. Session Calculus and the Liveness Property

As session calculus we may use a variant of the π-calculus with sessions [7, 11] which outlaws restriction on public channels, and allows replicated behaviour only for services. This choice is not crucial for the definition of live-sessions.

Semantics As semantics we use the standard reduction semantics $\rightarrow$ [7] except selection reduction steps are annotated with the selected label. This allows us to define the trace language for a process and the liveness properties our type system will enforce. Let $\mathsf{Tr}(\mathsf{P})$ refer to the set of finite and infinite traces of labels of the process P.

Liveness We can now define the liveness property proposed in our introduction.

DEFINITION 1 (Liveness). *Assume a disjunctive response $\rho = \delta_0 \wedge \delta_1 \wedge \ldots \wedge \delta_{n-1}$ for $\delta_i = l_{i,1} \vee \ldots \vee l_{i,m_i}$. A trace $\sigma \in \mathsf{Tr}(\mathsf{P})$ of a process P then satisfies the request-response liveness property $l \bullet\!\!\rightarrow \rho$ if for all $k < |\sigma|$, if $\sigma_k = l$ then for all $i \in \{0, \ldots, n-1\}$ there exists $j \in \{1, \ldots, m_i\}$ such that there exists $k' > k$ for which $\sigma_{k'} = l_{i,j}$.*

We say that a process P has the liveness property wrt a set of response liveness properties $\mathcal{P}$ whenever each trace σ of P satisfies every property in $\mathcal{P}$.

The intuitive meaning of the liveness property is as stated in the previous section, that a trace satisfy the property $l \bullet\!\!\rightarrow \delta$ for $\delta = l_1 \vee \ldots l_m$ if any occurrence of the label l is eventually followed by a label l_i for some $i \in \{0, \ldots, m\}$, and in general, a trace satisfies the property $l \bullet\!\!\rightarrow \rho$ for $\rho = \delta_0 \wedge \ldots \delta_{n-1}$ if it satisfies $l \bullet\!\!\rightarrow \delta_j$ for all $j \in \{0, \ldots, n-1\}$.

3. Live Session Typing

Session Types with Responses. The generalization of session types to session types with responses is given by the following grammar:

$$\begin{array}{lll} \alpha & ::= & ?(\theta).\ \alpha \mid !(\theta).\ \alpha \mid \&\{l_i[\rho_i] : \alpha_i\} \mid \oplus\{l_i[\rho_i] : \alpha_i\} \mid \\ & & \mathtt{end} \mid \mu\mathtt{t}.\ \alpha \mid \mathtt{t} \\ \theta & ::= & S \mid \alpha \qquad\qquad S ::= \mathsf{basic} \mid \langle\alpha\rangle \\ \rho & ::= & \top \mid \delta \mid \rho \wedge \rho \qquad\quad \delta ::= l \mid \delta \vee \delta \end{array}$$

Here, $?(\theta).\ \alpha$ and $!(\theta).\ \alpha$ denote in-session input and output followed by the communications in α. The type θ abstracts what is communicated: a basic value (basic denotes basic types, e.g., $\mathtt{int}$ or bool), a service channel of type $\langle\alpha\rangle$, or a session channel of type α. Finally, $\&\{l_i[\rho_i] : \alpha_i\}$ and $\oplus\{l_i[\rho_i] : \alpha_i\}$ denote branching and selection types, and $\mathtt{end}$ is the inactive session. Branching and selection have been enhanced with disjunctive responses (ρ) as introduced in the previous section. The response $\top$ (true) represents the empty response, and thus we will usually write l for $l[\top]$.

4. Conclusions, Ongoing and Future Work

Above we have suggested extending session types to allow expressing *liveness* properties. As a starting point, we considered liveness properties stated as a generalisation of the fundamental request-response pattern. We proposed that these properties could be expressed in session types by annotating labels in selections and branchings with *disjunctive responses*.

This is ongoing work. We believe we have a sound and complete set of live session type rules conservatively generalising standard session types. Soundness means in this setting, that every well-typed process satisfy the liveness property expressed by the response annotations as defined in Def. 1 and is well-typed according to the underlying (standard) session type. Completeness means that each process which is well-typed according to the underlying session type and satisfy the liveness property expressed as response annotations will be well-typed according to the live session type rules.The general idea of the typing rules are to keep track of accumulated response requirements for each session, and also for each loop within sessions, and check fulfilment of responses at recursion points and when sessions end.

Note that a consequence of the response annotations, for a given live session type there may be no well-typed processes if a required response is not allowed by the underlying session type, e.g. if none of the requested responses are possible future labels.

The response annotations relate to the recent work on transition systems with responses in [3] and also the work on DCR Graphs in [6, 9] and declarative process models in [10].

For future work we consider generalising the liveness properties to also allow session start as a request event, and session-end as a response event. This would e.g. allow expressing the general property that if a particular session is started, it must be ended. As liveness constraints are often verified under some fairness assumptions of the process, it would also be relevant to consider extensions where fairness, or even more general liveness assumptions, can also be expressed for the *processes*. For instance, a fair parallel, or fair branching operators could be introduced in the session calculus. This should then be taken into account when type checking against the live session types.

References

[1] B. Alpern and F. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1987.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] Marco Carbone, Thomas T. Hildebrandt, Gian Perrone, and Andrzej Wasowski. Refinement for transition systems with responses. In *FIT*, pages 48–55, 2012.

[4] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured interactional exceptions for session types. In *19th Int'l Conference on Concurrency Theory (Concur'08)*, LNCS, pages 402–417. Springer, 2008.

[5] Simon J. Gay, Vasco Thudichum Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *POPL*, pages 299–312, 2010.

[6] Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response

graphs. In Kohei Honda and Alan Mycroft, editors, *PLACES*, volume 69 of *EPTCS*, pages 59–73, 2010.

[7] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *7th European Symposium on Programming (ESOP'98)*, volume 1381 of *LNCS*, pages 22–138. Springer-Verlag, 1998.

[8] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *35th Symposium on Principles of Programming Languages (POPL'08)*, pages 273–284. ACM, 2008.

[9] Raghava Rao Mukkamala. *A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, March 2012.

[10] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 international conference on Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag.

[11] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An Interaction-based Language and its Typing System. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer-Verlag, 1994.

Type Checking Liveness for Collaborative Processes with Bounded and Unbounded Recursion

Søren Debois[1], Thomas Hildebrandt[1], Tijs Slaats[1,2], and Nobuko Yoshida[3]

[1] IT University of Copenhagen
{debois,hilde,tslaats}@itu.dk
[2] Exformatics A/S
[3] Imperial College
yoshida@doc.ic.ac.uk

Abstract. We present the first session typing system guaranteeing response liveness properties for possibly non-terminating communicating processes. The types augment the branch and select types of the standard binary session types for the π-calculus with a set of required responses, indicating that whenever a particular label is selected, a set of other labels, its responses, must eventually also be selected. We prove that these extended types are strictly more expressive than standard session types. We provide a type system for a process calculus similar to collaborative BPMN processes and prove that this type system is sound, i.e., it guarantees request-response liveness for dead-lock free processes. We exemplify the use of the calculus and type system on a concrete example of an infinite state system.

1 Introduction

Session types were originally introduced as typing systems for particular π-calculi, modelling the interleaved execution of some number of two-party protocols. Here, a well-typed process is guaranteed freedom from race-conditions and compatibility of communications, usually referred to as session fidelity [12,20,18]. Session types have subsequently been the subject of intense study, with much work on applications, typically to programming languages, e.g., [8,14,11,15]. A number of generalisations of the theory has been proposed, notably to multi-party session types [13]. Multi-party session types have a close resemblance to choreographies as found in standards for business process modelling languages such as Business Process Model and Notation (BPMN) [16] and Web-Services Choreography Description Language (WS-CDL) and has been argued in theory to be able to provide typed BPMN processes [5].

So far, behavioral types have focussed on *safety* properties, notably progress and lock-freedom [2,1,3,7,19]. The main contribution of the paper is the first extension of binary session types to allow specification of *liveness*—the property of a process eventually getting to "do something good".

Liveness properties have normally been dealt with by resorting to model-checking techniques (typically requiring a state-space exploration) or dependent types explicitly representing bounds of iterations. In the present paper we show that a common and fundamental class of liveness properties, so-called *request-response* properties, can be dealt with statically in the type rules, that is, without resorting to statespace exploration or the detail (and complexity) of dependent types. This means in particular, that we can deal with infinite state systems as we will exemplify below. Other benefits of the session type-checking approach are that it allows to specify properties as types that can be understood and used as interface specifications and for compositional reasoning.

We develop the theory in the setting of a process calculus with value passing, conditional branching, external branching and selection, and bounded and unbounded recursion. This calculus corresponds to a core subset of collaborative BPMN processes with structured loops and bounded iteration, in which both live and non-live infinite state space BPMN processes can be defined. As a running example we use the collaborative shopping cart process shown in Fig. A.

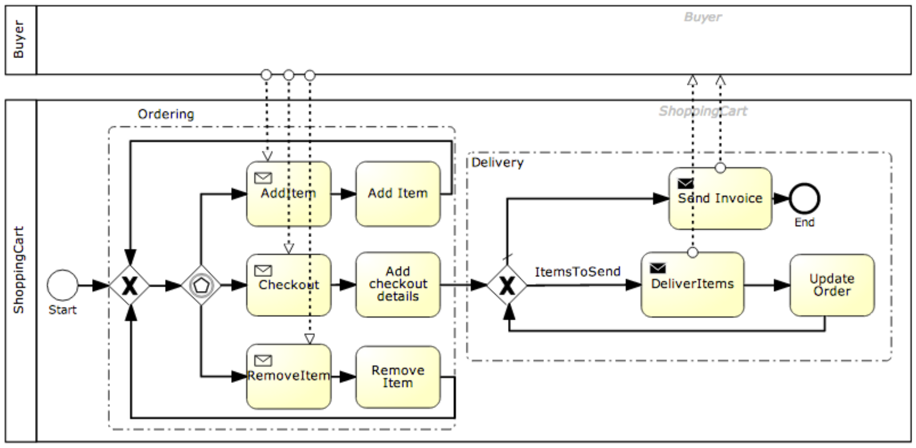


Fig. A. A Potentially Non-live Shopping Cart BPMN Process

The diagram contains two pools: The Buyer and the ShoppingCart. Only the latter contains a specified process, which consists of two parts: Ordering and Delivery. Ordering is a loop that starts with an event-based gateway, branching according to the message (AddItem, Checkout, RemoveItem) sent by the customer: AddItem and RemoveItem messages respectively provides an item to be added or removed from the order after which the loop repeats. A Checkout message provides delivery details of the order, exits the loop and proceeds to the Delivery part. Delivery is again a loop, delivering the ordered items and then sending the invoice.

A buyer who wants to communicate *safely* with the Shopping Cart, must follow the protocol described above, and in particular be able to receive an unbounded number of items before receiving the invoice. Writing $\mathsf{AI}, \mathsf{RI}, \mathsf{CO}, \mathsf{DI}$, and SI for the actions "Add Items", "Remove Items", "Checkout", "Deliver

Items" and "Send Invoice"; we can describe this protocol with a session type:

$$\mu t.\&\{\mathsf{AI}.?.t, \mathsf{RI}.?.t, \mathsf{CO}.?.\mu t'.\oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathtt{end}\}\}$$

This session type can be regarded as a *behavioral* interface, specifying that the process first expects to receive either an AI (AddItem), RI (RemoveItem) or a CO (CheckOut) event. The two first events must be followed by a message (indicated by ?), which in the implementation provides the item to be added or removed, after which the protocol returns to the initial state. The checkout event is followed by a message (again indicated by a ?) after which the protocol enters a new loop, either sending a DI (DeliverItem) event followed by a message (indicated by a !) and repeating, or sending an SI (SendInvoice) event followed by a message (the invoice) and ending.

However, standard session types can not specify the very relevant *liveness* property, that a CheckOut event is *eventually* followed by an invoice event. This is an example of a so-called *response* property: an action (the request) must be followed by a particular response. Thus, we conservatively extend binary session types to specify such response properties, and we show that this extension is strictly more expressive than standard session types. We do so by annotating the checkout selection in the type with the required response:

$$\mu t.\&\{\mathsf{AI}.?.t, \mathsf{RI}.?.t, \mathsf{CO}[\{\mathsf{SI}\}].?.\mu t'.\oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathtt{end}\}\}\ .$$

Intuitively: "if CO is selected, then subsequently also SI must be selected."

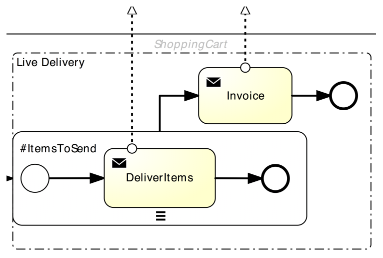


Fig. B. Live delivery with MI Sub-Process

It is not possible to determine from the flow graph alone, if this response property is guaranteed. The second loop may not terminate, e.g., if the Order is not updated properly when items are sent or the ItemsToSend condition is defined wrongly. However, we can remove this dependency on the data if we replace the delivery loop with a looping primitive which is guaranteed to be finite. In BPMN this can be achieved with e.g. the *Sequential Multiple Instance Sub-process*, which sequentially executes a finite number (determined by an expression computed at run-time) of instances of a sub-process. This allows us to re-define the Delivery as shown in Fig. B, yielding a re-defined Shopping Cart process, for which it can be inferred from the structure of the process that it satisfies the response property. In general, we want also to be able to check processes where responses are requested within (potentially) infinite loops. The type system we present gives such guarantees, essentially by collecting all requested responses in a forward analysis, exploiting that potentially infinite loops can guarantee a particular response only if every path through the loop can; and that order (request-response vs response-request) is in this case irrelevant. We prove that, if the system is lock free, then the typing system indeed guarantees that all requested responses are eventually fulfilled. Lock-freedom is needed because as is well known, collaborative processes with interleaved sessions may introduce dependency locks. Lock-freedom is well-studied for binary session types [2,1,3,7,19], or may alternatively be achieved by resorting to global types [13], the combination with which we leave for future work. In summary, our contributions are as follows.

- We extend binary session types with a notion of *required response*. This allows types to express that whenever a particular pattern of communication is chosen (a label selected in branch/select), that choice must subsequently be followed by a particular other choice, its *required response*.
- We prove that this extension induces a strictly more expressive language class than standard session types.
- We give a typing system conservatively extending standard binary session types which gives the further guarantee that a lock-free well-typed process will, in any execution, provide all its required responses.
- We exemplify the use of these extended types to guarantee both safety and liveness properties for a non-trivial, infinite state collaborative process, which exhibits both possibly infinite looping and bounded iteration.

Overview of this paper. In Sec. 2 we define our calculus and introduce a shopping cart process which will serve as a running example; in Sec. 3 we give an LTS-semantics for the calculus. In Sec. 4 extend binary session types to allow specification of response liveness properties, give a transition semantics for the types, and sketch a proof that these extended types induce a strictly larger class of languages than does standard binary session types. In Sec. 5 we define exactly how our extended session types induce a notion of liveness on processes. In Sec. 6 we give our extended typing rules for sessions with responses and state its subject reduction result. In Sec. 7 we prove that the extended typing rules guarantees liveness for lock-free processes. Finally, in Sec. 8 we conclude. For want of space, this paper omits details and proofs; for these, refer to [4].

2 Process Terms

Processes communicate only via named communication (session) channels by synchronizing send and receive actions or synchronizing select and branch events (as in standard session typed π-calculus). The session typing rules presented in the next session guarantees that there is always at most one active send and receive action for a given channel. To distinguish dual ends of communication channels, we employ *polarised names* [10,20]: If c is a channel name, c^+ and c^- are the dual ends of the channel c. We call these *polarised channel names*, with "+" and "-" polarities. If k is a polarised channel name, we write $\overline{k}$ for its dual, e.g., $\overline{c^+} = c^-$. In general c ranges or channel names; p over polarities $+, -$; k, h over polarised channel names; x over data variables; i over recursion variables (explained below); v over data values including numbers, strings and booleans;

e over data expressions; and finally X, Y over process variables.

$$\begin{aligned} P \ ::= & \ k!\langle e\rangle.P \ \mid \ k?(x).P \ \mid \ k!l.P \ \mid \ k?\{l_i.P_i\}_{i\in I} \ \mid \ \mathbf{0} \ \mid \ P|Q \\ & \mid \ \mathrm{rec}\, X.P \ \mid \ (\mathrm{rec}^e\, X(i).P; Q) \ \mid \ X[\tilde{k}] \ \mid \ \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \end{aligned}$$

The first four process constructors are prefixes taking part in a communication. These are standard for session typed π-calculi, except we allow only data and not channel names to be sent. The process $k!\langle e\rangle.P$ sends data v over channel k when $e \Downarrow v$, and proceeds as P. Dually, $k?(x).P$ receives a data value over channel k and substitutes it for the x binding in P. A *branch* process $k?\{l_i.P_i\}_{i\in I}$ offers a choice between labels l_i, proceeding to P_i if the i'th label is chosen. The process $\mathbf{0}$ is the standard inactive process (termination), and $P \mid Q$ is the parallel composition of processes P and Q.

Recursion comes in two forms: a general, potentially non-terminating recursion $\mathrm{rec}\, X.P$, where X binds in P; and a primitive recursion, guaranteed to terminate, with syntax $(\mathrm{rec}^e\, X(i).P; Q)$. The latter process, when $e \Downarrow n + 1$, executes $P\{n/i\}$ and repeats, and when $e \Downarrow 0$, evolves to Q. By convention in $(\mathrm{rec}^e\, X(i).P; Q)$ neither of $\mathbf{0}$, $\mathrm{rec}\, Y.Q$, $(\mathrm{rec}^Q\, Y(i).R;)$ and $P \mid Q$ occurs as subterms of P. These conventions ensure that the process $(\mathrm{rec}^e\, X(i).P; Q)$ will eventually terminate the loop and execute Q. Process variables $X[\tilde{k}]$ mentions the channel names $\tilde{k}$ active at unfolding time for technical reasons.

We define the free polarised names $\mathsf{fn}(P)$ of P as usual, with $\mathsf{fn}(X[\tilde{k}]) = \tilde{k}$; substitution of process variables from $X[\tilde{k}]\{P/X\} = P$; and finally value substitution $P\{v/x\}$ in the obvious way, e.g., $k!\langle e\rangle.P\{v/x\} = k!\langle e\{v/x\}\rangle.(P\{v/x\})$. Variable substitution can never affect channels.

Example 2.1. We now show how to model the example BPMN process given in the introduction. To illustrate the possibility of type checking infinite state systems, we use a persistent data object. As our calculus does not contain such a primitive, a data object will be represented by a process DATA(o) communicating on a session channel o.

$$\mathrm{DATA}(o) = \mathrm{rec}\, X.\ o^+?(x).\ \mathrm{rec}\, Y.\ o^+? \begin{cases} \mathsf{read}.\ o^+!\langle x\rangle.\ Y[o^+] \\ \mathsf{write}.\ X[o^+] \\ \mathsf{quit}.\ \mathbf{0} \end{cases}$$

After having received its first initial value, this process repeatedly accepts commands read and write on the session channel o for respectively reading and writing the value of the variable, or the command quit for discarding the data object.

To make examples more readable, we employ the following shorthands. We write $\mathsf{init}(o, v).P$ for $o^-!\langle v\rangle.P$, which initializes the data object; we write $\mathsf{free}\, o$ for $o^-!\mathsf{quit}.\ \mathbf{0}$, the process which terminates the data object session; we write $\mathsf{read}\, o(x).P$ for $o^-!\mathsf{read}.\ o^-?(x).P.$, the process which loads the value of the data object o into the process-local variable x; and finally, we write $o := e.P$ for $o^-!w.o^-!\langle e\rangle.P$, the process which sets the value of a data-object.

The shopping cart process can then be modelled as

$$P(Q) = \mathrm{DATA}(o) \ \mid\ \mathsf{init}(o,\epsilon).\ \mathrm{rec}\,X.k \begin{cases} \mathsf{AI}.\ k?(x).\ \mathsf{read}\,o(y).\ o := add(y,x).\ X[ko^-] \\ \mathsf{RI}.\ k?(x).\ \mathsf{read}\,o(y).\ o := rem(y,x).\ X[ko^-] \\ \mathsf{CO}.\ k?(x).\ \mathsf{read}\,o(y).\ o := add(y,x).\ Q \end{cases} .$$

Here k is the session channel for communicating with the customer and o is the session channel for communicating with the data object modelling order data. We assume our expression language has suitable operators "add" and "rem", adding an removing items from the order. Finally, the process Q is a stand-in for either the (non live) delivery part of the BPMN process in Fig. A or the live delivery part shown in Fig. B.

The non-live delivery loop can be represented by the process

$$D_0 = \mathrm{rec}\,Y.\ \mathsf{read}\,o(y).\ \mathsf{if}\,n(y) > 0 \begin{cases} \mathsf{then}\ k!\mathsf{DI}.\ k!\langle next(y)\rangle.\ o := update(y).\ Y[ko^-] \\ \mathsf{else}\ k!\mathsf{SI}.\ k!\langle inv(y)\rangle.\ \mathsf{free}\,o \end{cases}$$

where $n(y)$ is the integer expression computing from the order y the number of items to send, $next(y)$, $update(y)$ and $inv(y)$ are, respectively, the the next item(s) to be sent; an update of the order to mark that these items have indeed been sent; and the invoice from the order. Note that whether or not this process terminates is entirely dependent on the data operations.

The live delivery with the bounded iteration can be represented by a process using the bounded recursion

$$\begin{aligned} D = \mathsf{read}\,o(y).\ (\mathrm{rec}^{n(y)}\,Y(i).& \\ & k!\mathsf{DI}.\mathsf{read}\,o(y).\ k!\langle pickitem(y,i)\rangle.Y[ko^-]; \\ & \qquad\qquad k!\mathsf{SI}.\ \mathsf{read}\,o(y).\ k!\langle inv(y)\rangle.\ \mathsf{free}\,o) \end{aligned}$$

(The second line is the body of the loop; the third line is the continuation.) Here $pickitem(y,i)$ is the expression extracting the ith item from the order y. □

3 Transition Semantics

We give a labelled transition semantics in Figure C. We assume a total evaluation relation $e \Downarrow v$. Note the absence of a structural congruence. Transition labels are $\lambda ::= k!v \mid k?v \mid k{\oplus}l \mid k\&l \mid \tau \mid \tau : l$. We assume τ is neither a channel nor a polarised channel, define $\mathsf{subj}(k!v) = \mathsf{subj}(k?v) = \mathsf{subj}(k\&l) = \mathsf{subj}(k \oplus l) = k$ and $\mathsf{subj}(\tau) = \mathsf{subj}(\tau : l) = \tau$, and $\overline{\tau} = \tau$. We use these rules along with symmetric rules for [C-ParL] and [C-Com1/2]. Compared to standard CCS or π semantics, there are two significant changes: (1) In the [C-ParL], a transition λ of P is *not* preserved by parallel if the co-channel of the subject of λ is in P'; and (2) in prefix rules, the co-name of the subject cannot appear in the continuation. We impose (1) because if the co-channel of the subject of λ is in P', then $P \mid P'$

$$[\text{C-Out}]\ \frac{e \Downarrow v}{k!\langle e\rangle.P \xrightarrow{k!v} P}\ \overline{k} \notin \mathsf{fn}(P) \qquad [\text{C-In}]\ \frac{}{k?(x).P \xrightarrow{k?v} P\{v/x\}}\ \overline{k} \notin \mathsf{fn}(P)$$

$$[\text{C-Sel}]\ \frac{}{k!l.P \xrightarrow{k\oplus l} P}\ \overline{k} \notin \mathsf{fn}(P) \qquad [\text{C-Bra}]\ \frac{}{k?\{l_i.P_i\}_{i\in I} \xrightarrow{k\&l_i} P_i}\ \overline{k} \notin \mathsf{fn}(P)$$

$$[\text{C-ParL}]\ \frac{P \xrightarrow{\lambda} Q}{P \mid P' \xrightarrow{\lambda} Q \mid P'} \qquad \overline{\mathsf{subj}(\lambda)} \notin \mathsf{fn}(P')$$

$$[\text{C-Com1}]\ \frac{P \xrightarrow{\overline{k}!v} P' \qquad Q \xrightarrow{k?v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad [\text{C-Com2}]\ \frac{P \xrightarrow{\overline{k}\oplus l} P' \qquad Q \xrightarrow{k\&l} Q'}{P \mid Q \xrightarrow{\tau:l} P' \mid Q'}$$

$$[\text{C-Rec}]\ \frac{P\{\mathrm{rec}\,X.P/X\} \xrightarrow{\lambda} Q}{\mathrm{rec}\,X.P \xrightarrow{\lambda} Q} \qquad [\text{C-Prec0}]\ \frac{e \Downarrow 0 \qquad Q \xrightarrow{\lambda} R}{(\mathrm{rec}^e\,X(i).P;Q) \xrightarrow{\lambda} R}$$

$$[\text{C-PrecN}]\ \frac{e \Downarrow n+1 \qquad P\{n/i\}\{(\mathrm{rec}^n\,X(i).P;Q)/X\} \xrightarrow{\lambda} R}{(\mathrm{rec}^e\,X(i).P;Q) \xrightarrow{\lambda} R}$$

$$[\text{C-CondT}]\ \frac{e \Downarrow \mathsf{true} \qquad P \xrightarrow{\lambda} P'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} P'} \qquad [\text{C-CondF}]\ \frac{e \Downarrow \mathsf{false} \qquad Q \xrightarrow{\lambda} Q'}{\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\lambda} Q'}$$

Fig. C. Transition semantics for terms

does not offer synchronisation on λ to its environment; the synchronisation is offered only to P'. As an example, the process $P = c^+!\langle v\rangle.Q \mid c^-?(x).R$ does not have a transition $c^+!\langle v\rangle.Q \mid c^-?(x).R \xrightarrow{c^+!v} Q \mid c^-?(x).R$. If it had such a transition, no environment U able to receive on c^- could be put in parallel with P and form a well-typed process, since both U and $c^-?(d).R$ would then contain the name c^- free. The reason for (2) is similar: If a process $k!\langle e\rangle.P \xrightarrow{k!v} P$, and P contains $\overline{k}$, again no well-typed environment for that process can contain $\overline{k}$.

4 Session Types with Responses

In this section, we generalise binary session types to *session types with responses.* In addition to providing the standard communication safety properties, these also allow us to specify response liveness properties.

We have deliberately omitted types of values (e.g. integers, strings, booleans) being sent, since this can be trivially added and we want to focus on the behavioural aspects of the types. Note also, that compared to standard session types, we do not consider delegation (name passing). Firstly, as illustrated by our example calculus, the types are already expressive enough to cover a non-trivial subset of collaborative processes. Secondly, as we show in the end of the section, session types with responses are already in some sense strictly more expressive than standard session types. Finally, admitting delegation introduces some non-trivial challenges and choices with respect to how to define and guarantee liveness, which we have left for future work.

We first define request/response liveness in the abstract. In general, we shall take it to be the property that "a request is eventually followed by a response".

Definition 4.1. *A* request/response structure *is a tuple* $(A, R, \mathsf{req}, \mathsf{res})$ *where* A *is a set of actions,* R *is a set of responses, and* $\mathsf{req} : A \to R$ *and* $\mathsf{res} : A \to R$ *are maps defining the set of responses requested respectively performed by an action.*

Notation. We write ϵ for the empty string, we let ϕ, ψ range over finite strings, and we let α, β, γ range over finite or infinite sequences. We write sequence concatenation by juxtaposition, i.e., $\phi\alpha$.

Definition 4.2. *Suppose* $(A, R, \mathsf{req}, \mathsf{res})$ *is a request/response structure and* α *a sequence over* A*. Then the* responses $\mathsf{res}(\alpha)$ *of* α *is defined by* $\mathsf{res}(\alpha) = \cup\{\mathsf{res}(a) \mid \exists\varphi, \beta.\ \alpha = \varphi a \beta\}$*. Moreover,* α *is* live *iff* $\alpha = \phi a \beta \implies \mathsf{req}(a) \subseteq \mathsf{res}(\beta)$*.*

Definition 4.3 (LTS with requests/responses). *Let* $(S, L, \to)$ *be an LTS. When the set of labels* L *is the set of actions of a request/response structure, we say that* $(S, L, \to)$ *is an* LTS with requests/responses*, and that a transition sequence of this LTS is* live *when its underlying sequence of labels is.*

Next, the syntax of session types with responses. Let l range over labels and L sets of labels.

$$S, T \ ::= \ \&\{l_i[L_i].T_i\}_{i\in I} \ \mid \ \oplus\{l_i[L_i].T_i\}_{i\in I} \ \mid \ !.T \ \mid \ ?.T \ \mid \ \mu t.T \ \mid \ t \ \mid \ \mathtt{end}$$

By convention, the l_i in each $\&\{l_i[L_i].T_i\}_{i\in I}$ resp. $\oplus\{l_i[L_i].T_i\}_{i\in I}$ are distinct.

A session type is a (possibly infinite) tree of actions permitted for one partner of a two-party communication. The type $\&\{l_i[L_i].T_i\}_{i\in I}$, called *branch*, is the type of *offering* a choice between different continuations. If the partner chooses the label l_i, the session proceeds as T_i. Compared to standard session types, making the choice l_i also requests a subsequent response on every label mentioned in the set of labels L_i; we formalise this in the notion of *responsive trace* below. Dual to branch is *select* $\oplus\{l_i[L_i].T_i\}_{i\in I}$: the type of *making* a choice between different continuations. Like branch, making a choice l_i requests every label in L_i as future responses. The type $!.T$ and $?.T$ are the types of sending and receiving data values. Note that unlike standard session types, channels cannot be communicated. Finally, session types with responses include recursive types. We take the equi-recursive view, identifying a type T and its unfolding into a potentially infinite tree. We define the central notion of *duality* between types as the symmetric relation induced coinductively by the following rules.

$$\frac{}{\mathtt{end} \bowtie \mathtt{end}} \qquad \frac{T \bowtie T'}{!.T \bowtie\ ?.T'} \qquad \frac{T_i \bowtie T'_i \quad J \subseteq I}{\&\{l_i[L_i].T_i\}_{i\in I} \bowtie \oplus\{l_j[L'_j].T'_j\}_{j\in J}} \tag{1}$$

The first rule says that the dual processes agree on when communication stops. The second says that if a process sends a message its dual must receive. Finally, the third says that if one process offers a branch, its dual must select from the offered choices. However, required responses do not need to match for branch and select to be duals: the two participants in a session need not agree on the notion of liveness for the collaborative session.

Example 4.4. Recall from Ex. 2.1 the processes DATA(o) encoding data-object and $P(D)$ encoding the shopping-cart process with live delivery. The former treats the channel o as $T_D = \mu t.?.\mu s.\&\{\mathsf{read}.!.s,\ \mathsf{write}.t,\ \mathsf{quit}.\mathsf{end}\ \}$. The latter treats its channel k to the buyer as $T_P = \mu t.\&\{\mathsf{AI}.?.t,\ \mathsf{RI}.?.t,\ \mathsf{CO}[\{\mathsf{SI}\}].?.\mu t'.\oplus\{\mathsf{DI}.!.t',\ \mathsf{SI}.!.\mathsf{end}\}\}$. □

Having defined the syntax of session types with responses, we proceed to give their semantics. The meaning of a session type is the possible sequences of communication actions it allows, requiring that pending responses eventually be done. Formally, we equip session types with a labeled transition semantics in Fig. D. We emphasise that under the equi-recursive view of session types, the

Type transition labels: $\rho ::= \ ! \mid ? \mid \&l[L] \mid \oplus l[L]$

Type transition label duality: $! \bowtie ?$ and $\&l[L] \bowtie \oplus l[L']$

$$[\text{D-OUT}]\ \frac{}{!.T \xrightarrow{!} T} \qquad \frac{}{?.T \xrightarrow{?} T}\ [\text{D-IN}]$$

$$[\text{D-BRA}]\ \frac{i \in I}{\&\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{\&l_i[L_i]} T_i} \qquad \frac{i \in I}{\oplus\{l_i[L_i].T_i\}_{i\in I} \xrightarrow{\oplus l_i[L_i]} T_i}\ [\text{D-SEL}]$$

Fig. D. Transitions of types (1)

transition system of a recursive type T may in general be infinite.

Taking actions A to be the set of labels ranged over by ρ, and recalling that $\mathcal{L}$ is our universe of labels for branch/select, we obtain a request/response structure $(A, \mathcal{P}(\mathcal{L}), \mathsf{req}, \mathsf{res})$ with the latter two operators defined as follows.

$$\mathsf{res}(!) = \mathsf{res}(?) = \emptyset \quad \mathsf{res}(\&l[L]) = \mathsf{res}(\oplus l[L]) = \{l\}$$
$$\mathsf{req}(!) = \mathsf{req}(?) = \emptyset \quad \mathsf{req}(\&l[L]) = \mathsf{req}(\oplus l[L]) = L$$

Selecting a label l performs the response l; pending responses L associated with that label are conversely requested. The LTS of Fig. D is thus one with responses, and we may speak of its transition sequences being live or not.

Definition 4.5. *Let T be a type. We define:*

1. *The* traces $\mathsf{tr}(T) = \{(\rho_i)_{i\in I} \mid (T_i, \rho_i)_{i\in I}$ *transition sequence of* $T\ \}$
2. *The* responsive traces $\mathsf{tr}_R(T) = \{\alpha \in \mathsf{tr}(T) \mid \alpha$ *live* $\}$.

That is, in responsive traces any request is followed by a response.

Definition 4.6. *A type T is a* standard *session type if it requests no responses, that is, every occurrence of L in it is has $L = \emptyset$. Define* $\mathsf{sel}(T) = l$ *by when* $T = \mathsf{sel}(\&l[L])$ *or* $T = \mathsf{sel}(\oplus l[L]) = l$*, otherwise* ϵ*. We then define:*

1. *The* selection traces $\mathsf{str}(T) = \{\mathsf{sel}(\alpha) \mid \alpha \in \mathsf{tr}(T)\}$

9

2. The responsive selection traces $\mathsf{str}_R(T) = \{\mathsf{sel}(\alpha) \mid \alpha \in \mathsf{tr}_R(T)\}$.
3. The language of standard session types
$\mathcal{T} = \{\alpha \mid \alpha \in \mathsf{str}(T), T$ *is a standard session type*$\}$.
4. The language of responsive session types
$\mathcal{R} = \{\alpha \mid \alpha \in \mathsf{str}_R(T), T$ *is a session type with responses*$\}$.

That is, we compare standard session types and session types of responses by considering the sequences of branch/select labels they admit. This follows recent work on multi-party session types and automata [5,6].

Example 4.7. The type T_P of Example 4.4 has (amongst others) the following two selection traces.

$$t = \mathsf{AI\,CO\,DI\,DI\,SI}$$
$$u = \mathsf{AI\,CO\,DI\,DI\,DI}\cdots$$

Of these, only t is responsive; u is not, since it never selects SI as required by its CO action. That is, $u, v \in \mathsf{str}(T_P)$ and $u \in \mathsf{str}_R(T_P)$, but $v \notin \mathsf{str}_R(T_P)$. □

Theorem 4.8. *The language of session types with responses $\mathcal{R}$ is strictly more expressive than that of standard session types $\mathcal{T}$; that is, $\mathcal{T} \subset \mathcal{R}$.*

Proof (Sketch). The non-strict inclusion is immediate by definition; it remains to prove it strict. For this consider the session type with responses $T = \mu t. \oplus \{a[b].t; b[a].t\}$, which has as responsive traces all strings that have both inifitely many as and infinitely many bs. We can find every sequence a^n as a *prefix* of such a trace. But, (by regularity) any *standard* session type that has all a^n as a prefix of the traces must also have the trace a^ω, which is not a responsive trace of T, and thus the responsive traces of T can not be expressed as the traces of a standard session type.

5 Session Typing

Recall that the standard typing system [12,20] for session types has judgements $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$. We use this typing system without restating it; refer to either [12,20] or the full version of this paper [4]. In this judgement, Θ takes process variables to session type environments; in turn, a *session typing environment* Δ is a finite partial map from channels to types. We write Δ, Δ' for the union of Δ and Δ', defined when their domains are disjoint. We say Δ is *completed* if $\Delta(T) = \mathtt{end}$ when defined; it is *balanced* if $k : T, \overline{k} : U \in \Delta$ implies $T \bowtie U$.

We generalise transitions of types (Fig. D) to session typing environments in Fig. E, with transitions $\delta ::= \tau \mid \tau : l, L \mid k : \rho$. We define $\mathsf{subj}(k : \rho) = k$ and $\mathsf{subj}(\tau : l, L) = \mathsf{subj}(\tau) = \tau$. We lift $\mathsf{sel}(-), \mathsf{req}(-)$, and $\mathsf{res}(-)$ to actions δ in the obvious way, e.g., $\mathsf{req}(\tau : l, L) = L$. The type environment transition is thus an LTS with responses, and we may speak of its transition sequences being live.

$$[\text{E-Lift}]\ \frac{T \xrightarrow{\rho} T'}{k : T \xrightarrow{k:\rho} k : T'} \qquad [\text{E-Par}]\ \frac{\Delta \xrightarrow{\delta} \Delta'}{\Delta, \Delta'' \xrightarrow{\delta} \Delta', \Delta''}$$

$$[\text{E-Com1}]\ \frac{\Delta_1 \xrightarrow{k:!} \Delta_1' \qquad \Delta_2 \xrightarrow{\overline{k}:?} \Delta_2'}{\Delta_1, \Delta_2 \xrightarrow{\tau} \Delta_1', \Delta_2'} \qquad [\text{E-Com2}]\ \frac{\Delta_1 \xrightarrow{k:\oplus l[L]} \Delta_1' \qquad \Delta_2 \xrightarrow{\overline{k}:\& l[L']} \Delta_2'}{\Delta_1, \Delta_2 \xrightarrow{\tau:l, L\cup L'} \Delta_1', \Delta_2'}$$

Fig. E. Transitions of types (2)

Definition 5.1. *We define a binary relation on type transition labels δ and transition labels λ, written $\delta \simeq \lambda$, as follows.*

$$\tau \simeq \tau \qquad k : \& l[L] \simeq k\& l \qquad k : ! \simeq k!v$$
$$\tau : l, L \simeq \tau : l \qquad k : \oplus l[L] \simeq k \oplus l \qquad k : ? \simeq k?x$$

Theorem 5.2. *If $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta$ and $P \xrightarrow{\lambda} Q$, then there exists $\delta \simeq \lambda$ s.t. $\Delta \xrightarrow{\delta} \Delta'$ and $\Gamma \vdash_{\mathsf{std}} Q \triangleright \Delta'$.*

Definition 5.3. *The* typed transition system *is the transition system which has states $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta$ and transitions $\Gamma \vdash_{\mathsf{std}} P \triangleright \Delta \xrightarrow{\lambda,\delta} \Gamma \vdash_{\mathsf{std}} P' \triangleright \Delta'$ whenever there exist transitions $P \xrightarrow{\lambda} P'$ and $\Delta \xrightarrow{\delta} \Delta'$ with $\delta \simeq \lambda$.*

We can now say what it means for a process to be live.

Definition 5.4 (Live process). *A well-typed process $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$ is* live *wrt. Θ, Δ iff for any maximal transition sequence $(P_i, \lambda_i)_i$ of P there exists a live type transition sequence $(\Delta_i, \delta_i)_i$ of Δ s.t. $((P_i, \Delta_i), (\lambda_i, \delta_i))_i$ is a typed transition sequence of $\Theta \vdash_{\mathsf{std}} P \triangleright \Delta$.*

Example 5.5. Wrt. the standard session typing system, *both* of the processes $P(D_0)$ and $P(D)$ of Example 2.1 are typable wrt. the types we postulated for them in Example 4.4. Specifically, we have $\cdot \vdash_{\mathsf{std}} P(D_0) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$ and similarly $P(D)$. The judgement means that the process $P(D)$ treats k according to T_P and the (two ends of) the data object according to T_D and its dual $\overline{T_D}$. The standard session typing system of course does not act on our liveness annotations, and so does not care that $P(D_0)$ is not live.

6 Typing system for liveness

We now give our extended typing system for session types with responses. The central judgement will be $\Gamma; L \vdash P \triangleright \Delta$, with the intended meaning that "with process variables Γ and pending responses L, the process P conforms to Δ." We shall see in the next section that a well-typed lock-free P is live and will eventually perform every response in in L. We need:

1. *Session typing environments* Δ defined at the start of Section 5.
2. *Response environments* L are simply sets of branch/select labels.

3. *Process variable environments* Γ are finite partial maps from process variables X to tuples (L, L, Δ) or (L, Δ). We write these (A, I, Δ) for (A)ccumulated selections and request (I)nvariant. We define $(\Gamma+L)(X) = (A \cup L, I, \Delta)$ when $\Gamma(X) = (A, I, \Delta)$ and $\Gamma(X)$ otherwise, writing $\Gamma + l$ instead of $\Gamma + \{l\}$.

The liveness-guaranteeing type systems is in Fig. F. We explain the rules; first,

$$[\text{F-OUT}]\ \frac{\Gamma; L \vdash P \triangleright \Delta, k : T}{\Gamma; L \vdash k!\langle e\rangle.P \triangleright \Delta, k : !.T} \qquad [\text{F-IN}]\ \frac{\Gamma; L \vdash P \triangleright \Delta, k : T}{\Gamma; L \vdash k?(x).P \triangleright \Delta, k : ?.T}$$

$$[\text{F-BRA}]\ \frac{\forall i \in I : \quad \Gamma + l_i; (L \setminus l_i) \cup L_i \vdash P_i \triangleright \Delta, k : T_i}{\Gamma; L \vdash \ k?\{l_i.P_i\}_{i \in I} \triangleright \Delta, k : \&\{l_i[L_i].T_i\}_{i \in I}}$$

$$[\text{F-SEL}]\ \frac{\Gamma + l_j; (L \setminus l_j) \cup L_j \vdash P \triangleright \Delta, k : T_j}{\Gamma; L \vdash k!l_j.P \triangleright \Delta, k : \oplus\{l_i[L_i].T_i\}_{i \in I}} \quad (j \in I)$$

$$[\text{F-PAR}]\ \frac{\Gamma; L_1 \vdash P_1 \triangleright \Delta_1 \qquad \Gamma; L_2 \vdash P_2 \triangleright \Delta_2}{\Gamma; L_1 \cup L_2 \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2} \qquad [\text{F-INACT}]\ \frac{\Delta \text{ completed}}{\Gamma; \emptyset \vdash \mathbf{0} \triangleright \Delta}$$

$$[\text{F-VAR}]\ \frac{L \subseteq I \subseteq A \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (A, I, \Delta); L \vdash X[\tilde{k}] \triangleright \Delta} \qquad [\text{F-VARP}]\ \frac{L \subseteq L' \qquad \mathsf{dom}(\Delta) = \tilde{k}}{\Gamma, X : (L', \Delta); L \vdash X[\tilde{k}] \triangleright \Delta}$$

$$[\text{F-RECP}]\ \frac{\Gamma, X : (L', \Delta); L' \vdash P \triangleright \Delta \qquad \Gamma; L' \vdash Q \triangleright \Delta \qquad L \subseteq L'}{\Gamma; L \vdash (\mathrm{rec}^e\, X(i).P; Q) \triangleright \Delta}$$

$$[\text{F-REC}]\ \frac{\Gamma, X : (\emptyset, I, \Delta); I \vdash P \triangleright \Delta \quad L \subseteq I}{\Gamma; L \vdash \mathrm{rec}\, X.P \triangleright \Delta} \qquad [\text{F-COND}]\ \frac{\Gamma; L \vdash P \triangleright \Delta \quad \Gamma; L \vdash Q \triangleright \Delta}{\Gamma; L \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

Fig. F. Typing System

branch/select, rules [F-BRA]/[F-SEL]. To type $k!l.P$ wrt. $k : \oplus l[L'].T$, we must verify that P performs every response in L'. We maintain an environment L of currently pending responses for this. In the hypothesis, when typing P, we add the new pending responses L'. But selecting l performs the response l, so altogether, to support pending responses L in the conclusion, we must have pending responses $L \setminus \{l\} \cup L'$ in the hypothesis. Branching is similar.

For finite processes, if the inactive process must be typed with the empty request environment, liveness is ensured. Hence in the rule $\mathbf{0}$, the pending responses environment is required to be empty. For infinite processes there is no particular point at which we can insist there is no pending responses. Consider $\mathrm{rec}\, X.k \oplus a.\ k \oplus b.\ X[k]$ typeable under $k : \mu t. \oplus \{a[b].t; b[a].t\}$. This process has the single transition sequence $P \xrightarrow{k \oplus a} k \oplus b.\ P \xrightarrow{k \oplus b} P \xrightarrow{k \oplus a} \cdots$. At each state but the initial one, the process has a pending response: in states P it needs to respond b; in states $k \oplus b.\ P$ it needs to respond a. Yet the process is live: *any response requested in the body of the recursion is also discharged in the body*, although not necessarily in the proper order. In general, infinite behaviour arises because of unfolding of recursion, so if the body of every recursion discharges the requests of that body, even if not in the proper order, responses are ensured.

For general recursion, [F-Rec] and [F-Var], we need to check that there exists an invariant, a set of responses, such that the body of a recursion requests at most that set, and reponds with at least that set. In the process variable environment Γ we record this response invariant for each variable, along with a tally of the responses performed since the start of the recursion. That tally is then updated by the rules [F-Sel]/[F-Bra] for select and branch. The rule for process variable [F-Var] typing then check that the tally includes the invariant, and that the invariant includes every currently pending response.

Definition 6.1. *We define the* standard *process variable environment* $\mathsf{std}(\Gamma)$ *associated with a process variable environment* Γ $\mathsf{std}(\Gamma)(X) = \Delta$ *when* $\Gamma(X) = (A, I, \Delta)$ *or* $\Gamma(X) = \Delta$.

Theorem 6.2. *If* $\Gamma; L \vdash P \triangleright \Delta$ *then also* $\mathsf{std}(\Gamma) \vdash_{\mathsf{std}} P \triangleright \Delta$.

Theorem 6.3 (Subject reduction). *Suppose that* $\cdot; L \vdash P \triangleright \Delta$ *with and* $P \xrightarrow{\lambda} Q$. *Then there exists a type transition* $\Delta \xrightarrow{\delta} \Delta'$ *with* $\delta \simeq \lambda$, *such that* $\cdot; (L \setminus \mathsf{res}(\delta)) \cup \mathsf{req}(\delta) \vdash Q \triangleright \Delta'$. *Moreover, if* Δ *balanced then also* Δ' *balanced.*

Example 6.4. Wrt. the typing system of Figure F, the process $P(D)$ is typable wrt. the types we postulated for it in Example 4.4. The process $P(D_0)$ on the other hand is not. That is, we have $\cdot; \emptyset \vdash P(D) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$, but the same does *not* hold for $P(D_0)$.

We also exemplify a typing judgment with non-trivial guaranteed responses. The process D, the order-fulfillment part of $P(D)$, can in fact be typed

$$\cdot; \{\mathsf{SI}\} \vdash D \triangleright k : \mu t'. \oplus\{\mathsf{DI}.!.t', \mathsf{SI}.!.\mathsf{end}\},\ o^- : \overline{T_D}$$

The interesting bit is the left-most $\{\mathsf{SI}\}$, indicating intuitively that this process will eventually select SI in *any* execution. If you look back at the actual process D in Example 2.1, you will see that D has this property precisely because it implements a bounded recursion.

7 Liveness

We prove that a lock-free process well-typed under our liveness typing system is indeed live under fair executions. For defining lock-freedom and fairness, we must track occurrences of prefixes across transitions. This is straightforward in the absence of a structural congruence; refer to [9] for a formal treatment.

Definition 7.1. *A* prefix M *is a process on one of the forms* $k!\langle e\rangle.P$, $k?(x).P$, $k?\{l_i.P_i\}$, *or* $k!l.P$. *An* occurence *of a prefix* M *in a process* P *is a path in the abstract syntax tree of* P *to a subterm on the form* M *(see [9] for details). An occurrence of a prefix* P *in* M *where* $P \xrightarrow{\lambda} Q$ *is* preserved *by the latter if* M *has the same occurrence in* Q*;* executed *otherwise. It is* enabled *if it is executed by some transition, and* top-level *if it is not nested in another prefix.*

Definition 7.2. *An infinite transition sequence $s = (P_i, \lambda_i)_{i\in\mathbb{N}}$ is* fair *iff whenever a prefix M occurs enabled in P_n then some $m \geq n$ has $P_m \xrightarrow{\lambda_m} P_{m+1}$ executing that occurence.*

Definition 7.3. *A transition sequence sis* terminated *iff it has length n and $P_n \not\rightarrow$. It is* maximal *iff it is finite and terminated or infinite and fair.*

Definition 7.4. *A maximal transition sequence (P_i, λ_i) is lock-free iff whenever there is a top-level occurence of a prefix M in P_i, then there exists some $j \geq i$ s.t. $P_j \xrightarrow{\lambda_j} P_{j+1}$ executes that occurrence. A process is lock-free iff all its transition sequences are.*

The central liveness result hinges on the following proposition, which links the typing judgment to the semantics of a process.

Definition 7.5. *For a process transition label λ, define $\mathsf{sel}(\lambda)$ by $\mathsf{sel}(k!v) = \mathsf{sel}(k?v) = \mathsf{sel}(\tau) = \emptyset$ and $\mathsf{sel}(k\&l) = \mathsf{sel}(k \oplus l) = \mathsf{sel}(\tau : l) = l$. Given a trace α we lift $\mathsf{sel}(-)$ pointwise, that is, $\mathsf{sel}(\alpha) = \{\mathsf{sel}(\lambda) \mid \alpha = \phi\lambda\alpha'\}$.*

Proposition 7.6. *Suppose $\cdot\,; L \vdash P \triangleright \Delta$ with P lock-free, and let $s = (P_i, \alpha_i)_i$ be a maximal transition sequence of P. Then $L \subseteq \mathsf{sel}(\alpha)$.*

Example 7.7. We saw in Example 6.4 that the process D of Example 2.1 is typable $\cdot; \{\mathsf{SI}\} \vdash D \triangleright \cdots$. By Proposition 7.6 above, noting that D is clearly lock-free, every maximal transition sequence of D must eventually select SI.

Theorem 7.8. *Suppose $\cdot\,; L \vdash P \triangleright \Delta$ with P lock-free. Then P is live for $\cdot\,, \Delta$.*

Example 7.9. We saw in Example 6.4 that $P(D)$ is typable as $\cdot; \emptyset \vdash P(D_0) \triangleright k : T_P, o^+ : T_D, o^- : \overline{T_D}$ and so, according to the above Theorem, it is live, and so must will always uphold the liveness guarantee in T_P: if CO is selected, then eventually also SI is selected. Or in the intuition of the example: If the buyer performs “Checkout”, he is guaranteed to subsequently receive an invoice.

8 Conclusion and Future Work

We introduced a conservative generalization of binary session types to *session types with responses*, which allows to specify response liveness properties as part of the types. We showed that session types with responses are strictly more expressive (wrt. the classes of behaviours they can express) than standard binary session types. We provided a typing system for a process calculus of processes similar to a non-trivial subset of collaborative BPMN processes with possibly infinite loops and bounded iteration and proved that lock-free, well typed processes are live. We have identified several interesting directions for future work: Firstly, the present techniques could be lifted to multi-party session types, which guarantees lock-freedom outright. Second, investigate more general liveness properties. Third, add delegation (channel passing). It seems trivial to allow delegation of sessions, *if they have no pending (unfulfilled) responses*, but not otherwise. Finally, and more speculatively, we plan to investigate relations to “fair subtyping” studied in [17].

References

1. Bettini, L., M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini and N. Yoshida, *Global progress in dynamically interleaved multiparty sessions*, in: *CONCUR*, 2008, pp. 418–433.
2. Carbone, M. and S. Debois, *A graphical approach to progress for structured communication in web services*, in: *ICE*, 2010, pp. 13–27.
3. Coppo, M., M. Dezani-Ciancaglini, L. Padovani and N. Yoshida, *Inference of global progress properties for dynamically interleaved multiparty sessions*, in: *COORDINATION*, 2013, pp. 45–59.
4. Debois, S., T. Hildebrandt, T. Slaats and N. Yoshida, *Type checking liveness for collaborative processes with bounded and unbounded recursion (full version)*. URL `http://www.itu.dk/~hilde/liveness-full.pdf`
5. Deniélou, P.-M. and N. Yoshida, *Multiparty session types meet communicating automata*, in: *ESOP*, 2012, pp. 194–213.
6. Deniélou, P.-M. and N. Yoshida, *Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types*, in: *ICALP (2)*, 2013, pp. 174–186.
7. Dezani-Ciancaglini, M., U. de'Liguoro and N. Yoshida, *On progress for structured communications*, in: *TGC*, 2007, pp. 257–275.
8. Dezani-Ciancaglini, M., S. Drossopoulou, D. Mostrous and N. Yoshida, *Objects and session types*, Inf. Comput. **207** (2009), pp. 595–641.
9. Fossati, L., K. Honda and N. Yoshida, *Intensional and extensional characterisation of global progress in the π-calculus*, in: *CONCUR*, 2012, pp. 287–301.
10. Gay, S. J. and M. Hole, *Subtyping for session types in the pi calculus*, Acta Inf. **42** (2005), pp. 191–225.
11. Honda, K., A. Mukhamedov, G. Brown, T.-C. Chen and N. Yoshida, *Scribbling interactions with a formal foundation*, in: *ICDCIT*, 2011, pp. 55–75.
12. Honda, K., V. Vasconcelos and M. Kubo, *Language primitives and type discipline for structured communication-based programming*, in: *ESOP*, 1998, pp. 122–138.
13. Honda, K., N. Yoshida and M. Carbone, *Multiparty asynchronous session types*, in: *POPL*, 2008, pp. 273–284.
14. Hu, R., N. Yoshida and K. Honda, *Session-based distributed programming in Java*, in: J. Vitek, editor, *ECOOP '08*, LNCS **5142**, 2008 pp. 516–541.
15. Mostrous, D. and V. T. Vasconcelos, *Session typing for a featherweight Erlang*, in: *COORDINATION*, 2011, pp. 95–109.
16. Object Management Group BPMN Technical Committee, *Business Process Model and Notation, v2.0*, Webpage (2011), `http://www.omg.org/spec/BPMN/2.0/PDF`.
17. Padovani, L., *Fair subtyping for open session types*, in: *ICALP (2)*, 2013, pp. 373–384.
18. Vasconcelos, V., *Fundamentals of session types*, I&C **217** (2012), pp. 52–70.
19. Vieira, H. T. and V. T. Vasconcelos, *Typing progress in communication-centred systems*, in: *COORDINATION*, 2013, pp. 236–250.
20. Yoshida, N. and V. T. Vasconcelos, *Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication*, ENTCS **171** (2007), pp. 73–93.

7 Safe Runtime Adaptation of Flexible Processes

Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs

Raghava Rao Mukkamala
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
rao@itu.dk

Thomas Hildebrandt
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
hilde@itu.dk

Tijs Slaats
IT University of Copenhagen and
Exformatics A/S
2100 Copenhagen, Denmark
tslaats@itu.dk

***Abstract*—We describe how the declarative Dynamic Condition Response (DCR) Graphs process model can be used for trustworthy adaptive case management by leveraging the flexible execution, dynamic composition and adaptation supported by DCR Graphs. The dynamically composed and adapted graphs are verified for deadlock freedom and liveness in the SPIN model checker by utilizing a mapping from DCR Graphs to PROMELA code. We exemplify the approach by a small workflow extracted from a field study at a danish hospital.**

***Keywords*-Adaptive Case Management, Declarative Business Processes, Verification**

I. Introduction

It has been recognized early [2], [29], that supporting dynamic (i.e. run-time) changes of process descriptions is one of the key challenges in workflow management systems. The challenge has been receiving increasing interest recently as a consequence of the demand for efficient IT systems supporting so-called *adaptive case management* (ACM) processes [14], [23], [24], [27], characterized by being *unpredictable* and *individual* in nature and being carried out by knowledge workers.

Healthcare services are typical sources of case management processes that exercise the challenges of evolutionary changes and being unpredictable and individual in nature. And the lack of support for dynamic adaptation and composition of processes is indeed one of the limiting factors for the usage in practice of the many standardized treatment plans and clinical guidelines being defined around the world [15], [16].

Changing a process description while process instances are executing may cause side effects such as un-intentional repetition or skipping of tasks and introduction of deadlocks and livelocks. This situation is referred to as the *dynamic change bug*. Elimination of the dynamic change bug calls for the use of formal process models and development of verification techniques that support dynamic changes and analysis for deadlocks and livelocks. In particular, formal *declarative* models [4], [12], [31] have been put forward as offering more flexibility in execution than the traditional approaches using explicit flow-graphs.

In the present paper, we propose an approach to specification and execution of trustworthy adaptive case management processes based on Dynamic Condition Response Graphs (DCR Graphs) [4], [6]–[8], [19]. DCR Graphs is a formal, declarative process modeling language developed in the Trustworthy Pervasive Healthcare Services (TrustCare.dk) research project as part of the first author's PhD project [17], and currently being embedded in the Exformatics case management tools as part of the industrial PhD project carried out by the last author. A brief overview of the DCR Graphs Tools implemented in Exformatics is presented in [26] and a graphical editor for DCR Graphs can be downloaded from [25].

A DCR Graph specifies a process as a set of labelled events related by five different relations specifying the constraints on the execution of events. The label of an event typically indicates the name of an atomic activity and by whom/which role the activity can be executed, while the constraints declare rules for the ordering of events.

As a running example, we will consider a simple healthcare process inspired by a previous field study at a danish hospital [15]. A fragment of the process is shown as a DCR Graph in Fig. 1 below.

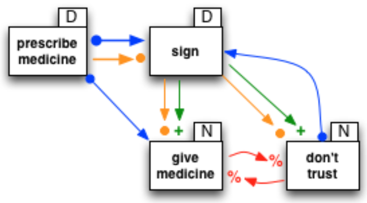


Figure 1. Prescribe medicine process fragment

The graph consists of four events shown as boxes labelled by the name of the activity (prescribe medicine, sign, give medicine, and don't trust) and the roles of whom can perform the activities, in this example either (D)octor or (N)urse. Intuitively, the process allows a doctor to prescribe medicine (any number of times) and subsequently being required to certify the prescription by a signature. The nurse must then give the medicine (to a patient) or alternatively indicate that the prescription is not trusted. In the latter case the doctor is required to sign again (possibly after making a new prescription).

The prescribe medicine event is related to the sign event by a *condition* relation ($\rightarrow\bullet$), which declares that the sign event can not happen before at least one prescribe medicine event has happened. Similarly, the sign event is related to the give medicine and don't trust events, meaning that the sign event must have happened before the give medicine and don't trust events can happen.

Dually, the prescribe medicine event is related to the sign and give medicine events by a *response* relation ($\bullet\rightarrow$). The response relation declares that if the prescribe medicine event happens during an execution, it must eventually be followed (as a response) by a sign event and a give medicine event for the execution to be accepting (completed). But note, a single sign event and give medicine event can fulfill the response requirement of several preceeding prescribe medicine events. For instance, an execution starting with two prescribe medicine events and then a sign event is possible (because the condition for the sign event is fullfilled) but not (yet) completed since the give medicine event is a pending response. Now, if the execution is continued with a give medicine event then it is completed. It may still however continue, e.g. with a new prescribe medicine event. In this case the execution is no longer completed, since sign and give medicine are again pending responses.

The give medicine and don't trust events are related to each other by the *exclude* relation ($\rightarrow\%$). The exclude relation from give medicine to don't trust declares that the don't trust event will be excluded from the process if give medicine is executed. Similarly, the exclude relation from don't trust to give medicine declares that the give medicine event will be excluded from the process if don't trust is executed. That is, the two events are mutually exclusive. However, sign is related to give medicine and don't trust by an *include* relation ($\rightarrow+$), which means that whenever sign happens, the two events give medicine and don't trust are included again (if they were excluded). Note that condition relations from an excluded event are not considered, and if an excluded event is required to be executed (as a response), this requirement is also ignored as long as the event is excluded.

The intuition is that give medicine will be executed if the nurse trusts the prescription and don't trust if the nurse does not trust the prescription. In the latter case, the sign event is required to be executed again due to the response relation from don't trust to sign. In that case, the doctor will check his prescription, and may make new prescriptions but must sign again, whereafter the choice of giving the medicine or not trusting is made possible again by the inclusion relation.

A key feature of DCR Graphs is that the operational semantics indicated above can be formalized by representing the state of a process by a *marking* of the graph. The marking consists of three finite sets of events, $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, representing respectively the previously executed events, the events that are required to be executed in the future (as responses) or excluded, and the currently included events. This information is sufficient to infer enabledness of an event from the relations of the graph and to infer if an execution is completed. If we again consider the execution starting with two prescribe medicine events and then a sign event, this execution leads to the marking $(\{\mathsf{prescribe\,medicine}, \mathsf{sign}\}, \{\mathsf{give\,medicine}\}, \mathsf{E})$, where $\mathsf{E} = \{\mathsf{prescribe\,medicine}, \mathsf{sign}, \mathsf{give\,medicine}, \mathsf{don't\,trust}\}$ is the set of all events in the graph. Continuing the execution by the give medicine event then leads to the marking $(\{\mathsf{prescribe\,medicine}, \mathsf{sign}, \mathsf{give\,medicine}\}, \emptyset, \mathsf{E}\backslash\{\mathsf{don't\,trust}\})$.

In [17], [19] it is shown that the operational semantics of DCR Graphs can be mapped to Büchi-automata [19]. This makes it possible to formally verify temporal properties of the processes, and in particular deadlock freedom and liveness, using standard tools as for instance the SPIN verification tool [10], [11], [18].

A. DCR Graphs for ACM

The new contribution of the present paper is to describe how DCR Graphs and formal verification of such can be used for trustworthy, adaptive case management. Due to its emergent nature, visibility and control of an ACM process can only be achieved in the context of the execution of a process instance [27]. Therefore, case/knowledge workers continuously adapt the process activities to achieve their (sub)goals successfully [20]. At the same time, due to frequent adaptation, a process may end up in a situation, where it is no longer possible to achieve the overall goal of the process. We primarily use the term trustworthy to indicate that the adapted processes represented as DCR Graphs can be verified before execution is continued. Ideally, the application of formal verification techniques will not only enhance the trustworthiness of the processes, but also help knowledge workers in making suitable adaptive changes.

We demonstrate below, that the declarative nature of DCR Graphs makes it well suited for handling run-time changes and thus the emergent nature of an ACM process. Declarative models are usually considered harder to perceive than imperative process models based on explicit flow graphs. However, the simple representation of the run-time state by a marking on the DCR Graph and the verification step, help to perceive the meaning (and state) of the process and to ensure that the process execution can still be completed, i.e. the goal of the process can be met.

Fig. 2 below illustrates the normal iteration cycle through three phases of a trustworthy execution of a DCR Graph.

The execution starts in the phase *model & adapt*, where an ACM process is modelled either from scratch or by

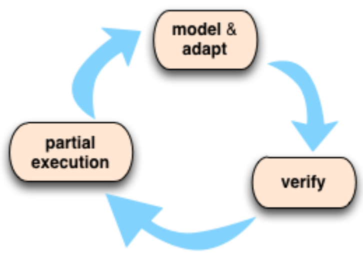
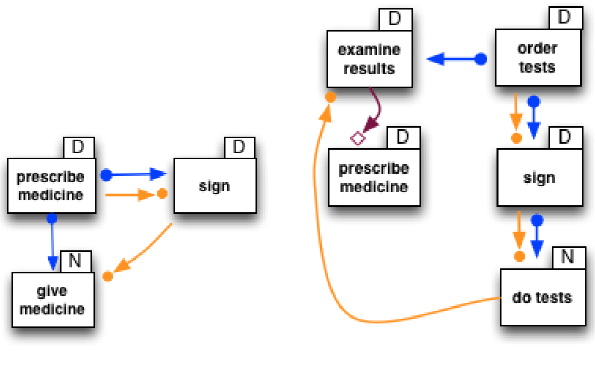


Figure 2. Execution phases of a DCR Graph for ACM

selecting one or more DCR Graphs process fragments (e.g. provided in repositories) which are composed and adapted. In dynamic environments, process knowledge can be local and fragmentary, confined to a certain situation or context. Process Fragments [1] represent a notion of partial and local knowledge, which can be integrated or composed dynamically at design time or run-time. Adopting the notion of process fragments, DCR Graphs can be used as process fragments to represent a partial perspective of a complex process, which can be combined through dynamic composition. Formally speaking, there is no difference between a normal DCR Graph and a DCR Graph representing a process fragment, except that the fragment DCR Graph might represent a subgoal or partial functionality like reusable templates. Fig 3 shows two such fragments for our healthcare example, to be explained in Sec. IV.



(a) prescribe medicine (b) order tests

Figure 3. DCR Graph fragments

In the next phase, *verify*, formal verification techniques will be applied to the process. For instance, one could verify that the process does not allow deadlocks, always allows progress and to achieve the overall goal, i.e. completion by (continued) execution or exclusion of all pending response events.

After successful verification of the process, the execution can proceed to the next phase, *partial execution*, where the process can be executed further until it reaches a point where further adaptation is required and the execution moves to the model & adapt phase, starting a new iteration. However, note that (in the spirit of ACM) it should also be possible to go back to the modelling phase after verification, to skip verification after the modelling phase, or go back to the verify phase after the partial execution.

In this way, an ACM process represented as DCR Graphs can be modelled, verified and executed iteratively, where the emergence of new knowledge can be used by the case workers to adapt the processes at run-time.

However, one may ask why the proposed approach can not simply prevent adoptions that will lead to dead/livelocks. We believe that the general approach allowing to compose/adapt and have intermediate models containing potential deadlocks/livelocks would be valuable as it gives more flexibility in the modeling and adaptation phase. It can be seen as analogous to be able to write a program that does not type check, and then correct the errors, as opposed to only be able to add code parts that lead to a well-typed program.

B. *Structure of paper*

The rest of the paper is structured as follows. We discuss related work in Sec. II whereas in Sec. III we further elaborate the idea of using DCR Graphs for ACM by our healthcare example. In Sec. IV we then formally define the adaptation operations on DCR Graphs, after recalling the formal definition of DCR Graphs and their execution. In Sec. V we then formally define what it means for a DCR Graph to be deadlocked and live, introducing new notions of *strongly* deadlock free and live processes which guarantee progress even if only events that are required as responses are executed. This is in particular relevant if the execution of the DCR Graphs is distributed on different peers (e.g. according to the different roles) as considered in [7]. In Sec. VI we then describe how to verify safety and liveness properties on DCR Graphs (as defined in Sec. V) using the SPIN model checking tool and based on the mapping of DCR Graphs to Büchi-automata [19]. Finally, Sec. VII concludes the paper.

II. Related Work

The issue of dynamic change [2] in workflow systems has been investigated thoroughly for Petri net and graph based models. In [29] Van der Aalst described an approach to find change regions in WorkFlow nets, which represent the parts of a model that are effected by a change. He also proved that a change can be safely applied to a part outside the change region, simultaneously preserving the soundness of a workflow instance. In [22] Reichert et al. presented a framework for the support of adaptive changes in the graph-based workflow model ADEPT. They developed a complete and minimal set of change operations that will allow for modifying an ADEPT workflow at run-time, while still preserving its consistency and correctness.

The previous work often take as a correctness criteria in previous approaches [2], [22], [29], that the state of the instance after applying the change, could have been reached from the initial state by replaying the past run. We find that only allowing changes that are consistent with the past history too strong for ACM. Instead we advocate recording the change as part of the execution sequence.

Recent studies [24], [28] have indicated that BPMN-like languages are not suitable for ACM. One reason is that the processes are described as procedures. Procedures tend to over-specify the processes, and also, the changes one can apply must be formulated as changes to the procedure. On the other hand, declarative workflow models [4], [12], [23], [31], including DCR Graphs, have been proposed as a alternative to traditional workflow models to handle unpredictability and emergent nature of ACM processes. Here process are described by declaring the constraints and goals, which usually under-specify the process and supports changes to the constraints and goals. Declare [30] is a constraint-based declarative workflow model formalized using linear time logic (LTL). Similar to DCR Graphs, Declare also supports adaptive changes such as add/remove constraints and activities, however, since they can not be interpreted in an immediate state, it is required that the trace of the past execution satisfies the LTL formulae corresponding to the change. That is, as discussed above, only changes that are consistent with the run so far are allowed.

A declarative approach using *Guard-Stage-Milestone* [12] based on ECA-like rules for specification of life cycles on Business artifacts was proposed in the recent years. To the best of our knowledge, no work on adaptive changes for the GSM model has been published yet. However, it has been advocated as a model for ACM due to its rich data-centric approach and declarative nature and forms the basis for the recent Case Management Model And Notation (CMMN) [21] proposed by OMG, which includes support for dynamic changes. Also the IBM case manger [3] includes some support for dynamic changes. Furthermore, a modeling approach based on Declarative Configurable Process Specifications [24] is being developed for automated support of case management processes. Using declarative modeling, their model supports process adaptability by using configurable data objects and context based configuration rules, but does not support process run-time adaptation when compared to DCR Graphs.

The use of SPIN for verification of business processes was studied earlier. In [13], authors used SPIN to verify business processes by translating an imperative process specification into state machine description in Promela. In this paper, we translated a declarative process specification in DCR Graphs to Promela by mapping it to Bu̇chi-automata [19].

Finally, we have recently proposed a *join* operator [8] for modular composition and refinement of DCR Graphs and to use it as a formal basis for modular implementation of context sensitive and aspect oriented processes. The *compose* and *change* operations can be derived from the *join* operator, however, in this paper we have chosen to define the adaptation operations directly to make them more straightforward and easier to understand.

III. DCR Graphs for ACM by Example

In this section, we will discuss the adaptation operations for DCR Graphs and exemplify the adaptiveness of DCR Graphs for ACM using the healthcare example.

As adaptation operations we consider the operations of *adding/removing an event*, *adding/removing a constraint between two events*, *changing an event*, *relabelling an event*, *adapting the marking* and *forcing execution* of a non-enabled event. The operations of adding events (to the graph or the marking) and constraints are facilitated by a general compositition operation, which simply takes the union of two graphs as described formally in the next section.

As an example of an ACM process, consider a healthcare workflow where a doctor during the initial consultation realizes that some medical tests are needed before giving the medicine to a patient. In the initial modelling phase, the doctor thus selects in a repository the prescribe medicine DCR Graph fragment in Fig. 3(a) and the order tests process fragment in Fig. 3(b) and compose them. The doctor then proceeds to execute the order test event resulting in the DCR Graph instance shown in Fig. 4.

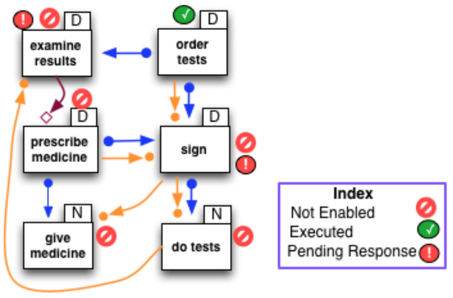


Figure 4. Composed prescribe medicine example with live lock.

The tick mark in the green circle on order tests represents that order tests has been executed, i.e. it is in the executed events set of the marking. Additionally, due to the response relation from order tests to the examine tests and sign events, they have a pending response, i.e. they are in the response events set of the marking. This is indicated by red circle with an exclamation mark. Now, the arrow from examine tests to prescribe medicine having a diamond at its head is the last constraint that we have not yet explained, called the *milestone* constraint. The milestone constraint disables the target event (in this case prescribe medicine) if the source event has a pending response and is included, as it is the case in Fig. 4. The intuition is that the source event must be in a completed state (the milestone reached) before the target event can be executed. Disabled events are indicated with red stop signs. Note that the sign, examine tests and give medicine events are also not enabled because of the condition constraints from prescribe medicine, do tests and sign respectively.

Looking carefully at the DCR Graph in Fig. 4, one may notice that, in order to sign for ordering the tests, the prescribe medicine event must have been done first, which requires the examine tests has been done first (to remove the pending response), which then requires that do tests event has to be done first, because of the condition relation from do tests to examine tests. Alas, the do tests event is blocked by the sign step. In other words, we have a cycle of events blocking each other, in which two of the events are actually required to be executed to complete the workflow. The DCR Graph is not deadlocked, since the order tests event can be repeatedly executed, but this will not change the marking and thus not allow the doctor to make further progress. Hence, the DCR Graph is live locked as explained in Def. 9, as it will never be able to execute or exclude the pending response events (sign and examine tests).

The process instance can be adapted in many ways to remove the live-lock. One way to solve the problem is to force execution of the disabled sign event. However, in fact, the live lock happens because of a modeling error. The doctor should have two separate sign events, one for prescribe medicine and one for order tests. This can again be achieved in many ways. A simple way is to rename the sign events to fresh event names before composing the graphs, which would result in the DCR Graphs in 5 (again after the execution of order tests). Verification of this graph shows that it is deadlock and livelock free.

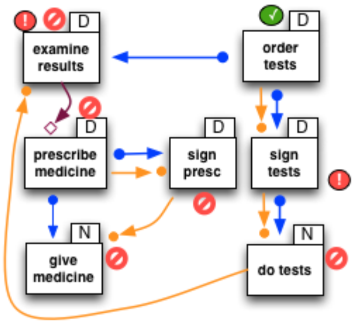


Figure 5. Adapted prescribe medicine example

IV. Adaptive DCR Graphs Formally

In this section we first recall from [17] the formal definitions of DCR Graphs and then give the formal definitions of the new adaptation operations. We employ the following notation: We assume infinite sets $\mathbb{E}$ and $\mathbb{L}$ for events and labels respectively. For a set E we write $\mathcal{P}(E)$ for the power set of E (i.e. set of all subsets of E). For a binary relation $\rightarrow \subseteq E \times E$ and a subset $\xi \subseteq E$ we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{e \in E \mid (\exists e' \in \xi \mid e \rightarrow e')\}$ and the set $\{e \in E \mid (\exists e' \in \xi \mid e' \rightarrow e)\}$ respectively, and abuse notation writing $\rightarrow e$ and $e \rightarrow$ for $\rightarrow \{e\}$ and $\{e\} \rightarrow$ respectively when $e \in E$.

A. Basic Definitions

Formally, a DCR Graph is defined as follows.

Definition 1: A Dynamic Condition Response Graph (DCR Graph) G is a tuple $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, where

(i) $\mathsf{E} \subset \mathbb{E}$ is a finite set of *events*,
(ii) $\mathsf{M} \in \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$ is the *marking*,
(iii) $\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\% \subseteq \mathsf{E} \times \mathsf{E}$ is the *condition*, *response*, *milestone*, *include* and *exclude* relation respectively.
(iv) $\mathsf{L} \subset \mathbb{L}$ is the finite set of *labels* and $l : \mathsf{E} \rightarrow \mathcal{P}(\mathsf{L})$ is a labeling function mapping events to sets of labels.

As explained in the introduction, the marking (ii) represents the state of the DCR Graph and the five binary relations over the events (iii) define the constraints on the events and dynamic inclusion and exclusion. Finally, each event is mapped to a set of labels (iv). In our running example simply assign a singleton set containing a pair consisting of the name of the activity and the role able to perform the activity.

In Def. 2 we formally define when an event e of a DCR Graph is *enabled* for a marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, written $\mathsf{M} \vdash_G e$. To be enabled, the event e must be included, i.e. $e \in \mathsf{In}$, all the included events that are conditions for it must be in the set of executed events, i.e. $(\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex}$, and none of the included events that are milestones for it can be in the set of scheduled response events, i.e. $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

We then further define the new marking $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$, resulting from executing an event e in the marking M. Firstly, the event e is added to the set of executed events, i.e. $\mathsf{Ex}' = (\mathsf{Ex} \cup \{e\})$. Secondly, the event e is removed from the set of scheduled responses and all events that are a response to the event e are added, i.e. $\mathsf{Re}' = ((\mathsf{Re} \backslash \{e\}) \cup e\bullet\!\rightarrow)$. Note that if an event is a response to itself, it will be removed and immediately added again, and thus remain in the set of scheduled responses after its execution. Finally, all the events that are excluded by e are removed from the included events set, and all the events that are included by e are added, i.e. $\mathsf{In}' = (\mathsf{In} \setminus e\rightarrow\!\%) \cup e\rightarrow\!+$.

Definition 2: For a Dynamic Condition Response Graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, and $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, we define that an event $e \in \mathsf{E}$ is *enabled*, written $M \vdash_G e$, if $e \in \mathsf{In} \wedge (\mathsf{In} \cap \rightarrow\!\bullet e) \subseteq \mathsf{Ex} \wedge (\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$. The result of executing the event e in the marking M of a DCR Graph G is the marking $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e =_{def} \big(\mathsf{Ex} \cup \{e\}, (\mathsf{Re} \setminus \{e\}) \cup e\bullet\!\rightarrow, (\mathsf{In} \setminus e\rightarrow\!\%) \cup e\rightarrow\!+\big)$.

Having defined when events are enabled for execution and the effect of executing an event we define in Def. 3 the notion of finite and infinite executions and when they are accepting (or completed). Intuitively, an execution is accepting if any required, included response in any intermediate marking is eventually executed or excluded in a subsequent marking during the execution.

We further define the subset of executions in which only events that are required as responses are executed, which we refer to as *must* executions. The reason for considering must executions is that if a process is deadlock and livelock free even when restricted to must executions, then we are guaranteed progress, even if the participants in any step only perform activities that are required as responses.

Definition 3: For a Dynamic Condition Response Graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ we define an *execution* of G to be a (finite or infinite) sequence of pairs of events and labels: $\bar{e} = (e_0, a_0), (e_1, a_0), \ldots$ such that $a_i \in l(e_i)$ and $\mathsf{M}_i \vdash_G e_i$ for $\mathsf{M}_0 = \mathsf{M}$, $\mathsf{M}_{i+1} = \mathsf{M}_i \oplus_G e_i$.

Assuming $\mathsf{M}_i = (\mathsf{Ex}_i, \mathsf{In}_i, \mathsf{Re}_i)$ we say the execution $\bar{e}$ is a *must* execution if for all $(e_i, a_i) \in \bar{e}$, $e_i \in \mathsf{Re}_i$ and an *accepting* (or completed) execution if for all $(e_i, a_i) \in \bar{e}$, $\forall e \in \mathsf{In}_i \cap \mathsf{Re}_i . \exists j \geq i . e_j = e \vee e_j \rightarrow\!\% \, e$. Let $\mathsf{exe}_{\mathsf{M}}(G)$, $\mathsf{mexe}_{\mathsf{M}}(G)$, $\mathsf{acc}_{\mathsf{M}}(G)$and $\mathsf{macc}_{\mathsf{M}}(G)$ denote respectively the set of all executions, all must executions, all accepting executions, and all accepting must executions of G starting in marking M.

B. Adaptation Operations on DCR Graphs

In order to support adaptive changes for case management, we define three operations on DCR Graphs: *compose*, *change* and *discard*.

The first operation *compose* is a binary composition of two DCR Graphs, where we glue together (take the union of) the events, constraints, labels and markings of both the DCR Graphs as formally defined in Def. 4. Note that the *compose* operation also glues the markings of DCR Graphs, therefore it really defines composition of process instances.

Definition 4: Let $\mathsf{G}_i = (\mathsf{E}_i, \mathsf{M}_i, \rightarrow\!\bullet_i, \bullet\!\rightarrow_i, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$, $\mathsf{M}_i = (\mathsf{Ex}_i, \mathsf{Re}_i, \mathsf{In}_i)$ for $i \in \{1, 2\}$. Then $\mathsf{G}_1 \oplus \mathsf{G}_2 = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, where

(i) $\mathsf{E} = (\mathsf{E}_1 \cup \mathsf{E}_2)$
(ii) $\mathsf{M} = (\mathsf{Ex}_1 \cup \mathsf{Ex}_2, \mathsf{Re}_1 \cup \mathsf{Re}_2, \mathsf{In}_1 \cup \mathsf{In}_2)$,
(iii) $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ for each $\rightarrow \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$
(iv) $l(e) = l_1(e) \cup l_2(e)$ and $\mathsf{L} = \mathsf{L}_1 \cup \mathsf{L}_2$

In Def. 5 we define event substitution operation on DCR Graphs, which is used for renaming of events. We use the shorthand $e''[e \mapsto e']$ to refer to the event e' if $e'' = e$ and e'' otherwise. First, the new event is substituted in the set of events (i) and labeling function is updated accordingly (ii). Further, all the constraints sets (iii) and the sets in the marking (iv) are updated accordingly for the event substitution. Note that there are no restrictions on the new name for the event, therefore if the new name already exists in the DCR Graph, then substitution operation allows merging of events.

Definition 5: Let $\mathsf{G} = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ and $e \in \mathsf{E}, e' \in \mathbb{E}$. The *event substitution operation* is defined as $\mathsf{G}[e \mapsto e'] = (\mathsf{E}', \mathsf{M}', \rightarrow\!\bullet', \bullet\!\rightarrow', \rightarrow\!\diamond', \rightarrow\!+', \rightarrow\!\%', \mathsf{L}, l')$ where

(i) $\mathsf{E}' = \mathsf{E} \backslash \{e\} \cup \{e'\}$
(ii) $(e''[e \mapsto e'], a) \in l'$ if $(e'', a) \in l$
(iii) $\forall \rightarrow \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}. e_1[e \mapsto e'] \rightarrow' e_2[e \mapsto e']$ if $e_1 \rightarrow e_2$
(iv) $\mathsf{M}' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ and $\forall R \in \{\mathsf{Ex}, \mathsf{Re}, \mathsf{In}\}. e''[e \mapsto e'] \in R'$ if $e'' \in R$

The *change* operation can be used for renaming labels as formally defined in Def. 6. First, an event substitution operation is applied (i) and then new labels are added to the set of labels. Finally, the labeling function is updated (ii) for the new labels.

Definition 6: Let $\mathsf{G} = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ and $e \in \mathsf{E}, e' \in \mathbb{E}, A \subset \mathbb{L}$. The *change event operation* is defined as $\mathsf{G}[e \mapsto (e', A)] = (\mathsf{E}', \mathsf{M}', \rightarrow\!\bullet', \bullet\!\rightarrow', \rightarrow\!\diamond', \rightarrow\!+', \rightarrow\!\%', \mathsf{L} \cup A, l'')$ where

(i) $\mathsf{G}[e \mapsto e'] = (\mathsf{E}', \mathsf{M}', \rightarrow\!\bullet', \bullet\!\rightarrow', \rightarrow\!\diamond', \rightarrow\!+', \rightarrow\!\%', \mathsf{L}, l')$

(ii) $l''(e'') = \begin{cases} A & \text{if } e'' = e \\ l'(e'') & \text{otherwise} \end{cases}$

In the Def. 7, we introduce three overloaded versions of the *discard* operation to delete: an event from a DCR Graph (a), a constraint from a DCR Graph (b) and an event from a marking (c). Discarding an event from a DCR Graph will delete it from the set of events along with its label mapping from the labeling function (ai), additionally, it will also be removed from all the sets in the marking (aii) and finally all the constraints from and to the event will also be deleted from the respective constraints sets (aiii). Similarly, discarding a constraint from a DCR Graph will delete it from the respective constraint set (b), where as discarding an event from a set in the marking of a DCR Graph is simply removing that event from the set (c).

Definition 7: Let $\mathsf{G} = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ with $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. We define three *discard* operations by

(a) $\mathsf{G} \ominus e = (\mathsf{E}', \mathsf{M}', \rightarrow\!\bullet', \bullet\!\rightarrow', \rightarrow\!\diamond', \rightarrow\!+', \rightarrow\!\%', \mathsf{L}, l')$ where

   (i) $\mathsf{E}' = \mathsf{E} \setminus \{e\}, l' = l \setminus \{(e, l(e))\}$
   (ii) $\forall R \in \{\mathsf{Ex}, \mathsf{Re}, \mathsf{In}\}. R' = R \setminus \{e\}$
   (iii) $\forall \rightarrow \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}.$
   $\rightarrow' = \rightarrow \backslash \{(e, e'), (e', e), (e, e) \mid e' \in \mathsf{E}\}$

(b) $\mathsf{G} \ominus (e \rightarrow_c e') = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet', \bullet\!\rightarrow', \rightarrow\!\diamond', \rightarrow\!+', \rightarrow\!\%', \mathsf{L}, l')$ where $\rightarrow \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$, and $\rightarrow' = \begin{cases} \rightarrow \backslash \{(e, e')\} & \text{if } \rightarrow_c = \rightarrow \\ \rightarrow & \text{otherwise} \end{cases}$

(c) $\mathsf{G}\ominus(e, R) = (\mathsf{E}, \mathsf{M}', \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ where M' is the obtained by removing the event from a set $R \in \{\mathsf{Ex}, \mathsf{Re}, \mathsf{In}\}$ in M.

V. Safety and Liveness

A deadlock state of a DCR Graph is a marking where there is an included, required response but no enabled events. Thus, a DCR Graph is *deadlock free* if and only if for any reachable marking, there is either an enabled event or no included required responses. It is *strongly* deadlock free if and only if for any reachable marking there is either an enabled event which is also a required response or no included required responses. As exemplified below, strongly deadlock freedom guarantees progress even if the execution of the DCR Graphs is distributed (e.g. according to the different roles) and every peer only executes events that are required as responses.

Definition 8: Let $\mathcal{M}_{\mathsf{M}\rightarrow^*}(G)$ denotes the set of all reachable markings from M. For a dynamic condition response graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ we define that G is *deadlock free*, if $\forall \mathsf{M}' = (\mathsf{Ex}', \mathsf{In}', \mathsf{Re}') \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.(\exists e \in \mathsf{E}.\mathsf{M}' \vdash_G e \vee (\mathsf{In}' \cap \mathsf{Re}' = \emptyset))$. We say that G' is *strongly deadlock free*, if $\forall \mathsf{M}' = (\mathsf{Ex}', \mathsf{In}', \mathsf{Re}') \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.(\exists e \in \mathsf{Re}'.\mathsf{M}' \vdash_G e \vee (\mathsf{In}' \cap \mathsf{Re}' = \emptyset))$.

If G is the DCR Graph in Fig. 1, then G is both deadlock free and also strongly deadlock free. However, the adapted graph $G \ominus (\mathsf{prescribe\ medicine} \bullet\!\rightarrow \mathsf{sign})$ in which the response relation from prescribe medicine to sign is discarded will only be deadlock free, but not strongly deadlock free. If the doctor starts by prescribing medicine, then there will be a pending response on give medicine (but not on sign), therefore there will be no enabled event which also is a pending response. The workflow is not deadlocked since the sign activity may be executed, even though it is not required as a response. The workflow is not in an accepting state either, since there is a required response for give medicine. However, if every participant only does what is required, i.e. scheduled as a response, the workflow will never progress to a completed state. This may in particular be a problem if the execution of the workflow is distributed, e.g. according to the roles, as supported by the algorithm given in [7]. If the doctor only sees activities assigned to the doctor role, she may never sign after doing a prescription if it is not required as response. However, the nurse will be required to perform the give medicine activity, but it is not enabled since the sign must have been done first.

Note that deadlock freedom only guarantees that the process can make some progress, but not that it can proceed a long an accepting (completed) execution. A DCR Graph is defined to be *live* if and only if, in every reachable marking, it is always possible to continue along an accepting run (i.e. eventually execute or exclude any of the pending responses). We defined that it is *strongly live* if and only if, from any reachable marking there exists an accepting must execution.

Definition 9: For a dynamic condition response graph $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ we define that the DCR Graph is *live*, if $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.\mathsf{acc}_{\mathsf{M}'}(G) \neq \emptyset$, and *strongly live*, if $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.\mathsf{macc}_{\mathsf{M}'}(G) \neq \emptyset$,

The give medicine example G in Fig. 1 is again both live and strongly live, and $G \ominus (\mathsf{prescribe\ medicine} \bullet\!\rightarrow \mathsf{sign})$ will be live, but not strongly live.

VI. Verification of DCR Graphs

This sections describes how the safety and liveness properties on DCR Graphs can be verified by using the Spin [10] model checking tool. Spin supports verification of properties for asynchronous process models and distributed systems, specified in the language called PROMELA.

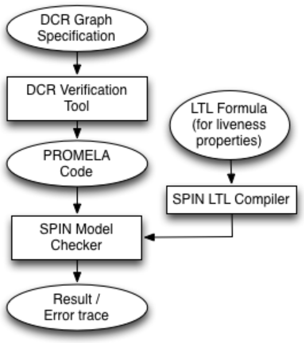


Figure 6. Verification of DCR Graphs using Spin tool

Fig. 6 shows the overall methodology of the verification of DCR Graphs using Spin. The DCR verification tool accepts a DCR Graph specification as an XML file and generates the necessary PROMELA code, compiles it and verifies it using Spin. The DCR Graph verification tool is available through a web interface [18].

The properties to be verified can be expressed as a Linear Temporal Logic (LTL) formula in the tool. The Spin LTL compiler generates a finite automaton for the negation of the formula, referred to as a *never claim*. Similarly, a finite automaton is generated for the DCR Graph model specified in PROMELA code. Finally, the Spin verifier searches for an acceptance cycle in the synchronous product of the two automata. In case the verifier finds an acceptance cycle, it reports an error by providing a trace for the property violation.

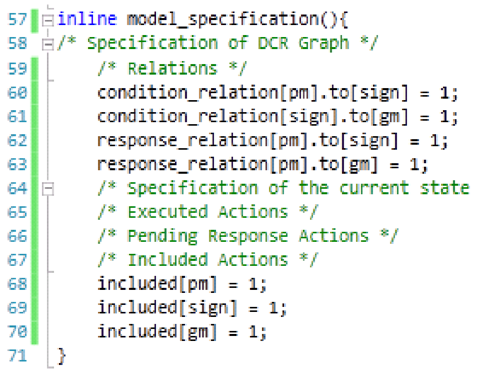
A. DCR Graphs to PROMELA

In the encoding of a DCR Graph to PROMELA, we employ the mapping from DCR Graphs to Büchi automata from [9], [19], as liveness properties are to be verified over infinite runs. The event names of a DCR Graph are mapped to integers, as PROMELA does not have support for strings. The constraint sets and marking of a DCR Graph are encoded as *arrays*, as PROMELA does not support sets. Furthermore, the language only supports fixed size arrays, therefore we have defined an event set of a DCR Graph as a bit array, where the index of an array represents the integer code of an event and the value (0 or 1) at that index defines whether the event is part of the set or not. The marking (M) of a DCR Graph is encoded as three bit arrays.

The PROMELA language supports one-dimensional arrays only. Therefore the constraints sets of a DCR Graph are defined by using typedef for two-dimensional arrays where the indices of the array are the integer codes of events and the values (0 or 1) represents whether the constraint exists from the first to the second event as shown in Fig. 7.

The assignment *included[2] = 1;* defines that the event *gm* (with integer code = 2) is part of the *included* set. Since all data types of PROMELA are initialized to 0 by default, all the events which are not explicitly mentioned in the initial marking or specification are not included.

PROMELA does not have procedure/function construct to structure the code, therefore the *inline* construct was used to group a sequence of statements related to one logical function as shown in Fig. 7.



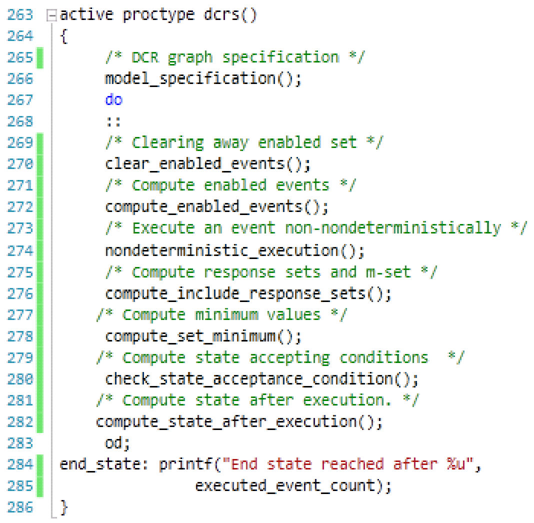
```
inline model_specification(){
/* Specification of DCR Graph */
    /* Relations */
    condition_relation[pm].to[sign] = 1;
    condition_relation[sign].to[gm] = 1;
    response_relation[pm].to[sign] = 1;
    response_relation[pm].to[gm] = 1;
    /* Specification of the current state
    /* Executed Actions */
    /* Pending Response Actions */
    /* Included Actions */
    included[pm] = 1;
    included[sign] = 1;
    included[gm] = 1;
}
```

Figure 7. Specification of give medicine example

The main logic for verification of safety and liveness properties is shown in Fig. 8. The main process function (*proctype dcrs*) contains one *do* loop in which code from different *inline* blocks will be executed.

The *model_specification()* inline block contains the specification of a DCR Graph as described previously and the *clear_enabled_events* block clears the list of events in the enabled set. The next inline block is *Compute_enabled_events*, which loops through the events in the *included* set and verifies whether all its included condition events have been executed. Similarly, all included milestone events of an event are also checked to make sure that none of them are part of the pending *response* set. An event satisfying these checks will be added to the *enabled* events set.

The next inline block, *nondeterministic_execution()*, contains code for executing one of the events from the *enabled* set. The tool generates options for an *if* block with a guard matching to status bit of an event in the *enabled* set. During verification, Spin will evaluate all the guards and choose one of the enabled options non-deterministically, by assigning it to the variable *random_event_executed*.



```
active proctype dcrs()
{
    /* DCR graph specification */
    model_specification();
    do
    ::
    /* Clearing away enabled set */
    clear_enabled_events();
    /* Compute enabled events */
    compute_enabled_events();
    /* Execute an event non-nondeterministically */
    nondeterministic_execution();
    /* Compute response sets and m-set */
    compute_include_response_sets();
   /* Compute minimum values */
    compute_set_minimum();
   /* Compute state accepting conditions  */
    check_state_acceptance_condition();
   /* Compute state after execution. */
   compute_state_after_execution();
    od;
end_state: printf("End state reached after %u",
                executed_event_count);
}
```

Figure 8. PROMELA code for main process

B. Deadlock and Liveness

In *nondeterministic_execution()*, if none of the guards are evaluated to true, then the else block will be executed. The code in the else blocks declares a deadlock if there are any included pending responses. In the absence of included pending responses, the program jumps to *end_state* to terminate the program.

In the case of enabled events in every marking, the *else* block will never get executed and thereby the *do* loop will continue forever without breaking. Spin detects such kind of cycles and terminates the program after inspecting all the states of the automaton.

Liveness properties of a DCR Graph can be verified by specifying a *never claim* in LTL. In the tool, liveness verification is done by specifying a correctness claim as $\Box\Diamond$ *accepting_state_visited* in LTL, from which Spin generates a *never claim* based on the negation of the formula.

C. Strong Deadlock freedom and Liveness

The encoding of deadlock for must executions is very much similar to that of the deadlock property introduced in the previous paragraphs. In the *nondeterministic_execution*, an additional check for *included* and *enabled* pending responses will be made, before choosing any enabled event for non-deterministic execution. In case of existence of a pending response without any *enabled* pending response, a violation of strongly deadlock freedom will be declared.

For verification of the strong liveness property on a DCR Graph, we generate an encoding for every possible reachable marking of the DCR Graph. In addition a check will be made in the non-deterministic execution of events, to make sure that the enabled events are also pending responses.

D. *Evaluation of Spin Verification*

Table. I shows statistics for the Spin verification of the healthcare workflow from the previous sections. The second and third columns represent the number of events and constraints in the DCR Graph. The number of reachable markings in the DCR Graph is shown in column 4. The last three columns display the statistics from Spin verification: the number of Spin program states, time taken in seconds and memory usage in megabytes respectively.

Model	DCR Graph			Spin statistics		
	E	→	states	program states	time sec.	memory MB.
prescribe medicine	3	4	13	14,741	0.04	613.04
order tests	5	7	72	231,731	0.60	759.70
prescribe + order	6	11	116	602,289	1.67	775.20
adapted prescribe + order	7	11	460	3,267,596	9.37	880.70
create case [5]	17	28	1386	15,614,513	63.1	1432.5

Table I
SPIN VERIFICATION STATISTICS

Even though Spin verification on DCR Graphs is quite useful, we have noticed certain drawbacks. First of all, the number of Spin program states grows exponentially with the number of events in a DCR Graph. For example, the *adapted prescribe medicine* example from Fig. 5 contains 7 events and 11 constraints. But as shown in Table. I, the Spin program states are more than three million, even though there are only 460 unique reachable markings in the büchi automaton for the DCR Graph. The automata construction is inherently exponential, however, a further blow-up of Spin program states is caused by the lack of good data structures in PROMELA for encoding sets and other complex types. This means that the event sets of a DCR Graphs have to be encoded as fixed size arrays and these arrays have to be iterated many times to calculate the updated markings. Moreover, every value change of a variable (e.g. loop index) is considered as unique program state in Spin. Additionally, the increase of Spin memory usage (last column) is also an alarming issue, which could be problematic in verification of larger models.

In addition to the above limitations, the output generated by Spin is also not user friendly. Especially, it is difficult for modellers to figure out the counter example from the Spin error trails. Therefore, we strongly believe that by performing the verification directly on the reachable markings of a DCR Graph, it is possible to verify much larger models and provide intuitive validation results.

VII. Conclusion

We have presented an approach to adaptive case management based on the recently introduced declarative process model Dynamic Condition Response (DCR) Graphs. Our work leverages three key features of DCR Graphs: 1) Its declarative nature with implicit definition of states allow for simple definitions of process composition and change, 2) its simple operational semantics based on markings of the graph allowed us to extend the definitions of process composition and change to running instances, and 3) the mapping of DCR Graphs to the SPIN model checking tool allowed us to formally verify deadlock freedom and liveness for the dynamically changed adapted models.

The definition of deadlocks for DCR Graphs is new, and exploited that the markings of DCR Graphs allow to distinguish between which events may happen now (the enabled events), and which events must eventually happen (the required future responses). This allowed us to define a deadlock as a state where some event must eventually happen, but no events may happen now. Moreover, we introduced a new notion of *strongly* deadlock freedom, which intuitively means that even in a situation where every actor only perform required actions there will be no deadlocks. We also introduced the notion of liveness and strong liveness for DCR Graphs

We found that the PROMELA language and its ability to verify both safety and liveness properties made SPIN easy to use as back-end verification tool for DCR Graphs and benchmarked the verification on a small set of examples. However, the benchmarks also show that the resulting SPIN models reach a quite large number of states for even small DCR Graphs, which indicate that there may be an advantage to implement the verification algorithms directly for DCR Graphs. This could potentially explore the partial order information of DCR Graphs and avoid constructing the interleaved transition system model.

In future work we plan to investigate a native implementation of model checking for DCR Graphs. We also plan to extend the approach to ACM presented in the present paper to DCR Graphs extended with data and nested sub graphs as defined in [17] and relate and compare our work to the GSM-approach [12].

References

[1] Hanna Eberle, Tobias Unger, and Frank Leymann. Process fragments. In *OTM '09*, pages 398–405. Springer-Verlag, 2009. 3

[2] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM. 1, 3, 4

[3] Wei-Dong Zhu et. al. *Advanced Case Management with IBM Case Manager*. IBM Redbooks, 2013. http://www.redbooks.ibm.com/redbooks/pdfs/sg247929.pdf. 4

[4] Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, volume 69 of *EPTCS*, pages 59–73, 2011. 1, 4

[5] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proceedings of IEEE International EDOC Conference*, 2011. 9

[6] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. 1

[7] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Safe distribution of declarative processes. In *9th International Conference on Software Engineering and Formal Methods (SEFM) 2011*, 2011. 1, 3, 7

[8] Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Modular context-sensitive and aspect-oriented processes with dynamic condition response graphs. In *Foundations of Aspect-Oriented Languages 2013*, 2013. 1, 4

[9] Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, pages 59–73, 2010. 7

[10] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23:279–295, May 1997. 2, 7

[11] Gerard J. Holzmann. *SPIN Model Checker, The: Primer and Reference Manual*. Addison-Wesley Professional, 2004. 2

[12] Richard Hull. Formal study of business entities with lifecycles: Use cases, abstract models, and results. In Tevfik Bravetti, Mario; Bultan, editor, *7th International Workshop on Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, 2010. 1, 4, 9

[13] Wil Janssen, Radu Mateescu, Sjouke Mauw, and Jan Springintveld. Verifying business processes using spin. In *Proceedings of the 4th International SPIN Workshop*, pages 21–36, 1998. 4

[14] Jana Koehler, Joerg Hofstetter, and Roland Woodtly. Capabilities and levels of maturity in it-based case management. In *Business Process Mangement (BPM)*, LNCS. Springer Verlag, 2012. 1

[15] Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Process-oriented information systems in healthcare (ProHealth 08)*, pages 36–43. BPM 2008 Workshops, 2008. 1

[16] K.M. Lyng. Clinical guidelines in everyday praxis, implications for computerization. *Journal of Systems and Information Technology*, 2009. 1

[17] Raghava Rao Mukkamala. *A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, June 2012. http://www.itu.dk/people/rao/phd-thesis/DCRGraphs-rao-PhD-thesis.pdf. 1, 2, 5, 9

[18] Raghava Rao Mukkamala. Formal verification of dcr graphs using spin. http://trustcare.itu.dk/dcrgraphs-verification/verificationWebUI.aspx, 2012. 2, 7

[19] Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010. 1, 2, 3, 4, 7

[20] Nicolas Mundbrod, Jens Kolb, and Manfred Reichert. Towards a system support of collaborative knowledge work. In *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, LNBIP. Springer, September 2012. 2

[21] Object Management Group (OMG). Case management model and notation (cmmn). http://www.omg.org/spec/CMMN/, January 2013. 4

[22] Manfred Reichert and Peter Dadam. A framework for dynamic changes in workflow management systems. In *Conference on Database and Expert Systems Applications*, 1997. 3, 4

[23] I. Rychkova and S. Nurcan. Towards adaptability and control for knowledge-intensive business processes: Declarative configurable process specifications. In *Hawaii International Conference on System Sciences*, 2011. 1, 4

[24] Irina Rychkova. Towards automated support for case management processes with declarative configurable specifications. In *BPM Workshops*. Springer Berlin Heidelberg, 2013. 1, 4

[25] Tijs Slaats. Dcr graphs editor. http://www.itu.dk/research/models/wiki/index.php/DCR_Graphs_Editor, February 2013. 1

[26] Tijs Slaats, Raghava Rao Mukkamala, Thomas Hildebrandt, and Morten Marquard. Exformatics declarative case management workflows as dcr graphs. In *International Conference on Business Process Management (BPM2013)*, 2013. 1

[27] Keith D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press, 2010. 1, 2

[28] KeithD. Swenson. Position: Bpmn is incompatible with acm. In Marcello Rosa and Pnina Soffer, editors, *BPM Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 55–58. Springer Berlin Heidelberg, 2013. 4

[29] W. M. P. Van Der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317, September 2001. 1, 3, 4

[30] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. 4

[31] Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006. 1, 4

Modular Context-Sensitive and Aspect-Oriented Processes with Dynamic Condition Response Graphs

Thomas Hildebrandt[1] Raghava Rao Mukkamala[1] Tijs Slaats[1,2] Francesco Zanitti[1]

[1] IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
[2] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark
{rao, hilde, tslaats, zanitti}@itu.dk

Abstract

We propose the recently introduced declarative and event-based Dynamic Condition Response (DCR) Graphs process model as a formal basis for modular implementation of context-sensitive and aspect-oriented processes. The proposal is supported by a new join operator allowing modular composition and refinement of DCR Graphs. We give small illustrative examples of DCR Graphs defining context-sensitive processes where context-events dynamically enable and disable the need for authentication and the join operator is used to add authentication to a process. Finally, we discuss the use of formal verification to ensure that processes satisfy safety and liveness properties, and define two liveness properties (deadlock freedom and liveness) that can be verified directly on the state graph for DCR Graphs.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory - Semantics, Syntax

General Terms Design, Languages, Reliability, Verification

1. Introduction

The terms *context-sensitivity*, *context-awareness* and *context-dependency* are generally used to describe systems that adapt their behavior according to changes in their context. Changes in the context are naturally described as events, either provided by sensors, user inputs or messages from other programs. This suggests the application of *event-driven programming* to implement context-aware systems. Event-driven systems are normally based on the publish-subscribe pattern. That is, at any time the system subscribes to specific (patterns of) events. Whenever a pattern has been detected, the system reacts by performing a block of code, possibly publishing new events.

In the present paper we propose to use the recently introduced event-based declarative process model, *Dynamic Condition Response Graphs* (DCR Graphs) [3, 6, 7, 13], as a formal basis for modular and aspect-oriented construction of context-sensitive systems. The modular and aspect-oriented construction is facilitated by a new, general *join* operator. We give small examples that illustrate how the join operator can be used to *merge* processes, and discuss how merging is similar to adding advices and weaving of aspects in AOP. The examples also show how the join operator can be used to *refine* and more generally *adapt* behavior of a process, i.e. by replacing an event or a complete sub process with a new process.

It should be stressed that the proposal in the present paper is very initial work. The DCR Graph model was developed with another application area in mind as a foundation for flexible workflow languages as part of the second author's PhD project [12]. It has then subsequently been continued in the (ongoing) PhD projects of the third and fourth authors focussing on respectively developing a formal foundation for the implementation of flexible, cross-organizational workflow systems [5] and developing a process-oriented event-based programming language (PEPL) [4] for context-sensitive services based on DCR Graphs.

In Fig. 1 below we give a concrete example of a context-sensitive authorization process that we will use as running example. The example is inspired by a concrete case study of an oncology workflow process at a danish hospital [11].

The graph consists of five *events* (shown as boxes). Each event corresponds to the (possibly repeated) execution of an activity, which is indicated by a label assigned to the event and shown inside the box. The activities in the example are thus action, authorize, reauthorize, normal and emergency.

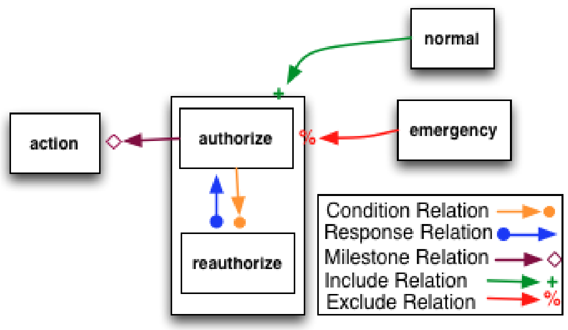


Figure 1. Context-sensitive authorization process as DCR Graph.

One way to think of a DCR Graph is as an event-driven reactive program, where the code-blocks usually executed when an event pattern is detected have been replaced by specifications of *response* events, specifying activities that must eventually be executed (by the process or the environment) whenever *possible*. That is, the occurrence of an event *schedules* other events that must be executed in the future in order for the process to progress. The response events are specified in the graph by the response relation, indicated graphically by the arrow with a bullet at the source and coloured blue to make it more easily visible. In the example we thus have a response relation from the reauthorize to the authorize event.

Moreover, every event can have a list of *condition* events and *milestone* events. In order for an event to be enabled, i.e. its activity to be executable, its condition events must have been executed at least once in the past and its milestone events must not currently be scheduled for execution. In the example, the authorize event is a condition for the reauthorize event and a milestone for the action event. The condition relation means in this example that reauthorize can not be executed if authorize has not been executed at least once in the past. The milestone relation means that the action event is blocked if an authorize event is scheduled for execution.

Finally, in addition to the specification of response, condition and milestone events, a novel idea of DCR Graphs is the specification of events that are *excluded* and *included* when an event happen. Exclusion and inclusion of events have a cross-cutting effect on the system similar to turning aspects on and off in aspect oriented programming: An excluded event is ignored as condition and milestone of events and even if it is currently scheduled it will not be required to be executed unless it is included again.

In the example, the event emergency excludes the authorize and reauthorize events and the event normal includes the two events. Note we use the nested DCR Graphs notation introduced in [6], a relation to a box around events is simply a short hand for having the relation to all the sub events. Now, if an emergency event happens (e.g. representing that the condition of the patient becomes critical), then the events authorize and reauthorize are excluded. This means that even if authorize is scheduled for execution and a milestone for action, then it is not disabling the action event. However, if the normal event happens, then authorize and reauthorize are again included.

The five relations (condition, response, milestone, include and exclude) completely describe the dynamic behavior of the process. Moreover, the state of a running process can be described by a triple of finite sets of events, $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ similar to markings of Petri Nets [15, 16]. The set Ex records the previously executed events, the set Re describes the events scheduled for execution, and the set In denotes the currently included events. The set Re thus contains events schedules as responses to events that have been executed, but we also allow to define that some events are scheduled in the initial marking of the graph.

We graphically visualize a marking $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ by adding a (green) checkmark to every box for events in the Ex set, a (red) exclamation mark to every box for events in the Re set, and making the border dashed of boxes for events which are *not* in the In set. A *run* of a DCR Graph is then a (possibly infinite) sequence of markings, where the $(n + 1)$th marking is obtained by executing one of the enabled events in the nth marking (adding it to the set Ex) and updating the Re and In sets according to the relations.

An example run of the process in Fig. 1 is shown in Fig. 2. In the initial marking (at the top), nothing has been executed, the events action and authorize are scheduled, and every event is included. This state is described by the marking $(\emptyset, \{\mathsf{action}, \mathsf{authorize}\}, \mathsf{E})$, where E is the set of all five events. In this state, only the events authorize, normal and emergency are enabled. The event action is blocked because it has authorize as milestone and authorize is scheduled for execution. The event reauthorize is blocked because it has authorize as condition, and this event has never been executed.

Now, if the emergency event is executed then the state changes to the marking $(\{\mathsf{emergency}\}, \{\mathsf{action}, \mathsf{authorize}\}, \mathsf{E}\backslash\{\mathsf{authorize}, \mathsf{reauthorize}\})$, shown at the graph in the middle of Fig. 2. That is, emergency is recorded as executed, whereas action and authorize are still scheduled, but authorize and reauthorize are excluded. This implies that authorize is not longer considered as milestone for action, which therefore is enabled. The marking shown at the bottom of Fig. 2 shows the state if the event action is executed.

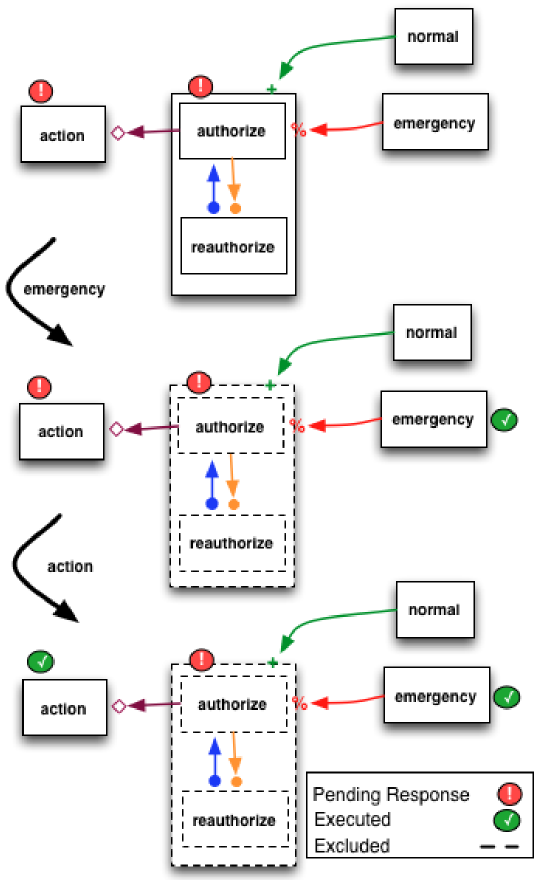


Figure 2. An example run of the authorization process.

Note that action can be executed several times. However, if normal is now executed, i.e. if the patient condition is no longer critical, then system moves to the marking shown in Fig. 3, where authorize and reauthorize are included again.

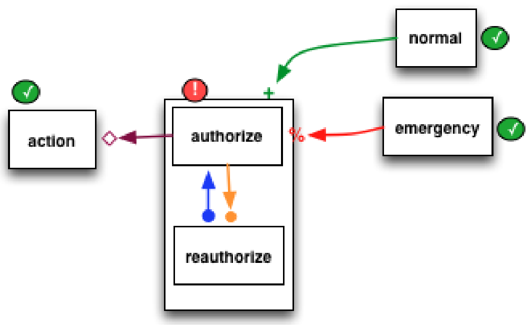


Figure 3. If the normal event is executed, then authorize is again required before action can be executed.

Since authorize is still scheduled in the marking in Fig. 3, the event action is not enabled and can not be executed unless authorize is executed or excluded because the event emergency is executed again.

Runs may be finite or infinite. We say that a run is *accepting* if whenever an event is scheduled, it eventually gets excluded or executed. As shown in [12, 13], DCR Graphs can be mapped to the standard Büchi-automata model, characterizing all runs as well as fair runs as the accepting runs, and subsequently verified for safety and liveness properties using the SPIN model checker [9]. However, it can also be represented directly (and more compactly) as a so-called *transition system with responses* [2], which is simply a labelled transition system where each state is annotated by a set of labels, referred to as the response actions. The accepting runs of a transition system with responses is then the finite or infinite runs where whenever a label is included in the response set of an intermediate state in the run, it will be excluded from the response set in a subsequent state or executed.

It is worth stressing that there is *no* explicit sequencing of commands in the DCR Graphs process model. This makes it possible to weave, refine and adapt processes by the general join operator introduced in Sec. 3. If a new event is joined as a milestone for some existing events in a process, and such that this new event is scheduled whenever the existing events are scheduled as responses, then the new event must be executed before the existing events become enabled. Thus, the new event is similar to an *advice* in AOP, that must be executed before the targetted set of existing events which correspond to *join points*.

After briefly recalling the formal definition of DCR Graphs in Sec. 2, we exemplify this aspect-oriented modularity in Sec. 3 where the DCR Graph shown in Fig. 1 is joined with a another process describing a context-sensitive request/reply process shown in Fig. 4

We end by briefly summarizing the techniques for formal verification and distribution of DCR Graphs developed so-far in the above mentioned research projects and briefly touch on related work.

2. DCR Graphs

In this section we give the formal definitions of DCR Graphs. We employ the following notation.

Notation: For a set E we write $\mathcal{P}(E)$ for the power set of E (i.e. set of all subsets of E). For a binary relation $\rightarrow \subseteq E \times E$ and a subset $\xi \subseteq E$ we write $\rightarrow \xi$ and $\xi \rightarrow$ for the set $\{e \in E \mid (\exists e' \in \xi \mid e \rightarrow e')\}$ and the set $\{e \in E \mid (\exists e' \in \xi \mid e' \rightarrow e)\}$ respectively, and abuse notation writing $\rightarrow e$ and $e \rightarrow$ for $\rightarrow \{e\}$ and $\{e\} \rightarrow$ respectively when $e \in E$.

Formally, a DCR Graph is defined as follows.

Definition 1. *A Dynamic Condition Response Graph (DCR Graph) G is a tuple* $(\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, where*

(i) E *is a set of* events*,*
(ii) $\mathsf{M} \in isthe$marking$, \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$*,*
(iii) $\rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\% \subseteq \mathsf{E} \times \mathsf{E}$ *is the* condition, response, milestone, include *and* exclude *relation respectively.*
(iv) L *is the set of* labels *and* $l : \mathsf{E} \rightarrow \mathcal{P}(\mathsf{L})$ *is a labeling function mapping events to labels.*

As explained in the introduction, the marking (ii) represents the state of the DCR Graph and the five binary relations over the events (iii) define the constraints on the events and dynamic inclusion and exclusion. Finally, each event is mapped to a set of labels (iv).

In Def. 2 we formally define that an event e of a DCR Graph with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ is *enabled*, written $G \vdash e$, when e is included in the current marking, i.e. $e \in \mathsf{In}$, and all the included events that are conditions for it are in the set of executed events, i.e. $(\mathsf{In} \cap \rightarrow\!\bullet\, e) \subseteq \mathsf{Ex}$, and none of the included events that are milestones for it are in the set of scheduled response events, i.e. $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$. We then further define the change of the marking when an enabled event e is executed: Firstly, the event e is added to the set of executed events $(\mathsf{Ex} \cup \{e\})$. Secondly, the event is removed from the set of scheduled responses and all events that are a response to the event e are added to the set of scheduled responses $((\mathsf{Re} \setminus \{e\}) \cup e \bullet\!\!\rightarrow)$. Note that if an event is a response to itself, it will remain in the set of scheduled responses after its execution. Finally, the included events set is updated to the set $(\mathsf{In} \setminus e \rightarrow\!\%) \cup e \rightarrow\!+$, i.e. all the events that are excluded by e are removed, and then all the events that are included by e are added.

Definition 2. *For a Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*, and* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *we define that an event* $e \in \mathsf{E}$ *is* enabled*, written* $G \vdash e$*, if* $e \in \mathsf{In} \wedge (\mathsf{In} \cap \rightarrow\!\bullet\, e) \subseteq \mathsf{Ex}$ *and* $(\mathsf{In} \cap \rightarrow\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$*. We further define the result of executing the event e as* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In}) \oplus_G e =_{def} \big(\mathsf{Ex} \cup \{e\}, (\mathsf{Re} \setminus \{e\}) \cup e \bullet\!\!\rightarrow, (\mathsf{In} \setminus e \rightarrow\!\%) \cup e \rightarrow\!+\big)$.

Having defined when events are enabled for execution and the effect of executing an event we define in Def. 3 the notion of finite and infinite executions and when they are accepting. Intuitively, an execution is accepting if any event which is scheduled and included in any intermediate marking, is eventually executed or excluded.

Definition 3. *For a Dynamic Condition Response Graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *we define an* execution *of G to be a (finite or infinite) sequence of tuples* $\{(\mathsf{M}_i, e_i, a_i, \mathsf{M}'_i)\}_{i \in [k]}$ *each consisting of a marking, an event, a label and another marking (the result of executing the event) such that* $\mathsf{M} = \mathsf{M}_0$ *and* $\forall i \in [k].a_i \in l(e_i) \wedge G_i \vdash e_i \wedge \mathsf{M}'_i = \mathsf{M}_i \oplus_G e_i$ *and* $\forall i \in [k-1].\mathsf{M}'_i = \mathsf{M}_{i+1}$*, where* $G_i = (\mathsf{E}, \mathsf{M}_i, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$*. We say the execution is* accepting *if* $\forall i \in [k].(\forall e \in \mathsf{In}_i \cap \mathsf{Re}_i.\exists j \geq i.e_j = e \vee e \notin \mathsf{In}'_j))$*, where* $\mathsf{M}_i = (\mathsf{Ex}_i, \mathsf{In}_i, \mathsf{Re}_i)$ *and* $\mathsf{M}'_j = (\mathsf{Ex}'_j, \mathsf{In}'_j, \mathsf{Re}'_j)$*. Let* $\mathsf{exe}_{\mathsf{M}}(G)$ *and* $\mathsf{acc}_{\mathsf{M}}(G)$ *denote respectively the set of all executions and all accepting executions of G starting in marking* M*. Finally we say that a marking* M' *is reachable in G (from the marking* M*) if there exists a finite execution ending in* M' *and let* $\mathcal{M}_{\mathsf{M}\rightarrow^*}(G)$ *denote the set of all reachable markings from* M.

A marking in a DCR Graph is accepting, if there are no included scheduled events that required as responses. Thus, a deadlock state can be defined as a state where there is an included scheduled event, but without any enabled events. We say that a DCR Graph is *deadlock free* if and only if there is no reachable deadlock state.

Definition 4. *For a dynamic condition response graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *we define that G is* deadlock free*, if* $\forall \mathsf{M}' = (\mathsf{Ex}', \mathsf{In}', \mathsf{Re}') \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.(\exists e \in \mathsf{E}.G' \vdash e \vee (\mathsf{In}' \cap \mathsf{Re}' = \emptyset))$*, for* $G' = (\mathsf{E}, \mathsf{M}', \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$.

A DCR Graph is defined to be *live* if and only if, in every reachable marking, it is always possible to continue along an accepting run.

Definition 5. *For a dynamic condition response graph* $G = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$ *we define that the DCR Graph is* live*, if* $\forall \mathsf{M}' \in \mathcal{M}_{\mathsf{M}\rightarrow^*}.\mathsf{acc}_{\mathsf{M}'}(G) \neq \emptyset$.

3. Process Composition

In this section we define a new general join operation on DCR Graphs which supports modular, aspect-oriented composition and refinement of DCR Graphs. The join operation is defined relative to two *join* relations $\lhd$ and $\rhd$, which specify which events in the left (right) graph are replaced by events in the right (left) graph.

In particular the join operation allows in special cases for basic union of graphs and refinement of events, that is, substituting an event by an entire new sub graph.

Before giving the formal definition, we will give an example of how to join a process graph with another, where one event in the former graph is refined by two events in the latter process.

The DCR Graph G_{rr} in Fig. 4 below shows a process for a context-sensitive request/reply pattern.

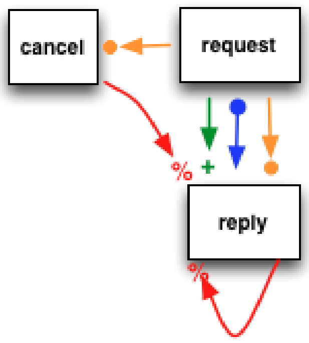


Figure 4. A context-sensitive request/reply process

If a request event happens, the reply is scheduled as response. The reply event is excluded if it is executed, or a cancel event subsequently happens. However, it will be included (and scheduled) again if a new request event happens. As for the example given in the introduction, the cancel event may be triggered by some change of conditions in the context of the process.

We can now use the join operation to define a context-sensitive request/reply process where the reply needs (context-sensitive) authorization as defined in the DCR Graph G_{auth} given in Fig. 1 in the introduction.

The intention is to merge the two process graphs, but such that the `reply` event in G_{rr} is replaced by the `authorize` and `action` events in G_{auth}. The result is the graph shown in Fig. 5 and formally defined as $G_{rrauth} = G_{rr} \lhd\rhd G_{auth}$ for $\lhd = \{(\texttt{reply}, \{\texttt{authorize}, \texttt{action}\})\}$ and $\rhd = \emptyset$.

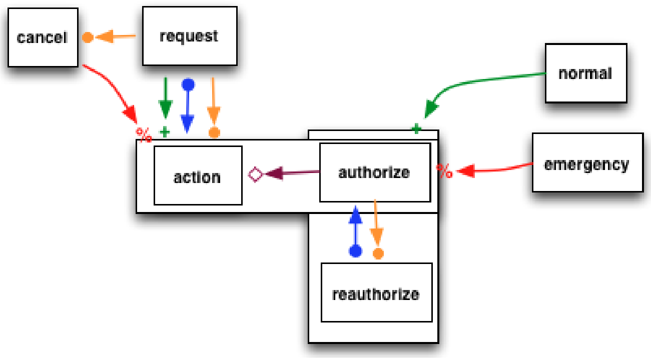


Figure 5. Joining context-sensitive request/reply and authorization processes

Informally, the two event sets have been joined, replacing the `reply` event in the graph G_{rr} with the two events `authorize` and `action` in the G_{auth} graph, while inheriting all relations between the replaced event and other events in G_{rr}. Recall, that relations to the box around `authorize` and `action` is merely a convinient way to represent relations to both `authorize` and `action`.

To ease readability we may adopt a programming language notation for the join operation, writing the above join as follows.

Listing 1. Using Join to Authorize a Reply

```
CSRequestReplyAuth =
join CSRequestReply and CSAuthorization
where reply < {authorize , action}
```

Below we give the formal definition of the join operator.

DEFINITION 6. *Assume DCR Graphs* $\mathsf{G}_i = (\mathsf{E}_i, \mathsf{M}_i, \rightarrow\!\bullet_i, \bullet\!\rightarrow_i, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$ *where* $\mathsf{M}_i = (\mathsf{Ex}_i, \mathsf{Re}_i, \mathsf{In}_i)$ *for* $i \in \{1,2\}$ *and join relations* $\lhd\colon \mathsf{E}_1 \rightharpoonup \mathcal{P}(\mathsf{E}_2)$ *and* $\rhd\colon \mathsf{E}_2 \rightharpoonup \mathcal{P}(\mathsf{E}_1)$. *The join of* G_1 *and* G_2 *relative to* $\lhd$ *and* $\rhd$ *is defined as* $\mathsf{G}_1 \lhd\rhd \mathsf{G}_2 = (\mathsf{E}, \mathsf{M}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%, \mathsf{L}, l)$, *where*

(i) $\forall e \in \mathsf{dom}(\lhd).\ \lhd(e) \cap \mathsf{dom}(\rhd) = \emptyset$ *and* $\forall e \in \mathsf{dom}(\rhd).\ \rhd(e) \cap \mathsf{dom}(\lhd) = \emptyset$

(ii) $\mathsf{E} = \mathsf{E}'_1 \cup \mathsf{E}'_2$ *for* $\mathsf{E}'_1 = \mathsf{E}_1 \backslash \mathsf{dom}(\lhd)$ *and* $\mathsf{E}'_2 = \mathsf{E}_2 \backslash \mathsf{dom}(\rhd)$

(iii) $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, *where:* $\mathsf{Ex} = \mathsf{Ex}_1 \cap \mathsf{E}'_1 \cup \mathsf{Ex}_2 \cap \mathsf{E}'_2$, $\mathsf{In} = \mathsf{In}_1 \cap \mathsf{E}'_1 \cup \mathsf{In}_2 \cap \mathsf{E}'_2$ *and* $\mathsf{Re} = \mathsf{Re}_1 \cap \mathsf{E}'_1 \cup \mathsf{Re}_2 \cap \mathsf{E}'_2$,

(iv) $\rightarrow = \unlhd^{-1}\!\rightarrow_1 \cup \rightarrow_1\!\unlhd \cup \unrhd^{-1}\!\rightarrow_2 \cup \rightarrow_2\!\unrhd$ *for each* $\rightarrow \in \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$ *and* $\unlhd = \{(e, e') \mid e \in E_1 \wedge e' \in \lhd(e) \cup \{e\}\}$ *and* $\unrhd = \{(e, e') \mid e \in E_2 \wedge e' \in \rhd(e) \cup \{e\}\}$

(v) $\mathsf{L} = \mathsf{L}_1 \cup \mathsf{L}_2$

(vi) $l(e) = \begin{cases} l_1(e) \cup l_2(e) & \textit{if } e \in \mathsf{E}'_1 \cap \mathsf{E}'_2 \\ l_1(e) & \textit{if } e \in \mathsf{E}'_1 \backslash \mathsf{E}'_2 \\ l_2(e) & \textit{if } e \in \mathsf{E}'_2 \backslash \mathsf{E}'_1 \end{cases}$

The first condition, (i), guarantees that there are no circular refinements. That is, an event in G_1 can not be refined by an event in G_2 which is also refined by an event in G_1 and vice versa. The second condition, (ii), states that the set of events in the joined graph consists of all the events of the two graphs, that have not been refined by events in the other graph. The third condition, (iii), defines the join of the markings of the graphs. It simply inherits the markings from the two graphs. Note that the fact that markings are also joined means that we can also apply the join operator on computing processes. Condition (iv) defines the extension of the relations to the refining events. The relation $\unlhd$ is the reflexive closure of the left join $\lhd$ relation, i.e. $e \unlhd e'$ if and only if $e \lhd e'$ or $e = e'$. Similarly, $\unrhd$ is the reflexive closure of $\rhd$. The relation $\unlhd^{-1}\!\rightarrow_1$ is then the relational composition of $\unlhd^{-1}$ (the inverse of $\unlhd$) and the relation $\rightarrow_1$, and similarly the relation $\rightarrow_1\!\unlhd$ is the relational composition of $\rightarrow_1$ and $\unlhd$. That is, $e(\unlhd^{-1}\!\rightarrow_1 \cup \rightarrow_1\!\unlhd)e'$ if

- $e \rightarrow_1 e'$, or
- $e \rightarrow_1 e'''$ and $e''' \lhd e'$, or
- $e'' \lhd e$ and $e'' \rightarrow_1 e'$.

Considering our example in Fig. 5, this definition implies that e.g. `cancel` will exclude both `action` and `authorize`. But note that the self-exclude relation on `reply`, enforcing the "linearity" constraint that only one reply can only be carried out for each request, is not kept. This is because relations between events that are replaced, and thus in particular self-relations are not kept. The rationale for not keeping these relations are that a join should allow for removing constraints on the refined events. To keep the property that the action can only be carried out once for each request, the refining graph G_{auth} should have an exclude relation from `action` to itself.

Finally, conditions (v) and (vi) define the label set as the union of the two label sets, and the labeling of events by the original label for events that are not shared and the union of the label set for shared events. Since we have not used the label function in the present paper it can safely be ignored. For the curious reader, the labeling function allows to have distinct events with the same "external" label, which is useful for some practical applications, as well as for proving that every Büchi-automaton can be represented by a DCR Graph. This means in particular that the DCR Graphsmodel is more expressive that LTL.

As indicated in the beginning of the section, we may derive operations $G \backslash e$, $G[e \lhd G']$ and $G \cup G'$ for respectively discarding an event e, refining an event e by G' and taking the union of two graphs G and G' as special cases of the join operator.

DEFINITION 7. *For a DCR Graph* $\mathsf{G} = (\mathsf{E}_1, \mathsf{M}_i, \rightarrow\!\bullet_i, \bullet\!\rightarrow_i, \rightarrow\!\diamond_i, \rightarrow\!+_i, \rightarrow\!\%_i, \mathsf{L}_i, l_i)$ *for* $i \in \{1, 2\}$ *and* $e \in \mathsf{E}_1$ *define*

- *(Discard)* $G_1 \backslash e = G_1 \lhd\!\rhd G_\emptyset$ *where* $\lhd = (e, \emptyset)$ *and* $G_\emptyset$ *is the empty DCR Graph, i.e. the DCR Graph with no events,*
- *(Refine)* $G_1[e \lhd G_2] = G_1 \lhd\!\rhd G_2$ *where* $\lhd = (e, E_2)$ *and* $\rhd = \emptyset$*,*
- *(Union)* $G_1 \cup G_2 = G_1 \lhd\!\rhd G_2$ *where* $\lhd = \emptyset$ *and* $\rhd = \emptyset$*.*

As an example of the discard operation, we may remove the ability to cancel requests in the graph G_{rrauth} in Fig. 5 by discarding the `cancel` event, taking the graph $G_{rrauth} \backslash \texttt{cancel}$.

As an example of the refine operation, we may add the "linearity" constraint to `action` in G_{rrauth}, i.e. that it can be executed at most once for every request. This is done with a refine operation $G_{rrauth}[\texttt{action} \lhd G_{linact}]$, where G_{linact} is the DCR Graph $(\{\texttt{action}\}, (\emptyset, \emptyset, \emptyset), \emptyset, \emptyset, \emptyset, \emptyset, (\texttt{action}, \texttt{action}), \emptyset, \emptyset)$, i.e. the graph with an empty marking and a single event `action` related to itself by the exclude relation, and no other relations.

3.1 Aspect Oriented and Modular DCR Graphs

In an aspect oriented language based on DCR Graphs, we propose using the join operator to define an operator for adding "before" advices for a subset of events (the joincut). This could for instance be done by refining every event e in the joincut by a DCR Graph that adds an advice sub process as a milestone before the event e, ad e.g. the `authorize` event before the `action` event in Fig. 1. Then, every time an event e in the join cut is scheduled for execution, every event in the advice sub process will also be scheduled for execution, and must be executed before executing e due to the milestone relation.

Dually, an "after" advice may be added to an event e by joining a graph that has a response, condition and include relation from e to the advice sub process, like for the `request` and `reply` events in the G_{rr} graph in Fig. 4. This ensures that the advice must be carried out once after the event e. These two operations can then be combined to have both "before" and "after" advices.

Toggling of advices, e.g. as a result of context-events, can be achieved by joining in events that include and exclude the advice sub processes, e.g. like the `normal` and `emergency` events in Fig. 1. One can further constrain when an advice can be toggled as in Fig. 4, where the reply action can be cancelled, but only after the request has happened.

Finally, the join operator can also be considered as a general way of allowing modular definition of DCR Graphs. However, it should be stressed, that the join operator provide no guarantees for preserving safety and liveness properties. Indeed, it is easy to join graphs and achieve a circular condition dependency between events that may lead to a deadlock state if one of the events are required as response (and can not be excluded). Similarly, the join operator may introduce the possibility of a life lock, i.e. a DCR Graph with an infinite run that is not accepting and has no way of breaking out of the loop.

It is possible to verify safety and liveness properties of the composed DCR Graphs, e.g. by mapping the DCR Graph to a Büchi-automaton and verify the properties using the SPIN model checker as shown in [12]. This is however time consuming when the size of the processes grow. A more efficient verification currently explored is to carry out the verification on the corresponding transition systems with responses. Finally, an even more efficient guarantee of well-behaved modular composition, which we are currently investigating, is to define a notion of *behavioral type* for DCR Graphs based on the work on session types, which then guarantee that 1) well-typed graphs are safe and live, and 2) if compatible, well-typed DCR Graphs are joined, then the resulting graph is again well-typed.

4. Conclusion

We have proposed the declarative, event-based DCR Graphs model as a foundation for modular construction of context-sensitive, aspect-oriented processes. Concretely, we showed how the dynamic exclusion and inclusion primitives of DCR Graphs allow to turn the relevance of events as conditions/milestones and responses for any other event in the model on and off. It was exemplified by a simple authorization process fragment, where the need for authorization can be toggled by events signaling wether the context situation is an emergency or normal. Another example was a request/reply process fragment, where the reply can be cancelled by a cancel event in the context. Moreover, we presented a new join operator for DCR Graphs, allowing both modular composition and refinement, exemplified by joining the request/reply process and the authorization process, by which the reply was refined into two events, an action event (i.e. representing the reply) and an authorization event, and the remaining events and relations for authorization were inherited.

A key point is that aspects are not turned on and off when processes are initiated, but asynchronously at any point during the execution of the process. Another key point is that the formal semantics allows us to verify if the defined process, e.g. obtained by joining processes, have safety or liveness problems. As shown in [12, 13], DCR Graphs can be mapped to Büchi-automata. In [12] an implementation of this mapping and its application to verify safety and liveness properties of DCR Graphs using the SPIN model-checker are described. Recently we have developed a model checker that allows to verify properties directly on DCR Graphs. It avoids the translation to Büchi-automata and seems more efficient, however this is still work in progress. In [7] we have developed a technique for distributing DCR Graphs, which is somehow reverse to the composition operator. It allows to divide a DCR Graph in a collection of (not necessarily disjoint) sub graphs, which can then be executed at different locations. Finally, the fourth author has developed a prototype implemenation of PEPL [4] where events are extended with data, and the third author has implemented a workflow engine at Exformatics A/S based on DCR Graphs with data.

Related work: It goes beyond the scope of the present paper to give a comprehensive overview of related work. *Context-oriented programming* (e.g. [8, 18]) introduces the notion of *layers* in normal object-oriented, block structured programming languages such as Java, allowing features (aspects) to be activated or deactivated depending on the current context. The selection of active layers are typically done using the *with* primitive when methods are invoked. In [10], the event-driven and context-oriented approaches are combined, allowing events triggered in other threads to activate and de-activate layers somewhat similar to the include and exclude primitives of DCR Graphs.

The use of declarative primitives for the description of processes, and workflow processes in particular, is also treated in the work on Declare [19, 20]. Declare is similar to DCR Graphs in that processes are specified by temporal relations between events, indeed, the condition and responses relations are specified using the same graphical notation. Another related approac would be to specify processes using a temporal logic like LTL [14, 17] or CTL [1]. In [14] is shown that primitives similar to those used in DCR Graphs can indeed be represented in LTL. However, neither Declare, LTL nor CTL have explicit operators for the inclusion and exclusion of events as in DCR Graphs. Consequently, toggling of aspects can not simply be added by joining additional constraints. It is instead necessary to rewrite the logical formula (or Declare process) in a non trivial way. As an example, consider the request-reply-cancel pattern in Fig. 4. The request-reply pattern alone would typically be expressed in LTL by a formula like $G(\texttt{request} \implies F\texttt{reply}) \land (F\texttt{reply} \implies \neg\texttt{reply}U\texttt{request}) \land G(\texttt{reply} \implies N(F\texttt{reply} \implies \neg\texttt{reply}U\texttt{request}))$, where G reads *Generally*, F reads *Future*, U reads *Until* and N is the *Next* operator. That is, 1) it is generally the case that if a request happens, then a reply happens in the future, 2) if a reply happens, then that reply can not happen before at least one request has happened, and 3) if a reply happens, then if another reply happens some time in the future, it will not happen before a new request has happened. Now, to add the cancel option, it is ncecessary to rewrite the formula, it is not possible simply to add new constraints e.g. as a conjunction. This is because 1) a reply is not required if a cancel event happens, i.e. one must change the first conjunct to $G(\texttt{request} \implies F(\texttt{reply} \lor \texttt{cancel}))$, 2) after a cancel event, a reply should not occur until we have seen a new request, i.e. the third conjunct becomes $G(\texttt{reply} \lor \texttt{cancel} \implies N(F\texttt{reply} \implies \neg\texttt{reply}U\texttt{request}))$ and 3) a cancel event should not occur before the first request has happened, i.e. we should add an additional conjunct $(F\texttt{cancel} \implies \neg\texttt{cancel}U\texttt{request})$. In general, if the LTL formula does not have this specific form as given by DCR Graphs, we claim that it would be non-trivial to define how formulas should be changed to toggle aspects on and off, and at least not as simple as the join operator of DCR Graphs proposed in the present paper.

Acknowledgments

This research is supported by IT University of Copenhagen, Exformatics A/S, an Industrial PhD grant and the Danish Council for Strategic Research through the Jingling Genies project, grant no.: 2106-080046. We would like to thank the anonymous reviewers for careful reviews and suggestions.

References

[1] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In J. White, R. J. Lipton, and P. C. Goldberg, editors, *POPL*, pages 164–176. ACM Press, 1981. ISBN 0-89791-029-X. 6

[2] M. Carbone, T. T. Hildebrandt, G. Perrone, and A. Wasowski. Refinement for transition systems with responses. In S. S. Bauer and J.-B. Raclet, editors, *FIT*, volume 87 of *EPTCS*, pages 48–55, 2012. 3

[3] T. Hildebrandt and R. R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, volume 69 of *EPTCS*, pages 59–73, 2011. URL `http://www.itu.dk/people/rao/rao_files/dcrsplacescamredver.pdf`. 1

[4] T. Hildebrandt and F. Zanitti. A process-oriented event-based programming language. In F. Bry, A. Paschke, P. T. Eugster, C. Fetzer, and A. Behrend, editors, *DEBS*, pages 377–378. ACM, 2012. ISBN 978-1-4503-1315-5. 1, 5

[5] T. Hildebrandt, R. R. Mukkamala, and T. Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proceedings of IEEE International EDOC Conference*, 2011. URL `http://www.itu.dk/people/rao/pubs_accepted/dcrscasestudy-edoc11.pdf`. 1

[6] T. Hildebrandt, R. R. Mukkamala, and T. Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. URL `http://www.itu.dk/people/rao/pubs_accepted/fsenpaper.pdf`. 1, 2

[7] T. Hildebrandt, R. R. Mukkamala, and T. Slaats. Safe distribution of declarative processes. In *9th International Conference on Software Engineering and Formal Methods (SEFM) 2011*, 2011. 1, 5

[8] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-oriented programming. *Journal of Object Technology*, 7(3):125151, March Aprile 2008. 5

[9] G. J. Holzmann. *SPIN Model Checker, The: Primer and Reference Manual*. Addison-Wesley Professional, 2004. 3

[10] T. Kamina, T. Aotani, and H. Masuhara. Eventcj: a context-oriented programming language with declarative event-based context transition. In *Proceedings of the tenth international conference on Aspect-oriented software development*, AOSD '11, pages 253–264, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0605-8. doi: 10.1145/1960275.1960305. URL `http://doi.acm.org/10.1145/1960275.1960305`. 5

[11] K. M. Lyng, T. Hildebrandt, and R. R. Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Process-oriented information systems in healthcare (ProHealth 08)*, pages 36–43. BPM 2008 Workshops, 2008. URL `http://www.itu.dk/people/hilde/Papers/ProHealth08.pdf`. 1

[12] R. R. Mukkamala. *A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, June 2012. `http://www.itu.dk/people/rao/phd-thesis/DCRGraphs-rao-PhD-thesis.pdf`. 1, 3, 5

[13] R. R. Mukkamala and T. Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010. URL `http://www.itu.dk/people/rao/rao_files/dcrsextendedabstractTase2010.pdf`. 1, 3, 5

[14] R. R. Mukkamala, T. Hildebrandt, and J. B. Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008. 6

[15] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981. ISBN 0136619835. 2

[16] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universitet Hamburg, 1962. 2

[17] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. 6

[18] G. Salvaneschi, C. Ghezzi, and M. Pradella. Context-oriented programming: A programming paradigm for autonomic systems. *CoRR*, abs/1105.0069, 2011. 5

[19] W. van der Aalst, M. Pesic, H. Schonenberg, M. Westergaard, and F. M. Maggi. Declare. Webpage, 2010. `http://www.win.tue.nl/declare/`. 5

[20] W. M. van der Aalst and M. Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006. ISBN 978-3-540-38444-1. 5

Towards a Foundation for Modular Run-time Adaptable Process-Aware Information Systems[⋆]

Søren Debois[1], Thomas Hildebrandt[1], and Tijs Slaats[1,2]

[1] IT University of Copenhagen
debois,hilde,tslaats@itu.dk
[2] Exformatics A/S
tslaats@itu.dk

Abstract. We present the *DCR* process language* as a first step towards a foundation for modular process-aware information systems allowing dynamic creation of sub-processes with independent life-cycles and safe, run-time adaptation. The DCR* process language is a constraint-based process language generalising event structures. As the first main technical result we formalise when an adaptation is *conservative*, i.e. preserves the constraints of the adapted process and provide a decidable approximation to conservative adaptation referred to as *non-intrusive* adaptations. Such approximation is crucial, since we as the second main technical contribution prove that the DCR* language is Turing complete. This is in contrast to the sub-language of DCR processes, without dynamic sub-processes. This language corresponds to the model of DCR graphs introduced in previous work and known to charatise exactly languages that are the union of a regular and an ω-regular language. The developments throughout the paper is exemplified with a running example inspired by a process-aware information system for handling grant applications, for which our industrial partner has recently provided a DCR graph based implementation. Also, an online prototype implementation of the DCR* language can be found at http://tiger.itu.dk:8018/.

1 Introduction

Software systems today control increasingly complex processes operating in unpredictable contexts. At the same time, it is becoming more and more critical that the software behaves correctly, e.g. that it is compliant with safety, security and legal regulations. The combination of complexity, unpredictability and need for compliancy has lead to a general understanding that a foundation for the implementation of modular, run-time adaptable and formally verifiable software systems is needed. This is not least the case in the fields of *Process-Aware Information Systems* (PAIS) [34] and *Business Process Management* (BPM) [3], which constitute the context of the present work. The fields deal with systems driven by explicit process designs for the enactment and management of business processes and human workflows, and the study of formalisms for describing

[⋆] supported by the Velux foundation (grant 33295) and Innovation Fund Denmark.

processes has always been central in these fields: As a vehicle for communication, it is vital that a business process model is unambiguous; as a vehicle for understanding, it is vital that it is analysable; and as a foundation for practical systems, it is vital that it can be made executable. Popular models include in particular models which specify explicit sequencing of business activities as flow graphs, such as Petri Nets and Workflow Nets [2] which are the closest formal counterpart to the industrial standard Business Process Model and Notation (BPMN) [32].

However, an approach to process implementation based on flow graphs implicitly assumes the initial design of a *pre-specified* process graph, that implements the believed best practice given the initial required set of business rules and legal constraints. This is problematic in several ways: Firstly, the explicit flow graph often imposes more constraints than necessary. Secondly, procedures, rules and regulations change or the process graph turns out not to be the desired practice anyway. For long running processes, such as control software in hardware systems that can not be stopped or mortgages of credit institutions, the changes need to be reflected in running processes. And while the graph may be initially verified to implement the given business rules and legal constraints, it does typically not represent the rules *explicitly*. Thus, it is very difficult if at all possible to identify the required changes to the flow graph.

Declarative process languages [4,18] address this deficiency by leaving the exact sequencing of activities undefined, yet specifying the constraints processes must respect. This gives a workflow system the maximum flexibility available under the rules and regulations of the process. In practice, the caseworker or process engine is empowered to take what is considered the appropriate steps (e.g. considering resource usage) for the process and situation at hand, subject only to the constraints expressed in the process model. If the constraint language is well designed, the constraints can directly represent the business and legal regulations, making it easy to add or update constraints if the regulations change.

As a running example, we consider a grant application process of a funding agency that our industry partner, Exformatics, has recently implemented in a commercial solution [14].[3] The high-level requirements are: 1) applications can be received after a round is opened and until the deadline, 2) if a round is opened, the board must eventually meet, and 3) a board meeting can only happen when a round is open (i.e. before the deadline), if at least one application was received. A BPMN process implementing the requirements may be defined as shown in Fig. 1 below.

The process is initiated by an event Round indicating the start of a round, followed by a loop for receiving applications until the deadline, after which a board meeting is

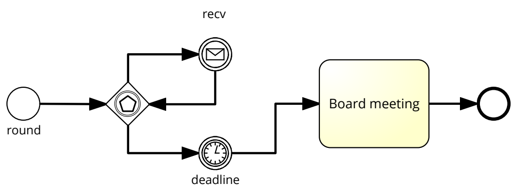


Fig. 1. Grant Application BPMN Process

[3] We deviate from the implemented solution when it makes our examples clearer.

held. However, the requirements are not explicitly represented in the process diagram, and moreover, the process introduces unnecessary, or at least unspecified constraints. For instance, no board meeting can be held before the deadline, even if applications are received, and it is not possible to reopen the round, e.g. if insufficiently many good applications were received. Of course, these possibilities may be modelled, but with the cost of making the process graph more complex.

Our industry partner Exformatics employs a declarative, graphical process notation, *Dynamic Condition Response (DCR) Graphs*, introduced in [18,30] and developed further in [19,7,31,20,12]. As exemplified in Fig. 2 below (produced with the tool at `http://tiger.itu.dk:8018/`), the DCR Graphs notation allows to specify the process by the four events (Receive, Deadline, Round, Board Meeting) and four relations between the events.

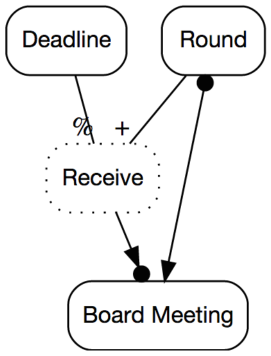


Fig. 2. Grant Application DCR Graph Process

The Receive event is dashed, representing that it is initially excluded from the process. The line from Round to Receive with a + sign at the end is an *include* relation, meaning that Receive is dynamically included if the event Round (the start of a round) happens. Dually, the line from Deadline to Receive with a % sign is an *exclude* relation, meaning that Receive is dynamically excluded if the event Deadline happens. Together, these two relations represent the first requirement. The arrow from Round to Board Meeting with a bullet at the start represents that Board Meeting is a *response* to (i.e. must happen eventually after) the Round event, as stated in the second requirement. Dually, the arrow from Receive to Board Meeting with a bullet at the end represents that Receive is a *condition* for Board Meeting, meaning that if Receive is included, i.e. a round is open, it must have happened before Board Meeting can happen, as stated in the third requirement.

The operational semantics of a DCR Graph is defined in terms of a *marking* assigning a triple of booleans (h, i, r) to each event, indicating whether or not the event previously (h)appened, is currently (i)ncluded, and/or is (r)estless. An event may be excluded in the initial marking and be dynamically included and excluded if it is related to other events by include and exclude relations, as illustrated by the Receive event above. Similarly, an event may be restless in the initial marking or become restless because it is a response to an event that happened as for the Board Meeting event above. Finite or infinite executions are then defined to be accepting only if every restless event is executed or excluded at a later point in the execution. As described in [18,30], (labelled) DCR Graphs is a conservative generalisation of (labelled) prime event structures, allowing *finite representations of infinite behaviour* and to express *liveness* properties. The former is achieved by allowing an event to happen more than once and change dynamically between being in conflict (excluded) and being consistent (included) with the current state. The latter is achieved via the notion of restless events and the acceptance criteria described above.

Recently, an extension of DCR Graphs to so-called *hierarchical DCR Graphs*, supporting dynamic creation of sub-processes with (local) events and independent life cycles has been presented [12]. This extension was motivated in practice by the funding agency process above, e.g. to allow each received application to have its own approval events and decision life cycle, and thereby recording the decision made for each application. However, although the graphical notation of DCR Graphs has been adopted by industry, it does not scale well to larger and more complex processes due to lack of compositionality. Also, the expressive power of hierarchical DCR Graphs was left open by [12].

The central contribution of the present paper is to provide the *DCR* process language*, a core constraint-based process language for modular process-aware information systems with *dynamic creation of sub-processes with independent life-cycles*, based on the primitives introduced in (hierarchical) DCR Graphs, but equipped with a *compositional* operational semantics. The compositional semantics provides means for modular definition of and reasoning for DCR* processes. As an application, we show that the DCR* language supports *run-time adaptation* by composition. We formalise when an adaptation is *conservative*, i.e. preserve requirements of the adapted process and provide as our first technical contribution a syntactic, and thus decidable, approximation to conservative adaptation referred to as *non-intrusive* adaptations. Such approximation is crucial, since we as a second technical contribution prove by an encoding of Minsky machines, that the DCR* language is in fact Turing complete. This is in contrast to the sub-language of DCR processes corresponding to DCR Graphs, i.e. without dynamic sub-processes and local events, that can be shown to charatise exactly languages that are the union of a regular and an ω-regular language (sketched in [30] and proven in the full version of the paper).

Overview of the paper: In Sec. 2 we provide the DCR process language and its compositional semantics. We then extend the language in Sec. 3 to DCR* supporting dynamic creation of interacting sub-processes with fresh (local) events. We address run-time adaptation by composition in Sec. 4, formalise when and adaptation is conservative and provide the syntactic, non-invasiveness approximation to conservative adaptation. In Sec. 5 we prove that the DCR* language is Turing complete. Finally, we conclude and outline in Sec. 6 related, current and future directions of work exploiting the results in the present paper. An online, research prototype implementation of the process languages presented in the paper, with a mapping to DCR Graphs, can be found at `http://tiger.itu.dk:8018/`.

2 Dynamic Condition Response (DCR) Processes

Below we introduce the Dynamic Condition Response (DCR) process language. As already informally described in the introduction, it is based on the notions of dynamic inclusion and exclusion of labelled events, related by conditions and response relations introduced in DCR Graphs [30,18].

We assume fixed universes of *events* $\mathcal{E}$ and *labels* $\mathcal{L}$; each event $e \in \mathcal{E}$ has an associated label $\ell(e) \in \mathcal{L}$. A DCR process $[M]\ T$ comprises a *marking* M and a *term* T. The syntax of both are given in Fig. 3 below.

$T, U ::= f \rightarrow\!\bullet\ e$	condition	$\phi ::= \mathsf{t} \mid \mathsf{f}$	boolean value
$\mid f \leftarrow\!\bullet\ e$	response	$\Phi ::= (\phi, \phi, \phi)$	event state
$\mid f +\!\!\leftarrow e$	inclusion	$M, N ::= M, e : \Phi$	marking
$\mid f \%\!\!\leftarrow e$	exclusion		
$\mid T \mid U$	parallel	$P, Q ::= [M]\ T$	process
$\mid 0$	unit		

Fig. 3. DCR Processes Syntax.

A term is a parallel composition of *relations* between *events*. We recall from the introduction how the relations regulate what behaviour the term may exhibit:

1. A *condition* $f \rightarrow\!\bullet\ e$ imposes the *constraint* that for event e to happen, the event f must either previously have happened or currently be excluded.
2. A *response* $f \leftarrow\!\bullet\ e$ imposes the *effect* that when e happens, f becomes restless and must eventually happen or be excluded.
3. An *exclusion* $f \%\!\!\leftarrow e$ imposes the *effect* that when e happens, it *excludes* f. An excluded event cannot happen; it is ignored as a condition; and it need not happen if restless, unless it is re-included by the final relation:
4. An *inclusion* $f +\!\!\leftarrow e$ imposes the *effect* that when the event e happens, it re-includes the event f.

All four relations refer to a marking M, a finite map from events to triples of booleans (h, i, r), referred to as the *event state* and indicating whether or not the event previously (h)appened, is currently (i)ncluded, and/or is (r)estless. A restless event represents an unfulfilled obligation: once it happens, it ceases to be restless. As commonly done for environments, we write markings as finite lists of pairs of events and event states, e.g. $e_1 : \Phi_1, \ldots, e_k : \Phi_k$ but treat them as maps, writing $\mathsf{dom}(M)$ and $M(e)$, and understand $M, e : \Phi$ to be undefined when $e \in \mathsf{dom}(M)$. The *free events* $\mathsf{fe}(T)$ of a term T is (for now) simply the set of events appearing in it. (This changes when we introduce local events in Sec. 3 below.) We require of a process $P = [M]\ T$ that $\mathsf{fe}(T) \subseteq \mathsf{dom}(M)$, and so define $\mathsf{fe}(P) = \mathsf{dom}(M)$. The *alphabet* $\mathsf{alph}(P)$ is the set of labels of its free events.

Example 1 (Grant process term). The example of fig. 2 can be mapped to the following term:

$$T_0 = \mathsf{recv}\ \%\!\!\leftarrow \mathsf{deadline} \mid \mathsf{recv} +\!\!\leftarrow \mathsf{round} \mid \mathsf{bm} \leftarrow\!\bullet\ \mathsf{round} \mid \mathsf{recv} \rightarrow\!\bullet\ \mathsf{bm}$$

Initially, no event has happened, no event is restless, and every event but recv is included, giving us the marking:

$$M_0 = \mathsf{round} : (\mathsf{f}, \mathsf{t}, \mathsf{f}), \mathsf{deadline} : (\mathsf{f}, \mathsf{t}, \mathsf{f}), \mathsf{recv} : (\mathsf{f}, \mathsf{f}, \mathsf{f}), \mathsf{bm} : (\mathsf{f}, \mathsf{t}, \mathsf{f})\ .$$

□

Example 2 (Event structures). A labelled prime event structure [38] can be defined as a tuple $\mathbf{E} = (E, \leq, \#, \ell, L)$ where E is a set of events, $\leq$ is a partial order on events defining the *causal dependency* relation (satisfying an axiom of finite cause), $\#$ is the binary, symmetric and irreflexive *conflict* relation (satisfying an axiom of hereditary conflict) and ℓ is a labelling function assigning every event to a label. A finite event structure $\mathbf{E}$ can be represented as the DCR term

$$T_{\mathbf{E}} = \prod_{e<e'} e \rightarrow\!\bullet\; e' \mid \prod_{e\#e' \vee e=e'} e \;\%\!\leftarrow e'$$

A state of an event structures is referred to as a *configuration*, defined as a finite, downwards closed and conflict free set $C \subseteq E$ of events. Define $C^{\#} = \{e \mid \exists e' \in C.e\#e'\}$. A configuration for finite event structures can then be represented by the marking $M_{\mathbf{E}}$ defined by $M_{\mathbf{E}}(e) = (\mathsf{t}, \mathsf{f}, \mathsf{f})$ for $e \in C$, $M_{\mathbf{E}}(e) = (\mathsf{f}, \mathsf{f}, \mathsf{f})$ for $e \in C^{\#}$ and $M_{\mathbf{E}}(e) = (\mathsf{f}, \mathsf{t}, \mathsf{f})$ for $e \notin C \cup C^{\#}$. The DCR process $[M_{\mathbf{E}}]\ T_{\mathbf{E}}$ then represent a pair of a configuration and an event structure, which indeed will have the same behaviour as the event structure. An event structure with a set $R \subseteq E$ of *restless* events as considered in [37] is then defined in the same way, except that the events in R will initially be restless in the marking representing the configuration, i.e. the third component of the event state will be t. □

We give semantics to DCR processes incrementally. First, the notion of an event being *enabled* and what *effects* it has. The judgement $[M]\ T \vdash e : E, I, R$, defined in Fig. 4, should be read: "in the marking M, the term T allows the event e to happen, with the effects of excluding events E, including events I, and making events R restless."

$$[M, f : (h, i, \_), e : (\_, \mathsf{t}, \_)]\ f \rightarrow\!\bullet\; e \vdash e : \emptyset, \emptyset, \emptyset \qquad \text{(when } i \Rightarrow h\text{)}$$
$$[M, e : (\_, \mathsf{t}, \_)]\ f \bullet\!\!\leftarrow e \vdash e : \emptyset, \emptyset, \{f\}$$
$$[M, e : (\_, \mathsf{t}, \_)]\ f +\!\!\leftarrow e \vdash e : \emptyset, \{f\}, \emptyset$$
$$[M, e : (\_, \mathsf{t}, \_)]\ f \;\%\!\!\leftarrow e \vdash e : \{f\}, \emptyset, \emptyset$$
$$[M, e : (\_, \mathsf{t}, \_)]\ 0 \vdash e : \emptyset, \emptyset, \emptyset$$
$$[M, e : (\_, \mathsf{t}, \_)]\ f'\ \mathcal{R}\ f \vdash e : \emptyset, \emptyset, \emptyset \qquad \text{(when } e \neq f\text{)}$$

Fig. 4. Enabling & effects. We write "_" for "don't care", i.e., either true t or false f, and write $\mathcal{R}$ for any of the relations $\rightarrow\!\bullet, \bullet\!\!\leftarrow, +\!\!\leftarrow, \%\!\!\leftarrow$.

The first rule says that if f is a condition for e, then e can happen only if (1) it is itself included, and (2) if f is included, then f previously happened. The second rule says that if f is a response to e and e is included, then e can happen with the effect of making f restless. The third (fourth) rule says that if f is included (excluded) by e and e is included, then e can happen with the effect of including (excluding) f. The fifth rule says that the completely unconstrained process 0, an event e can happen if it is currently included. The last rule says that a relation allows any included event e to happen without effects when e is not the relation's right-hand–side event.

Given enabling and effects of events, we define the *action* of respectively an *event* e and an *effect* $\delta = (E, I, R)$ on a marking M pointwise by the action on individual event states $f : (h, i, r)$ as follows.

$$\text{(Event action)} \quad e \cdot \big(f : (h, i, r))\big) \overset{\text{def}}{=} f : \big(\, \underbrace{h \vee (f = e)}_{\text{happened?}},\ i,\ \underbrace{r \wedge (f \neq e)}_{\text{restless?}} \,\big)$$

$$\text{(Effect action)} \quad \delta \cdot \big(f : (h, i, r)\big) \overset{\text{def}}{=} f : \big(\, h,\ \underbrace{(i \wedge f \notin E) \vee f \in I}_{\text{included?}},\ \underbrace{r \vee f \in R}_{\text{restless?}} \,\big)$$

That is, for the event action, if $f = e$, the event is marked "happened" (first component becomes t) and it ceases to be restless (last component becomes f). For the effect action, the event only stays included (second component) if $f \notin E$ (it is not excluded) or $f \in I$ (it is included). This also means that if an event is both excluded and included by the effect, inclusion takes precedence. Finally, f is marked restless (third component) if either it was already restless or it became restless ($f \in R$). We then define the combined action of an event and effect by $(e : \delta) \cdot M = \delta \cdot (e \cdot M)$.

In defining the compositional semantics, we need to merge parallel markings and effects. Merge on markings is *partial*, since it is only defined on markings that agree on their overlap:

$$(M_1, e : m) \oplus (M_2, e : m) = (M_1 \oplus M_2), e : m$$
$$(M_1, e : m) \oplus M_2 = (M_1 \oplus M_2), e : m \quad \text{when } e \notin \mathsf{dom}(M_2)$$

The *merge of effects* δ is always defined; it is simply the pointwise union:

$$(E_1, I_1, R_1) \oplus (E_2, I_2, R_2) = (E_1 \cup E_2, I_1 \cup I_2, R_1 \cup R_2)$$

With these mechanics in place, we give transition semantics of processes in Fig. 5.

$$\frac{[M]\ T \vdash e : \delta}{[M]\ T \xrightarrow{e:\delta} T}\ [\text{INTRO}] \qquad \frac{[M]\ T_1 \xrightarrow{e:\delta_1} T_1' \qquad [M]\ T_2 \xrightarrow{e:\delta_2} T_2'}{[M]\ T_1 \mid T_2 \xrightarrow{e:\delta_1 \oplus \delta_2} T_1' \mid T_2'}\ [\text{PAR}]$$

$$\frac{[M]\ T \xrightarrow{e:\delta} T'}{[M]\ T \xrightarrow{e} [e : \delta \cdot M]\ T'}\ [\text{EFFECT}]$$

Fig. 5. Basic transition semantics.

We use two form of transitions: the *effect transition* $[M]\ T \xrightarrow{e:\delta} T'$ says that $[M]\ T$ may exhibit event e with effect δ, in the process updating the term T

to become T'. (At this stage we will always have $T = T'$; we will need updates only when we extend the calculus in Section 3 below.) The *process transition* $[M]\ T \xrightarrow{e} [N]\ U$ takes a process to another process, applying the effect of e to the marking M, and thus only exhibiting the event e. The [INTRO] rule elevates an enabled event with an effect to an effect transition. The [PAR] rule combines effect transitions from the two sides of a parallel when they have compatible markings. The [EFFECT] rule lifts an effect transition to a process transition by applying the effect to the marking. Process transitions gives rise to an LTS, which we equip with a notion of *acceptance* defined formally below, corresponding to that for DCR Graphs [10]: a run is accepting if every restless event eventually either happens or is excluded.

Definition 1. *A DCR process defines an LTS with states $[M]\ T$ and (process) transitions $[M]\ T \xrightarrow{e} [N]\ U$. A run of $[M]\ T$ is a finite or infinite sequence of transitions $[M]\ T = [M_0]\ T_0 \xrightarrow{e_0} \cdots$. A run is* accepting *iff for every state $[M_i]\ T_i$, when $M_i(e) = (\_, \mathsf{t}, \mathsf{t})$ then there exists $j \geq i$ s.t. either $M_j(e) = (\_, \mathsf{f}, \_)$ or $[M_j]\ T_j \xrightarrow{e:\delta} [M_{j+1}]\ T_{j+1}$. A* trace *of a process $[M]\ T$ is a possibly infinite string $s = (s_i)_{i \in I}$ s.t. $[M]\ T$ has an accepting run $[M_i]\ T_i \xrightarrow{e_i} [M_{i+1}]\ T_{i+1}$ with $s_i = \ell(e_i)$.*

Example 3 (Grant process transitions). As transitions change only marking, not terms, we show a run by showing changes in the marking. In the table below, rows indicate changes to the marking as the event on the left happens. Columns "h,i,r" indicate whether an event is marked (h)appened, (i)ncluded, and/or (r)estless. The column "Accepts?" indicates whether the current marking is accepting or not and the final column "Enabled" indicates which events are enabled after executing the event on the left.

Event	round			deadline			recv			bm			Accepts?	Enabled
happening	h	i	r	h	i	r	h	i	r	h	i	r		
(none)	f	t	f	f	t	f	f	f	f	f	t	f	t	{round, deadline, bm}
round	t							t				t	f	{round, deadline, recv}
deadline				t				f					f	{round, deadline, bm}
bm										t		f	t	{round, deadline, bm}
round								t		t		t	f	{round, deadline, recv}
recv							t						f	{round, deadline, recv, bm}
bm												f	t	{round, deadline, recv, bm}

After the first round event, bm cannot happen because of recv $\rightarrow\bullet$ bm. When deadline happens, it excludes recv because of bm $\%\!\leftarrow$ recv, and exclusion of recv voids the condition recv $\rightarrow\bullet$ bm; so after deadline, bm may again happen. When round subsequently re-includes recv, bm is again disabled. Acceptance of the processes changes throughout. Because of bm $\leftarrow\bullet$ round, whenever round executes it makes bm restless, preventing the process from accepting until bm later happens, ceasing to be restless. In our examples, we identify events and labels, so the above table indicates an accepting trace $\langle$round, deadline, bm, round, recv, bm$\rangle$. □

We conclude by noting that DCR processes as presented here correspond exactly to DCR graphs [30,36,12].

Theorem 1. *There exists a language-preserving map from DCR processes to DCR graphs; and vice versa.*

3 DCR* Processes: Local events and Reproduction

As stated in the introduction, through our cooperation with industrial partners we identified a need for instantiating sub-processes dynamically [12]. We therefore extend the DCR process language by adding *local events* and *reproductive events*, giving rise to the DCR* process language.

The extended syntax is given in Fig. 6 to the right. We assume that for each label $l \in \mathcal{L}$, there exists infinitely many events e with that label, that is, with $\ell(e) = l$.

$T, U ::=$	$\dots$	
	$\mid (\nu e : \Phi)\ T$	local event
	$\mid e\{T\}$	reproductive event

Fig. 6. DCR* syntax.

The *local event* $(\nu e : \Phi)\ T$ asserts that e is local to the term T. This construct is binding, and we take terms up to *label preserving* α-conversion, i.e., $(\nu e : \Phi)\ T = (\nu f : \Phi)\ T[f/e]$ iff $\ell(e) = \ell(f)$ and f not free in T. As usual, we assume the Barendrecht-convention and may thus assume binders are distinct. A *reproductive event* $e\{T\}$ unfolds a copy of T whenever it happens.

We define *free events* and *alphabet* for DCR* processes.

Definition 2. *The* free events $\mathsf{fe}(T)$ *of a term* T *is defined recursively as follows.*

$$\begin{aligned}
\mathsf{fe}(e\ \mathcal{R}\ f) &= \{e, f\}\\
\mathsf{fe}(T \mid U) &= \mathsf{fe}(T) \cup \mathsf{fe}(U)\\
\mathsf{fe}(0) &= \emptyset\\
\mathsf{fe}((\nu e : \Phi)\ T) &= \mathsf{fe}(T) \setminus \{e\}\\
\mathsf{fe}(e\{T\}) &= \{e\} \cup \mathsf{fe}(T)
\end{aligned}$$

The free events of a process $\mathsf{fe}([M]\ T)$ *is simply* $\mathsf{fe}([M]\ T) = \mathsf{dom}(M)$*; we maintain the requirement that a process* $[M]\ T$ *has* $\mathsf{fe}(T) \subseteq \mathsf{dom}(M)$. *The* alphabet $\mathsf{alph}(P)$ *of a process is the set of labels associated with its events, defined recursively as follows.*

$$\begin{aligned}
\mathsf{alph}(e\ \mathcal{R}\ f) &= \{\ell(e), \ell(f)\}\\
\mathsf{alph}(T \mid U) &= \mathsf{alph}(T) \cup \mathsf{alph}(U)\\
\mathsf{alph}(0) &= \emptyset\\
\mathsf{alph}((\nu e : \Phi)\ T) &= \{\ell(e)\} \cup \mathsf{alph}(T)\\
\mathsf{alph}(e\{T\}) &= \{\ell(e)\} \cup \mathsf{alph}(T)
\end{aligned}$$

The following Lemma states that transitions preserve free events and alphabet.

Lemma 1. *Transitions* $[M]\ T \xrightarrow{\lambda} T'$ *and* $[M]\ T \xrightarrow{\gamma} [M']\ T'$ *preserve free events and alphabet, that is* $\mathsf{fe}(M) = \mathsf{fe}(M')$, $\mathsf{fe}(T) = \mathsf{fe}(T')$, $\mathsf{alph}(T) = \mathsf{alph}(T')$, *and* $\mathsf{alph}(M) = \mathsf{alph}(M')$.

Proof. Preservation of free events and alphabet of terms for effect transitions follows by easy induction on the derivation of the transition. For preservation for process transitions, observe that by cases on the rules admitting a transition $[M]\ T \xrightarrow{\gamma} [M']\ T'$, we must have $\mathsf{dom}(M) = \mathsf{dom}(M')$ by definition of the action operator $- \cdot M$; the desiderata now follows.

The transition rules for the new constructs are given in Fig. 7. Only terms and transition rules are extended; markings are the same.

$$\frac{[M, f : \Phi]\ T \xrightarrow{e:\delta} T' \quad f : \Phi' = (e : \delta) \cdot (f : \Phi) \quad \gamma = \nu e \text{ if } e = f, \text{ o.w. } \gamma = e}{[M]\ (\nu f : \Phi)\ T \xrightarrow{\gamma:(\delta\backslash f)} (\nu f : \Phi')\ T'}\ [\textsc{local}]$$

$$\frac{[M]\ T \xrightarrow{\nu e:\delta} T'}{[M]\ T \mid U \xrightarrow{\nu e:\delta} T' \mid U}\ [\textsc{par-2}] \qquad \frac{[M]\ T \xrightarrow{e:\delta} T'}{[M]\ e\{T\} \xrightarrow{e:\delta} e\{T\} \mid T'}\ [\textsc{rep}]$$

$$\frac{[M]\ T \xrightarrow{\nu e:\delta} T'}{[M]\ T \xrightarrow{\nu e} [\delta \cdot M]\ T'}\ [\textsc{effect-2}]$$

Here $\delta\backslash f = (E\backslash\{f\}, I\backslash\{f\}, R\backslash\{f\})$. We omit the obvious rule symmetric to [Par-2].

Fig. 7. Transition semantics for local and reproductive events.

Rule [local] gives semantics to events happening in the scope of a local event binder. An effect on the local event is recorded in the marking in the binder of that event. The event might have effects on non-local events, e.g., in $(\nu f : M)\ e \mathbin{+\!\!\leftarrow} f$, the local f has effects on the non-local e. Thus the effects are preserved in the conclusion, except that part of the effect which pertain only to f. Rule [par-2] propagates a local effect through a parallel composition. It's possible that the effect δ mentions events in U; however, it cannot mention events *local* to U. So the effects of δ on U are fully expressed in the (eventual) effect of δ on M. Rule [effect-2] lifts effect transitions with local events to process transitions. Finally, the rule [rep] implements reproductive events: If the guarding event e happening would update the body T to become T', then e can unfold to such a T'.

To define accepting runs we need to track local restless events across transitions. Fortunately, DCR* is simple enough that we can simply assume that α-conversion happens only during replication (i.e., local events duplicated by [rep] are chosen globally fresh).

Definition 3. *A run of a DCR* process $[M]\ T$ is a finite or infinite sequence $[M_i]\ N_i \xrightarrow{\lambda_i} [M_{i+1}]\ N_{i+1}$ with $\lambda = e_i$ or $\lambda = \nu e_i$. The trace of a run is the sequence of labels of its events, i.e., the string given by $\ell(\lambda_i)$ where $\ell(\nu e) \overset{def}{=} \ell(e)$. A run is* accepting *if whenever an event e is marked as restless in M_i respectively a local event νe is marked as restless by its binder in T_i, then there exists some $j \geq i$ s.t. either $[M_j]\ T_j \xrightarrow{\lambda_i} [M_{j+1}]\ T_{j+1}$ with $\lambda_i = e$ respectively $\lambda_i = \nu e$; or the event state of e in M_j respectively T_j has e excluded.*

Example 4 (Grant process with reproductive and local events). We now consider the requirement that when an application is received, a committee recommends either approval or rejection to the board. The committee might rescind an approval, but cannot reverse a rejection. Moreover, when applications are competing for resources, the board cannot make a final decision until it has a recommendation for every received application. We again use events recv and bm for receiving an application and convening a board meeting. We use *local* events νapprove and νreject to model the per-application evaluation process replicated by recv. We first give a term A modelling a *single* recommendation sub-process.

$$A = (\nu\mathsf{approve} : (\mathsf{f},\mathsf{t},\mathsf{t}))\ (\nu\mathsf{reject} : (\mathsf{f},\mathsf{t},\mathsf{f}))\ \big(\mathsf{approve}\ \%\!\leftarrow \mathsf{reject} \mid \mathsf{approve} \rightarrow\!\bullet\ \mathsf{bm}\big)$$

Note that approve and reject are local and can thus not be constrained further outside the scope, yet approve has a condition relation to the non-local bm. We make the approve event initially restless, which will mean that in order for the process to be accepting either approve must happen or be excluded (because reject happens).
We can now model a process $[M_1]\ T_1$ reproducing the per-application sub-process A every time recv happens

$$T_1 = \mathsf{recv}\{A\} \mid \mathsf{recv} \rightarrow\!\bullet\ \mathsf{bm} \qquad M_1 = \mathsf{recv} : (\mathsf{f},\mathsf{t},\mathsf{f}), \mathsf{bm} : (\mathsf{f},\mathsf{t},\mathsf{f})$$

(To keep the process simple, we only retained from the previous example the constraint that a board meeting requires receiving an application). In DCR* , the term *does* change as the process evolves. Let's see $[M_1]\ T_1$ evolve:

$$\begin{aligned}
[M_1]\ T_1 &\xrightarrow{\mathsf{recv}} [M_2]\ T_1 \mid A_1 \\
&\xrightarrow{\mathsf{recv}} [M_2]\ T_1 \mid A_1 \mid A_2 && (1)\\
&\xrightarrow{\nu\mathsf{approve}_1} [M_2]\ T_1 \mid \big((\nu\mathsf{approve}_1 : (\boxed{\mathsf{t}},\mathsf{t},\boxed{\mathsf{f}}))\ (\nu\mathsf{reject}_1 : (\mathsf{f},\mathsf{t},\mathsf{f})) && (2)\\
&\qquad\qquad \mathsf{approve}_1\ \%\!\leftarrow \mathsf{reject}_1 \mid \mathsf{approve}_1 \rightarrow\!\bullet\ \mathsf{bm}\big) \mid A_2 \\
&\xrightarrow{\nu\mathsf{reject}_2} [M_2]\ T_1 \mid \big((\nu\mathsf{approve}_1 : (\mathsf{t},\mathsf{t},\mathsf{f}))\ (\nu\mathsf{reject}_1 : (\mathsf{f},\mathsf{t},\mathsf{f})) && (3)\\
&\qquad\qquad \mathsf{approve}_1\ \%\!\leftarrow \mathsf{reject}_1 \mid \mathsf{approve}_1 \rightarrow\!\bullet\ \mathsf{bm}\big) \\
&\qquad \big((\nu\mathsf{approve}_2 : (\mathsf{f},\boxed{\mathsf{f}},\mathsf{t}))\ (\nu\mathsf{reject}_2 : (\boxed{\mathsf{t}},\mathsf{t},\mathsf{f})) \\
&\qquad\qquad \mathsf{approve}_2\ \%\!\leftarrow \mathsf{reject}_2 \mid \mathsf{approve}_2 \rightarrow\!\bullet\ \mathsf{bm}\big) \\
&\xrightarrow{\mathsf{bm}} [M_3]\ T_1 \mid \cdots && (4)
\end{aligned}$$

Here $M_2 = \mathsf{recv} : (\boxed{\mathsf{t}}, \mathsf{t}, \mathsf{f}), \mathsf{bm} : (\mathsf{f}, \mathsf{t}, \mathsf{f})$ and $M_3 = \mathsf{recv} : (\mathsf{t}, \mathsf{t}, \mathsf{f}), \mathsf{bm} : (\boxed{\mathsf{t}}, \mathsf{t}, \mathsf{f})$.

At (1), the processes A_1 and A_2 are copies of A where the local events $\mathsf{approve}$ and reject have been α-converted to $\mathsf{approve}_1, \mathsf{approve}_2$ and $\mathsf{reject}_1, \mathsf{reject}_2$ respectively, following the convention of unique local names. Moreover, because they have not happened in the local markings under the binders, bm cannot happen. To see this, observe that by the [PAR]-rule, for the whole process to exhibit bm, every part of it must also exhibit bm. But $(\nu\mathsf{approve}_1 : (\mathsf{f}, \mathsf{t}, \mathsf{t})) \dots \mathsf{approve}_1 \rightarrow\!\bullet\ \mathsf{bm}$ cannot: the hypothesis of rule [LOCAL], that bm could happen if $\mathsf{approve}_1$ is considered global with marking $(\mathsf{f}, \mathsf{t}, \mathsf{t})$, cannot be established.

When a local $\mathsf{approve}_i$ event happens, its local marking changes to reflect that the event happened and is no longer restless, as indicated with grey background in (2). However, $\mathsf{approve}_1$ happening is not enough to enable bm; it is still disabled by the other copy. Also, the entire process is not in an accepting state, since $\mathsf{approve}_2$ is still restless and included. Once reject happens in the second copy (3), excluding $\mathsf{approve}$ in that copy, bm is enabled and the process is in an accepting state: of the two local $\mathsf{approve}$ events bm is conditional upon, one has happened (and thus also no longer restless), and the other is excluded (and thus also no longer required for acceptance).

□

4 Run-time adaptation by composition

In this section, we consider run-time adaptation by dynamic composition. The section ends with the first main technical result of the paper (Theorem 2), that we can find a (practically useful and) decidable approximation to adaptations that preserve compliancy with existing requirements.

Definition 4. *Given DCR* processes $[M]\ T$ and $[N]\ S$ their* merge *is defined when $M \oplus N$ is, in which case it is $[M]\ T \oplus [N]\ S = [M \oplus N]\ T \mid S$. When the merge of two processes is defined, we say that they are* marking compatible.

Example 5. Suppose now *as the grant process runs*, a new requirement comes up: For regulatory reasons, a board meeting must eventually be followed by an audition. We easily model this constraint using restlessness and a new event, audit. However, as we are introducing a new event, we must also introduce additional marking. The following process R_1 embodies the adaptation we wish to achieve.

$$R_1 = [\mathsf{bm} : (\mathsf{f}, \mathsf{t}, \mathsf{f}), \mathsf{audit}(\mathsf{f}, \mathsf{t}, \mathsf{f})]\ \mathsf{audit} \leftarrow\!\bullet\ \mathsf{bm}$$

We can now adapt the process $P = [M_1]\ T_1$ of Example 4 with respect to R_1:

$$P_1 = P \oplus R_1 = [M_1, \mathsf{audit} : (\mathsf{f}, \mathsf{t}, \mathsf{f})]\ T_1 \mid \mathsf{audit} \leftarrow\!\bullet\ \mathsf{bm}$$

Note that adaptation is not restricted to apply only to the initial process. As a second example, suppose further that it is also decreed that *during* an audit, no further applications can be received. We adapt P_1 with R_2 as follows:

$$\begin{aligned}R_2 &= [\mathsf{recv} : (\mathsf{f},\mathsf{t},\mathsf{f}), \mathsf{audit} : (\mathsf{f},\mathsf{t},\mathsf{f}), \mathsf{pass} : (\mathsf{f},\mathsf{t},\mathsf{f})]\ \mathsf{recv}\ \%\!\leftarrow \mathsf{audit} \mid \mathsf{recv} +\!\!\leftarrow \mathsf{pass}\\ P_2 &= P_1 \oplus R_2\\ &= [M_1, \mathsf{audit} : (\mathsf{f},\mathsf{t},\mathsf{f}), \mathsf{pass} : (\mathsf{f},\mathsf{t},\mathsf{f})]\ T_1 \mid \mathsf{audit} \leftarrow\!\bullet\ \mathsf{bm}\\ &\quad \mid \mathsf{recv}\ \%\!\leftarrow \mathsf{audit} \mid \mathsf{recv} +\!\!\leftarrow \mathsf{pass}\end{aligned}$$

□

However, when we extend the set of requirements, the run-time adaptation of P to P' should ideally not allow traces that were not allowed by the old set of requirements. We may try to formulate this property by language inclusion $\mathsf{lang}(P') \subseteq \mathsf{lang}(P)$. But as we have seen, run-time adaptation might entail adding *new* events (audit), so we cannot in general expect language inclusion. Hence, we consider language inclusion only w.r.t. the alphabet of P. In doing so we employ the following notation.

Notation. Given a sequence s, write $s|_\Sigma$ for the largest sub-sequence s' of s s.t. $s'_i \in \Sigma$; e.g, if $s = AABC$ then $s|_{A,C} = AAC$. We lift projection to sets of sequences point-wise.

Definition 5. *Given marking compatible DCR* processes P and Q, we say that Q* is conservative for *P iff* $\mathsf{lang}(P \oplus Q)|_{\mathsf{alph}(P)} \subseteq \mathsf{lang}(P)$.

Example 6. Continuing the above example, we now see a distinction between the adaptation by R_1 and R_2: the former is conservative for P, whereas the latter is not for P_1. To see that R_1 is conservative, observe that it only makes P_2 *less* accepting (because of the potential restlessness of the new event audit). To see that R_2 is not conservative, observe that $P_1 \oplus R_2$ has the following accepting execution:

$$P_1 \oplus R_2 \xrightarrow{\mathsf{audit}} \xrightarrow{\mathsf{bm}} \xrightarrow{\mathsf{audit}}$$

Here audit excludes recv, and so enables bm to execute; bm in turn makes audit restless, so after a second audit, we have an accepting trace $t = \langle \mathsf{audit},\ \mathsf{bm},\ \mathsf{audit}\rangle$. However, bm cannot not be the first event of a trace of P_1 because it is conditional on the non-executed recv. Formally, we found a counter-example to conservation:

$$t|_{\mathsf{alph}(P_1)} = \langle \mathsf{audit},\ \mathsf{bm},\ \mathsf{audit}\rangle|_{\{\mathsf{bm},\mathsf{recv},\mathsf{approve},\mathsf{reject}\}} = \langle \mathsf{bm}\rangle \notin \mathsf{lang}(P_1)$$

Inspecting the adaptation R_2 more closely, one see that the problem comes from the dynamic exclusion of the recv event, since it does not only makes the reception of applications impossible, but also enables events such as bm that are conditioned on recv. A better way is to block recv by introducing a new condition:

$$R'_2 = [\mathsf{recv} : (\mathsf{f},\mathsf{t},\mathsf{f}), \mathsf{audit} : (\mathsf{f},\mathsf{t},\mathsf{f})]\ \mathsf{audit}\{(\nu \mathsf{pass} : (\mathsf{f},\mathsf{t},\mathsf{f}))\ \mathsf{pass} \rightarrow\!\bullet\ \mathsf{recv}\}$$

Here, once audit happens, recv is barred from executing until the local event pass has happened. The corresponding adaptation *is* conservative. □

4.1 A Decidable Approximation of Conservative Adaptations

We shall see in the next section that it is unfortunately *undecidable* to identify the language of a DCR* process, so we will need an approximation of whether some P is conservative for some Q. The key problem is that new exclusions might remove conditions preventing events from firing, as we saw above, and that new inclusions might enable events to fire. This leads us to the following approximation:

Definition 6 (Non-invasive adaptation). *Let $P_1 = [M_1]\,T_1$ and $P_2 = [M_2]\,T_2$ be processes. We say that P_1* non-invasive *for P_2 iff*

1. *For every context $C(-)$, such that $T_1 = C(e \rightarrow\% f)$ or $T_1 = C(e \rightarrow+ f)$, either f is bound in $C(-)$ or $f \notin \mathsf{fe}(P_2)$; and*
2. *For every label $l \in \mathsf{alph}(P_1) \cap \mathsf{alph}(P_2)$, no bound event of T_1 is labelled l, and if $e \in \mathsf{fe}(P_1)$ is labelled l, then $e \in \mathsf{fe}(P_2)$.*

It's straightforward to verify that non-invasiveness is decidable and that R_1 is non-invasive for P, whereas R_2 is not for P_2 (because of the exclusion of bm).

It takes more to prove that non-invasiveness guarantees conservative adaptations. We first observe that transitions do not introduce new constraints or effects on free events.

Lemma 2 (Transitions reflect relational sub-terms). *If $[M]\,T \xrightarrow{\lambda} T'$ and $T' = C'(e\ \mathcal{R}\ f)$, then there exists a context $C(-,-)$ s.t. $T = C(e\ \mathcal{R}\ f)$ with f free in C' iff it is in C.*

Proof. Easy induction on the derivation of the transition.

Next we prove that for processes that are composed of two processes, the marking can be canonically separated in the three disjoint parts: The events only occurring in the first process, the events that are shared, and the events only occurring in the second process.

Definition 7 (Separation of Processes). *Let $P = [M]\,T_1 \mid T_2$. A* separation *of P comprises disjoint markings M_1, M_2, S such that $M = M_1 \oplus S \oplus M_2$, that $\mathsf{fe}(T_1) \cap \mathsf{fe}(T_2) \subseteq \mathsf{dom}(S)$, and that $\mathsf{fe}(T_i) \setminus \mathsf{fe}(T_{3-i}) \subseteq \mathsf{dom}(M_i)$. A process $[M_1 \oplus S \oplus M_2]\,T_1 \mid T_2$ is* separated *iff M_1, M_2, S is a separation.*

Lemma 3 (Canonical Separation). *Let $P_1 = [M_1]\,T_1$ and $P_2 = [M_2]\,T_2$ with $P_1 \oplus P_2$ defined. Then there exists a unique separation N_1, N_2, S of $P_1 \oplus P_2$ satisfying $\mathsf{dom}(N_i) = \mathsf{dom}(M_i) \setminus \mathsf{dom}(M_{3-i})$ and $\mathsf{dom}(S) = \mathsf{dom}(M_1) \cap \mathsf{dom}(M_2)$. We call this separation the* canonical *separation of $P_1 \oplus P_2$.*

Lemma 4. *If a process $[M]\,T_1 \mid T_2$ with canonical separation $M_1 \oplus S \oplus M_2$ has a transition*

$$[M]\,T_1 \mid T_2 \xrightarrow{\lambda} T' \quad \text{or} \quad [M]\,T_1 \mid T_2 \xrightarrow{\gamma} [M']\,T'$$

then the following holds:

1. *For some T_1', T_2' we have $T' = T_1' \mid T_2'$.*
2. *There exists a unique separation M_1', M_2', S' of M' with $\mathsf{dom}(M_i) = \mathsf{dom}(M_i')$ and $\mathsf{dom}(S) = \mathsf{dom}(S')$.*
3. *This separation satisfies $\mathsf{alph}([M_i \oplus S]\ T_i) = \mathsf{alph}([M_i' \oplus S']\ T_i')$*
4. *If the original separation was canonical for $P_1 = [M_1 \oplus S]\ T_1$ and $P_2 = [S \oplus M_2]\ T_2$, then so is M_1', M_2', S' for $P_1' = [M_1' \oplus S']\ T_1'$ and $P_2' = [S' \oplus M_2']\ T_2'$.*
5. *If P non-invasive for Q, then also P' non-invasive for Q'.*

Proof. Note that only the rules [PAR] and [PAR-2] allows term transitions for a term on the form $T_1 \mid T_2$; part 1 is then immediate by inspection of these rules; and part 2 and 3 follows from Lemma 1. Part 4 is then immediate from parts 2 and 3. Part 5 follows (1) by Lemma 2 and (2) by parts (2–4) and Lemma 1.

We will need the following auxiliary ordering on markings with identical domains: Smaller markings have more restless events.

Definition 8. *We order states $(h, i, r) \sqsubseteq (h', i', r')$ iff $h = h'$, $i = i'$ and $r' = \mathsf{t}$ implies $r = \mathsf{t}$. We order markings $M \sqsubseteq N$ point-wise when $\mathsf{dom}(M) = \mathsf{dom}(N)$.*

Lemma 5. *If $M \sqsubseteq N$ and both $[M]\ T$ and $[N]\ T$ are processes, then:*

1. *$[M]\ T \vdash e : \delta$ iff $[N]\ T \vdash e : \delta$;*
2. *$[M]\ T \xrightarrow{\lambda} T'$ iff $[N]\ T \xrightarrow{\lambda} T'$; and*
3. *For every process transition $[M]\ T \xrightarrow{\gamma} [M']\ T'$, there exists a unique N' s.t. $[N]\ T \xrightarrow{\gamma} [N']\ T'$. This N' satisfies $M' \sqsubseteq N'$.*

Proof. Part 1 is immediate by Definition of "$\vdash$". Part 2 then follows by induction on the derivation of the term transition, using part 1 in the base case [INTRO]. Part 3 follows by cases on the process transition rules [EFFECT] and [EFFECT-2], observing that for any $M \sqsubseteq N$ and any event or effect x, $x \cdot M \sqsubseteq x \cdot N$.

Lemma 6. *Both term and process transitions are unique in the following sense:*

1. *If $[M]\ T \xrightarrow{\gamma:\delta} T'$ and $[M]\ T \xrightarrow{\gamma:\delta'} T''$ then $\delta = \delta'$ and $T' = T''$.*
2. *If $P \xrightarrow{\gamma} Q$ and $P \xrightarrow{\gamma} Q'$ then $Q = Q'$.*

Proof. (1) By induction on the derivation of the transition. For the base case, [INTRO], by assumption we have

$$[M \oplus N]\ T \xrightarrow{e:\delta} T \qquad \text{and} \qquad [M' \oplus N']\ T \xrightarrow{e:\delta'} T\ ,$$

with $M \oplus N = M' \oplus N'$ and $[M]\ T \vdash e : \delta$ and $[M']\ T \vdash e : \delta'$. We now find by cases on T and inspection of the rules in Figure 4 that $M = M'$ and $\delta = \delta'$.

The cases [PAR], [PAR-2], and [REP] cases are straightforward; we exemplify with [PAR-2]. Suppose $[M]\ T \mid U \xrightarrow{\nu e:\delta_1} T_1$ and $[M]\ T \mid U \xrightarrow{\nu e:\delta_2} T_2$. By [PAR-2] we must have $T_1 = T_1' \mid U$ and $T_2 = T_2' \mid U$, and moreover $[M]\ T \xrightarrow{\nu e:\delta_1} T_1'$ and $[M]\ T \xrightarrow{\nu e:\delta_2} T_2'$. But then by IH $\delta_1 = \delta_2$ and $T_1' = T_2'$ whence $T_1 = T_2$.

Finally, [LOCAL]. Suppose

$$[M]\ (\nu f : \Phi)\ T \xrightarrow{\gamma:\delta_1} T_1 \qquad \text{and} \qquad [M]\ (\nu f : \Phi)\ T \xrightarrow{\gamma:\delta_2} T_2\ .$$

By [LOCAL] we must have

$$[M, f : \Phi]\ T \xrightarrow{e:\delta_1} T_1' \qquad \text{and} \qquad [M, f : \Phi]\ T \xrightarrow{e:\delta_2} T_2'\ ,$$

with $T_1 = (\nu f : \Phi_1)\ T_1'$ and $T_2 = (\nu f : \Phi_2)\ T_2'$. By IH $\delta_1 = \delta_2$ and $T_1' = T_2'$. It remains to prove that also $\Phi_1 = \Phi_2$. But again by [LOCAL] we have $f : \Phi_1 = (e : \delta_1) \cdot (f : \Phi) = (e : \delta_2) \cdot (f : \Phi) = f : \Phi_2$.

(2) Straightforward by inspection of the rules [EFFECT] and [EFFECT-2] using part (1) of this Lemma.

Lemma 7 (Weakening). *Suppose* $[M \oplus N]\ T \xrightarrow{\lambda} T'$. *If* $\lambda = e : \delta$ *and* $\mathsf{fe}(T) \cup \{e\}$ *is disjoint from* $\mathsf{dom}(N)$*, or* $\lambda = \nu e : \delta$ *and* $\mathsf{fe}(T)$ *is disjoint from* $\mathsf{dom}(N)$*, then also* $[M]\ T \xrightarrow{\lambda} T'$*.*

Proof. By induction on the derivation of the transition.

For [INTRO], note that we must have $\lambda = e : \delta$ and for some M', N' with $M' \oplus N' = M \oplus N$ that

$$[M \oplus N]\ T \xrightarrow{e:\delta} T' \qquad \text{and} \qquad [M']\ T \vdash e : \delta$$

By inspection of the rules for the enabling relation in Figure 4 we find that $\mathsf{dom}(M') \subseteq \mathsf{fe}(T) \cup \{e\}$ and so $\mathsf{dom}(M')$ disjoint from $\mathsf{dom}(N)$ and so $M = M' \oplus M''$ for some M'', whence $[M]\ T \xrightarrow{\lambda} T'$.

For [PAR] we have for some δ_1, δ_2 that $\lambda = e : \delta_1 \oplus \delta_2$ with

$$[M \oplus N]\ T_1 \xrightarrow{e:\delta_1} T_1' \qquad \text{and} \qquad [M \oplus N]\ T_2 \xrightarrow{e:\delta_2} T_2'$$

and $\mathsf{fe}(T_1 \mid T_2) \cup \{e\}$ disjoint from $\mathsf{dom}(N)$, so also $\mathsf{fe}(T_1) \cup \{e\}$ and $\mathsf{fe}(T_2) \cup \{e\}$ disjoint from $\mathsf{dom}(N)$. By IH we find then transitions

$$[M]\ T_1 \xrightarrow{e:\delta_1} T_1' \qquad \text{and} \qquad [M]\ T_2 \xrightarrow{e:\delta_2} T_2'$$

establishing by [PAR] a transition $[M]\ T_1 \mid T_2 \xrightarrow{e:\delta_1 \oplus \delta_2} T_1' \mid T_2'$.

For [LOCAL] we are given a transition

$$[M \oplus N]\ (\nu f : \Phi)\ T \xrightarrow{\gamma(\delta \backslash f)} (\nu f : \Phi')\ T'\ .$$

such that for some e

$$[M \oplus N, f : \Phi]\ T \xrightarrow{e:\delta} T' \qquad \text{and} \qquad f : \Phi' = (e : \delta) \cdot f : \Phi$$

and either $e = f$ and $\gamma = \nu e$ or $\gamma = e$. In the former case, we have by assumption $\mathsf{fe}(T) = \mathsf{fe}((\nu f : \Phi)\ T)$ disjoint from $\mathsf{dom}(N)$ and by the bound variable convention we may assume $e = f$ also not in the domain of $\mathsf{dom}(N)$. Hence $\mathsf{fe}(T) \cup \{e = f\}$ also disjoint from N and by IH we have $[M, f : \Phi]\ T \xrightarrow{e:\delta} T'$ which by [LOCAL] yields the requisite transition. In the latter case, we have because $\gamma = e$ that $\mathsf{fe}(T) \cup \{e\} = \mathsf{fe}(f\{\Phi\}T) \cup \{e\}$ disjoint from N and again by IH we find the requisite transition.

Finally, the cases [REP] and [PAR-2] are straightforward applications of IH, noting for the former that $\mathsf{fe}(T) = \mathsf{fe}(e\{T\})$ and for the latter that $\mathsf{fe}(T) \subseteq \mathsf{fe}(T \mid U)$ and so in both cases disjointness with N is preserved as we move to the hypothesis.

Lemma 8. *If $[M]\ T \xrightarrow{\gamma:\delta} T'$ with $\delta = (X, I, R)$ then $e \in X$ resp. $e \in I$ implies $T = C(f \rightarrow\!\% \ e)$ resp. $T = C(f \rightarrow\!+ \ e)$ with e not bound in $C(-)$.*

Proof. Easy induction on the derivation of the transition.

Lemma 9. *Let P be non-invasive for Q, and suppose M_1, S, M_2 is the canonical separation of $P \oplus Q = [M_1 \oplus S \oplus M_2]\ T_1 \mid T_2$. If also $[M_1 \oplus S \oplus M_2]\ T_1 \xrightarrow{\gamma:\delta} T_1'$ with $\delta = (X, I, R)$, then X, I are both disjoint from $\mathsf{fe}(Q)$.*

Proof. Immediate from the Definition of non-invasiveness and Lemma 8.

Lemma 10. *Let P be non-invasive for Q, and suppose M_1, S, M_2 is the canonical separation of $P \oplus Q = [M_1 \oplus S \oplus M_2]\ T_1 \mid T_2$. If also $[M_1 \oplus S \oplus M_2]\ T_1 \mid T_2 \xrightarrow{\gamma:\delta} T_1' \mid T_2'$ then the following are true.*

1. *If $\ell(\gamma) \in \mathsf{alph}(Q)$ then for some δ' we have $[S \oplus M_2]\ T_2 \xrightarrow{\gamma:\delta'} T_2'$ and $(\gamma : \delta) \cdot (S \oplus M_2) \sqsubseteq (\gamma : \delta') \cdot (S \oplus M_2)$.*
2. *If $\ell(\gamma) \notin \mathsf{alph}(Q)$ then $(\gamma : \delta) \cdot (S \oplus M_2) \sqsubseteq S \oplus M_2$.*

Proof. We proceed by cases on γ; suppose first $\gamma = \nu e$. If νe is a binder of T_2, we must have $\ell(\gamma) \in \mathsf{alph}(Q)$ and the transition must arise by (the rule symmetric to) [PAR-2]. By definition of canonical separation we have $\mathsf{fe}(T_2)$ disjoint from $\mathsf{dom}(M_1)$ and so by Lemma 7 we find a transition $[S \oplus M_2]\ T_2 \xrightarrow{\nu e:\delta} T_2'$, altogether establishing (1). If instead νe is a binder of T_1, we must have $\ell(\gamma) \notin \mathsf{alph}(Q)$ lest non-invasiveness be contradicted. In that case we must have a transition

$$[M_1 \oplus S \oplus M_2]\ T_1 \xrightarrow{\nu e:\delta} T_1'$$

by Lemma 9 we find $(\gamma : \delta) \cdot (S \oplus M_2) \sqsubseteq S \oplus M_2$.

Suppose instead $\gamma = e$. In this case the transition must be derived by [PAR], and so by Lemma 6 there exists unique δ_1, δ_2 such that

$$[M_1 \oplus S \oplus M_2]\ T_1 \xrightarrow{e:\delta_1} T_1' \qquad \text{and} \qquad [M_1 \oplus S \oplus M_2]\ T_2 \xrightarrow{e:\delta_2} T_2'$$

Suppose for (1) that $\ell(e) \in \mathsf{alph}(Q)$. By non-invasiveness and canonicity of separation we then have that $e \notin \mathsf{dom}(M_1)$ and that $\mathsf{fe}(T_2)$ is disjoint from $\mathsf{dom}(M_1)$, and so by Lemma 7 we have a transition

$$[S \oplus M_2]\ T_2 \xrightarrow{e:\delta_2} T_2'$$

By Lemma 9 it now follows that $(e : \delta_1 \oplus \delta_2) \cdot (S \oplus M_2) \sqsubseteq (e : \delta_2) \cdot (S \oplus M_2)$. Suppose instead for (2) that $\ell(e) \notin \mathsf{alph}(Q)$. It follows that $e \notin \mathsf{fe}(T_2)$, and so $\delta_2 = (\emptyset, \emptyset, \emptyset)$. We now find $(\gamma : \delta) \cdot (S \oplus M_2) \sqsubseteq S \oplus M_2$ by Lemmas 7 and 9.

Lemma 11. *Let P be non-invasive for Q, and suppose M_1, S, M_2 is the canonical separation for $P \oplus Q = [M_1 \oplus S \oplus M_2]\ T_1 \mid T_2$. If also $[M_1 \oplus S \oplus M_2]\ T_1 \mid T_2 \xrightarrow{\gamma} [M_1' \oplus S' \oplus M_2']\ T_1' \mid T_2' = R$ where the latter is also a canonical separated then the following are true.*

1. *If $\ell(\gamma) \in \mathsf{alph}(Q)$ then $[S \oplus M_2]\ T_2 \xrightarrow{\gamma} [N]\ T_2'$ where $S' \oplus M_2' \sqsubseteq N$.*
2. *If $\ell(\gamma) \notin \mathsf{alph}(Q)$ then $S' \oplus M_2' \sqsubseteq S \oplus M_2$.*

Proof. Immediate from the preceding Lemma and rules [EFFECT] and [EFFECT-2].

We are now finally ready for the first main technical result of the paper: That the syntactically decidable non-invasiveness property implies conservation.

Theorem 2. *If P is non-invasive for Q then P is conservative for Q.*

Proof. Let $M = M_1 \oplus S \oplus M_2$ be the canonical separation of $P \oplus Q$, and consider a finite or infinite run of $R_0 = P \oplus Q = [M]\ T_1 \mid T_2$:

$$R_0 \xrightarrow{\gamma_0} R_1 \xrightarrow{\gamma_1} \dots$$

By induction on i using Lemma 4, we can write each R_i as a canonically separated process

$$R_i = [M_1^i \oplus S^i \oplus M_2^i]\ T_1^i \mid T_2^i = ([M_1^i \oplus S^i]\ T_1^i) \oplus ([M_2^i \oplus S^i]\ T_2^i)$$

where $\mathsf{alph}([M_1^i \oplus S^i]\ T_1^i) = \mathsf{alph}(P)$ and $\mathsf{alph}([S^i \oplus M_2^i]\ T_2^i) = \mathsf{alph}(Q)$, and $[M_1^i]\ T_1^i$ is non-invasive for $[M_2^i]\ T_2^i$. We prove by induction that there exists a sequence N^i satisfying (a) $N^0 = S^0 \oplus M_2^0$, (b) $S^i \oplus M_2^i \sqsubseteq N^i$, and (c)

$$\begin{aligned} N^{i+1} &= N^i && \text{when } \ell(\gamma_i) \notin \mathsf{alph}(Q) \\ [N^i]\ T_2^i \xrightarrow{\gamma_i} [N^{i+1}]\ T_2^{i+1} & && \text{when } \ell(\gamma_i) \in \mathsf{alph}(Q) \end{aligned}$$

The Theorem then follows. We have immediately N^0, obtaining (a). For (b) and (c), consider some $i > 0$, and assume first $\ell(\gamma_i) \notin \mathsf{alph}(Q)$. By Lemma 11, Part 2, we then have $S^{i+1} \oplus M_{i+1} \sqsubseteq S^i \oplus M_i \sqsubseteq N_i = N_{i+1}$, obtaining (b) and (c). Assume instead $\ell(\gamma_i) \in \mathsf{alph}(Q)$. Then take $N^{i+1} = N$ where N is given by Lemma 11, Part 1, immediately obtaining (b) and (c).

5 Turing completeness of DCR*

In this section we show that DCR* has the full power of Turing machines by reduction from the Halting Problem for Minsky machines [28]. This is in contrast to the fact that the DCR process language of Section 2 can be shown (essentially via encoding to and from Büchi-automata) to characterise exactly languages that are the union of a regular and an ω-regular language.[4]

A Minsky machine $m = (R_1, R_2, P, c)$ comprises two unbounded *registers* R_1, R_2; a *program* P, which is a list of pairs of addresses and instructions; and a *program counter* c, giving the address of the current instruction. It has the following instruction set.

$\texttt{inc}(i, a)$	Add 1 to the contents of register i. Proceed to a.
$\texttt{decjz}(i, a, b)$	If register i is zero, proceed to a. Otherwise subtract 1 from register i and proceed to b.
$\texttt{halt}$	Halt execution (wlog assumed to appear exactly once).

We construct, given a Minsky machine m, a term $\mathsf{t}(m)$ and a marking $\mathsf{m}(m)$. We model machine instructions as events. To maintain execution order, we model program addresses explicitly as events a. These events serve only to constrain the execution of other events; they should not themselves happen, and we prevent them from doing so with a condition $a \rightarrow\!\bullet\, a$ for each a. By making each instruction event e conditional on its program point a, $a \rightarrow\!\bullet\, e$, we ensure that *only if a is excluded may e happen.* To move the program counter from a to b, we re-include a and exclude b. We define a shorthand $\mathsf{insn}(e, a, b)$ for an instruction event e at program point a proceeding to program point b as follows:

$$\mathsf{insn}(e, a, b) = a \rightarrow\!\bullet\, e \mid a \mathrel{+\!\!\leftarrow} e \mid b \mathrel{\%\!\!\leftarrow} e$$

Now, registers. We model each $a : \texttt{decjz}(i, b, c)$ by two events: one, $\texttt{decjz}^a$, which can happen only when the register is zero, and a second, $\texttt{decjn}^a$, which can happen only when it is not. Then we model increments by making each increment reproductive, replicating a new copy of $\texttt{decjn}^a$ for every decrement instruction $a : \texttt{decjz}(i, b, c)$ in P. The copies produced by a single increment represents the *opportunity* for exactly one of these instructions to decrement. Thus, we make the copies in a single increment exclude each other. To make sure that $\texttt{decjz}^a$ cannot happen if the register is non-zero, that is, if no $\texttt{decjn}^a$ is present, we make the latter a condition of the former: $\texttt{decjn}^a \rightarrow\!\bullet\, \texttt{decjz}^a$. Altogether, the term for one increment is constructed by the following function. (We write $(N_{i \in I} x_i : M)$ for $(\nu x_{i_1} : M) \ldots (\nu x_{i_n} : M)$ when $I = \{i_1, \ldots, i_n\}$.)

$$\begin{aligned}\mathsf{one}(i) = \Big(\mathop{\text{N}}_{a:\texttt{decjz}(i,c,d)} \texttt{decjn}^a : (\mathsf{f}, \mathsf{t}, \mathsf{f}) \Big) \prod_{a:\texttt{decjz}(i,c,d)} \Big(&\mathsf{insn}(\texttt{decjn}^a, a, d) \mid \\ &\texttt{decjn}^a \rightarrow\!\bullet\, \texttt{decjz}^a \mid \prod_{a':\texttt{decjz}(i,b',c')} \texttt{decjn}^{a'} \mathrel{\%\!\!\leftarrow} \texttt{decjn}^a \Big)\end{aligned}$$

Adding one to a register i is accomplished by making a new copy of $\mathsf{one}(i)$.

$$\mathsf{inc}(a, i, b) = \mathsf{insn}(\texttt{inc}^a, a, b) \mid \texttt{inc}^a\{\mathsf{one}(i)\}$$

[4] This result is similar to the expressiveness result for DCR Graphs sketched in [30], and can be found in the full version [11].

We put it all together and define $\mathsf{t}(m)$ for a Minsky machine $m = (R_1, R_2, P, c)$.

$$\mathsf{t}(m) = \prod_{a:\texttt{inc}(i,b)\in P} \mathsf{inc}(a,i,b) \mid \prod_{a:\texttt{decjz}(i,b,c)\in P} \mathsf{insn}(\texttt{decjz}^a, a, b)$$
$$\mid \prod_{a:\texttt{halt}\in P} a \rightarrow\!\bullet\ \texttt{halt} \mid \prod_{a:I\in P} a \rightarrow\!\bullet\ a \mid \prod_{i<R_1} \mathsf{one}(1) \mid \prod_{i<R_2} \mathsf{one}(2)$$

Finally, the marking $\mathsf{m}(m)$ is given below. (Recall that c is the program counter.)

	c	a when $a \neq c$	$\texttt{decjz}^a$	$\texttt{inc}^a$	$\texttt{halt}$
Happened	f	f	f	f	f
Included	f	t	t	t	t
Restless	f	f	f	f	t

Example 7. As an example, let us consider a Minsky machine adding the contents of register 2 to register 1. We'll consider the machine $(0, 1, P, 1)$, where P is the program:

$1 : \texttt{decjz}(2, 3, 2)$
$2 : \texttt{inc}(1, 1)$
$3 : \texttt{halt}$

Applying the above construction, we get the following term (split out in a table for readability).

$\prod_{a:\texttt{inc}(i,b)\in P} \mathsf{inc}(a,i,b)$	$\prod_{a:\texttt{decjz}(i,b,c)\in P} \mathsf{insn}(\texttt{decjz}^a, a, b)$
$2 \rightarrow\!\bullet\ \texttt{inc}^2$ $2\ \%\!\leftarrow\ \texttt{inc}^2$ $1\ +\!\leftarrow\ \texttt{inc}^2$ $\texttt{inc}^2\{0\}$	$1 \rightarrow\!\bullet\ \texttt{decjz}^1$ $1\ +\!\leftarrow\ \texttt{decjz}^1$ $3\ \%\!\leftarrow\ \texttt{decjz}^1$

$\prod_{a:\texttt{halt}\in P} a \rightarrow\!\bullet\ \texttt{halt}$	$\prod_{a:I\in P} a \rightarrow\!\bullet\ a$	$\prod_{i<R_1} \mathsf{one}(1)$	$\prod_{i<R_2} \mathsf{one}(2)$
$3 \rightarrow\!\bullet\ \texttt{halt}$	$1 \rightarrow\!\bullet\ 1$ $2 \rightarrow\!\bullet\ 2$ $3 \rightarrow\!\bullet\ 3$	0	$(\nu\texttt{decjn}^1 : (\mathsf{f},\mathsf{t},\mathsf{f}))$ $1 \rightarrow\!\bullet\ \texttt{decjn}^1$ $1\ +\!\leftarrow\ \texttt{decjn}^1$ $2\ \%\!\leftarrow\ \texttt{decjn}^1$ $\texttt{decjn}^1 \rightarrow\!\bullet\ \texttt{decjz}^1$ $\texttt{decjn}^1\ \%\!\leftarrow\ \texttt{decjn}^1$

We emphasise that in the column $\Pi_{i<R_2}\mathsf{one}(2)$, all instances of $\texttt{decjn}^1$ are within the scope of the binder and thus local. □

Theorem 3. *A Minsky machine m halts iff $[\mathsf{m}(m)]\ \mathsf{t}(m)$ has an accepting run.*

Proof. (Sketch) The proof is based on a bisimulation relation between finite execution traces of the Minsky machine m and reachable markings of the encoding $[\mathsf{m}(m)]\ \mathsf{t}(m)$. First we observe that in every reachable marking of $[\mathsf{m}(m)]\ \mathsf{t}(m)$

exactly one of the program address events will be included and exactly one event is enabled. The bisimulation relation will relate an execution trace of the Minsky machine ending in address j to a marking in which that event is excluded. Next we prove that for every pair, the machine can perform an instruction iff the encoding can execute the corresponding event, and that the form of the process $t(m)$ is preserved as well as the global marking $m(m)$, except that instruction events are being recorded as executed (and excluded in the case of `decjn`) is preserved by steps. It follows that the restless `halt` event can be eventually executed if and only if the machine can execute the halt command.

6 Conclusion, Related and Future Work

We first presented the constraint-based Dynamic Condition Response (DCR) language, which generalises event structures using the notions of dynamic inclusion and exclusion of events, and conditions and responses introduced in our work on DCR Graphs. We provided a *compositional semantics* for the DCR language and noted that it provides a term language for (modular definition of) DCR Graphs. We then extended the DCR language to the *DCR* language*, allowing dynamic creation of sub processes with (fresh) local events. This extension was inspired by the recent extension of DCR Graphs to *hierarchical* DCR Graphs [12], which is motivated in practice and exemplified in the paper by a grant application process, in which each grant application needs an independent decision of accept or reject. As the next contribution we introduced a notion of *run-time adaptation by composition* for DCR* and formalised when an adaptation is *conservative*, i.e. preserves the constraints of the adapted process. As a key technical result, we provided a decidable approximation to conservative adaptation referred to as *non-intrusive* adaptations. We illustrated that non-intrusive adaptation is useful in practice by showing examples, and also showing how an intrusive adaptation could be replaced by a non-intrusive one. As the final technical contribution we proved that the DCR* language is Turing complete. This also means that the notion of conservative adaptation is undecidable, and thus establish the importance of the decidable approximation. The Turing completeness result should be seen in contrast to the fact that the DCR process language corresponds to the DCR graphs model, which is known to characterise exactly languages that are the union of a regular and an ω-regular language.

Related Work. As the DCR language is essentially a term language for DCR Graphs [30,36,18]; which in turn is a descendant of event structures that are closely related to Petri Nets, DCR processes are also naturally related to Petri Nets. Petri Nets have been extended to allow modular definition (e.g. via shared transitions [27]) and to represent infinite computations and ω-regular languages (e.g. Büchi Net [15]). However, as mentioned in the introduction, Petri net introduces the intentional construct of *places* marked with *tokens*, as opposed to event structures and DCR* processes only relying on causal and conflict relations between events. A number of variants of event structures with asymmetric conflict relation related to the asymmetric exclude relation of DCR processes have been proposed, including extended bundle event structures [25,17], dual event structures [24,26], asymmetric event structures [6], and precursor event structures [16]. Automata based models like Event automata [33] and local event structures [21] also allows for asymmetric conflicts, but they introduce the notion of states explicitly and do not express the notion of causality and conflicts as relations between events. Besides the early work on restless events in [37], we are not aware of other published work generalising event-structures to be able to express liveness properties or to distinguish between events that *may* and events that *must eventually be executed*. Reproductive events of the DCR* process language relate to replication in process calculi and higher-order Petri nets [23]. We are not aware of other work combining such higher-order features and liveness.

Run-time adaptation is emerging as an important topic in many branches of computer science. Much previous work dealt with imperative process notations such as Petri nets [35] and process calculi [5,8,9] requiring predefined adaptation points and often dealing with adaptations via higher-order primitives. In contrast, adaptation in DCR* is dealt with by composition, which due to the declarative nature allow for cross-cutting adaptations without the need for prespecified adaptation points.

Within the BPM community, interest in declarative languages arose with the work on Declare [4,1]. Declare is in essence a template language, consisting of typical constraints encountered in business processes which are given a formal semantics by mapping them to LTL formulae. In contrast to DCR Graphs, where there is a close connection between the design-time and run-time of a model, the execution and analysis of Declare models relies on a mapping to LTL and then to automata (in fact, Declare semantics only consider finite executions). Additionally Declare has a relatively large set of basic constraints, for which the formal expressiveness is still an open question, but is clearly limited to at most that of LTL, while DCR Graphs with only 4 basic constraints offers the full expressiveness of regular and ω-regular languages. As another alternative [29] provides a mapping from Declare constraints to the CLIMB Computational Logic-based language, which allows the use of its reasoning techniques for support and verification of Declare processes at design- and run-time.

More recently the Guard-Stage-Milestone (GSM) approach [22] has been developed at IBM Research, which offers a data-centric business process modelling notation. The notation consists of stages, which have guards controlling when the stage may start, and milestones controlling when and how a stage may close. While the guards and milestones give a declarative flavour to the notation, it is still mainly data-centric, whereas DCR is mainly focussed on describing the events (or activities) of the system and the relations between them.

While imperative process models such as BPMN [32] have supported dynamic sub-processes for some time now, they are only recently being studied for declarative languages [39]. In most cases, there is a tendency to emphasise sub-processes that do not have independent life cycles, that is, a sub processes is spawned, and must run to completion before the super process may resume; this includes [39]. Interestingly, it is noted in *ibid.* that extending the model with sub-processes seems to increase its expressive power; we formally confirm that supposition here, finding DCR graphs with sub-processes to be Turing complete.

Future work. The work opens several interesting paths for future work. Firstly, the DCR* processes as defined only interact via shared events. We are currently working on adding interaction between concurrent events, labelled with send and receive labels as found e.g. in the π-calculus, thereby lifting the results of the present paper to π-like languages. In relation to that, we also work on exploring *independence model* (also referred to as true-concurrency) semantics of DCR* processes, which benefits from the explicit definition of events in DCR* processes. An independence model semantics could support partial order verification techniques and refinement of events. We have also initiated work on exploiting the idea of responses and restless events in the domain of behavioural types [13] and run-time monitoring [30], which provides an avenue for analysing infinite-state systems. Along another path, we have in [10] initiated an exploration of branching simulations between transition systems with responses, a generalisation of the LTS-with-acceptance of Def. 1, which we believe will generalise modal transition systems and refinement to models expressing more general liveness properties. The next step from that exploration will be to investigate branching bisimulation congruences. The DCR* process language should also be formally related to the hierarchical DCR Graphs introduced in [12] and the relation to modular [27] and higher-order Petri Nets [23] should be investigated. We conjecture that DCR* and hierarchical DCR Graphs are equivalent, but the mapping is not as straightforward as the one between basic DCR processes and Graphs. Finally, time constraints and more general adaptations as initiated in [20,31], e.g. allowing to remove constraints and events should be further investigated.

Acknowledgments. We thank the anonymous reviewers for helpful comments.

References

1. van der Aalst, W., Pesic, M., Schonenberg, H., Westergaard, M., Maggi, F.M.: Declare. Webpage (2010), `http://www.win.tue.nl/declare/`
2. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2678, pp. 1–12. Springer (2003), `http://dx.doi.org/10.1007/3-540-44895-0_1`
4. van der Aalst, W.M., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer (2006)
5. Anderson, G., Rathke, J.: Dynamic software update for message passing programs. In: Jhala, R., Igarashi, A. (eds.) APLAS. Lecture Notes in Computer Science, vol. 7705, pp. 207–222. Springer (2012)
6. Baldan, P., Corradini, A., Montanari, U.: Contextual petri nets, asymmetric event structures, and processes. Information and Computation 171, 1–49 (2001)

7. Barthe, G., Pardo, A., Schneider, G. (eds.): Software Engineering and Formal Methods - 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings, LNCS, vol. 7041. Springer (2011)
8. Bravetti, M., Di Giusto, C., Pérez, J.A., Zavattaro, G.: Steps on the road to component evolvability. In: Proceedings of the 7th International Conference on Formal Aspects of Component Software. pp. 295–299. FACS'10 (2012), http://dx.doi.org/10.1007/978-3-642-27269-1_19
9. Bravetti, M., Giusto, C.D., Pérez, J.A., Zavattaro, G.: Adaptable processes. Logical Methods in Computer Science 8(4) (2012)
10. Carbone, M., Hildebrandt, T.T., Perrone, G., Wasowski, A.: Refinement for transition systems with responses. In: FIT. EPTCS, vol. 87, pp. 48–55 (2012)
11. Debois, S., Hildebrandt, T., Slaats, T.: Towards a foundation for modular run-time adaptable process-aware information systems (full version), http://www.itu.dk/~hilde/dcrpl-full.pdf
12. Debois, S., Hildebrandt, T.T., Slaats, T.: Hierarchical declarative modelling with refinement and sub-processes. In: Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8659, pp. 18–33. Springer (2014), http://dx.doi.org/10.1007/978-3-319-10172-9_2
13. Debois, S., Hildebrandt, T.T., Slaats, T., Yoshida, N.: Type checking liveness for collaborative processes with bounded and unbounded recursion. In: FORTE. Lecture Notes in Computer Science, vol. 8461, pp. 1–16. Springer (2014)
14. Debois, S., Hildebrandt, T., Marquard, M., Slaats, T.: A case for declarative process modelling: Agile development of a grant application system. In: 3nd International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (AdaptiveCM 2014) (2014)
15. Esparza, J., Melzer, S.: Model checking ltl using constraint programming. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997, Lecture Notes in Computer Science, vol. 1248, pp. 1–20. Springer Berlin Heidelberg (1997), http://dx.doi.org/10.1007/3-540-63139-9_26
16. Fecher, H., Majster-Cederbaum, M.: Event structures for arbitrary disruption. Fundam. Inf. 68(1-2), 103–130 (Apr 2005)
17. van Glabbeek, R., Vaandrager, F.: Bundle event structures and ccsp. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003 - Concurrency Theory, Lecture Notes in Computer Science, vol. 2761, pp. 57–71. Springer Berlin Heidelberg (2003), http://dx.doi.org/10.1007/978-3-540-45187-7_4
18. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES. EPTCS, vol. 69, pp. 59–73 (2010)
19. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: FSEN. LNCS, vol. 7141, pp. 343–350. Springer (2011)
20. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. J. Log. Algebr. Program. 82(5-7), 164–185 (2013), http://dx.doi.org/10.1016/j.jlap.2013.05.005
21. Hoogers, P., Kleijn, H., Thiagarajan, P.: An event structure semantics for general petri nets. Theoretical Computer Science 153(1–2), 129 – 170 (1996), http://www.sciencedirect.com/science/article/pii/0304397595001204
22. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Introducing the

guard-stage-milestone approach for specifying business entity lifecycles. In: WS-FM. LNCS, vol. 6551, pp. 1–24. Springer (2010)
23. Janneck, J.W., Esser, R.: Higher-order petri net modelling: Techniques and applications. In: Proceedings of the Conference on Application and Theory of Petri Nets: Formal Methods in Software Engineering and Defence Systems. pp. 17–25. CRPIT '02 (2002), http://dl.acm.org/citation.cfm?id=846335.846338
24. Katoen, J.P.: Quantitative and qualitative extensions of event structures. Ph.D. thesis, University of Twente, Enschede (April 1996)
25. Langerak, R.: Transformations and Semantics for LOTOS. Universiteit Twente (1992), http://books.google.dk/books?id=qB4EAgAACAAJ
26. Langerak, R., Brinksma, E., Katoen, J.P.: Causal ambiguity and partial orders in event structures. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR '97: Concurrency Theory, Lecture Notes in Computer Science, vol. 1243, pp. 317–331. Springer Berlin Heidelberg (1997)
27. Latvala, T., Mäkelä, M.: Ltl model checking for modular petri nets. In: Cortadella, J., Reisig, W. (eds.) Applications and Theory of Petri Nets 2004, Lecture Notes in Computer Science, vol. 3099, pp. 298–311. Springer Berlin Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-27793-4_17
28. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
29. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, Lecture Notes in Business Information Processing, vol. 56. Springer (2010)
30. Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (June 2012)
31. Mukkamala, R.R., Hildebrandt, T., Slaats, T.: Towards trustworthy adaptive case management with dynamic condition response graphs. In: EDOC. pp. 127–136. IEEE (2013)
32. Object Management Group BPMN Technical Committee: Business Process Model and Notation, version 2.0, http://www.omg.org/spec/BPMN/2.0/PDF
33. Pinna, G., Poigné, A.: On the nature of events: another perspective in concurrency. Theoretical Computer Science 138(2), 425 – 454 (1995), http://www.sciencedirect.com/science/article/pii/030439759400174H, meeting on the mathematical foundation of programing semantics
34. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
35. Sibertin-Blanc, C., Mauran, P., Padiou, G.: Safe Adaptation of Component Coordination. Proceedings of the Third International Workshop on Coordination and Adaption Techniques for Software Entities 189, 69–85 (juillet 2007), http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B75H1-4P5RTKV-6-1&_cdi=13109&_user=4373277&_orig=browse&_coverDate=07%252F17%252F2007&_sk=998109999&view=c&w
36. Slaats, T., Mukkamala, R.R., Hildebrandt, T.T., Marquard, M.: Exformatics declarative case management workflows as DCR graphs. In: BPM. LNCS, vol. 8094, pp. 339–354. Springer (2013)
37. Winskel, G.: Events in Computation. Ph.D. thesis, University of Edinburgh (1980)
38. Winskel, G.: Event structures. In: Advances in Petri Nets. LNCS, vol. 255, pp. 325–392. Springer (1986)
39. Zugal, S., Soffer, P., Pinggera, J., Weber, B.: Expressiveness and understandability considerations of hierarchy in declarative business process models. In: BMMDS/EMMSAD. Lecture Notes in Business Information Processing, vol. 113, pp. 167–181. Springer (2012)

8 Hybrid BPM Technologies

Declarative Modeling—An Academic Dream or the Future for BPM?

Hajo A. Reijers[1,2], Tijs Slaats[3,4], and Christian Stahl[1]

[1] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{H.A.Reijers, C.Stahl}@tue.nl
[2] Perceptive Software, Piet Joubertstraat 4, 7315 AV Apeldoorn, The Netherlands
[3] IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
TSlaats@itu.dk
[4] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark

Abstract. Declarative modeling has attracted much attention over the last years, resulting in the development of several academic declarative modeling techniques and tools. The absence of empirical evaluations on their use and usefulness, however, raises the question whether practitioners are attracted to using those techniques. In this paper, we present a study on *what practitioners think of declarative modeling.* We show that the practitioners we involved in this study are receptive to the idea of a *hybrid approach* combining imperative and declarative techniques, rather than making a full shift from the imperative to the declarative paradigm. Moreover, we report on requirements, use cases, limitations, and tool support of such a hybrid approach. Based on the gained insight, we propose a *research agenda* for the development of this novel modeling approach.

1 Introduction

Imperative modeling is currently the most prominent modeling paradigm in BPM. Imperative modeling techniques are implemented in almost every modeling tool, and many imperative modeling languages have been developed, most prominently, Event-Driven Process Chains (EPCs) and Business Process Modeling Notation. Imperative models take an *"inside-out"* approach; that is, every possible execution sequence must be modeled explicitly. As a consequence, imperative modeling may lead to over-specification and lack of flexibility, making it difficult to defer decisions at runtime and to change existing process models [21,2].

To overcome these shortcomings, *declarative modeling* approaches have been proposed [3]. In contrast to imperative approaches, declarative models take an *"outside-in"* approach. Instead of describing how the process has to work exactly, only the essential characteristics are described. To this end, constraints are specified that restrict the possible execution of activities.

Research on declarative modeling has gained increasing interest over the last years. Declarative languages, such as Declare [3] (formerly known as DecSerFlow), DCR Graphs [12] and SCIFF [14], have been developed. These languages have been integrated in academic and industrial modeling tools [24].

Beside the development of declarative techniques, also empirical research has been conducted to study the relation between imperative and declarative approaches [8,9,22,20]. It is well understood how to specify properties of a business process, but it is still not clear how to define a business process modeling languages that is understandable [8] on the one hand, and enables maintainability [9], expressiveness and modeling comfort, on the other hand.

To the best of our knowledge, there does not exist any studies that reflect on the question whether declarative techniques can be used in practise from a practitioner's standpoint. This raises a question, which has not been answered yet: *Do practitioners see opportunities to use declarative techniques?*

The contribution of this paper is to present *what practitioners think of declarative modeling*. In that way, we close the gap between research on declarative techniques and empirical investigations on declarative modeling. Our results are based on a workshop on declarative modeling with ten professionals from industry, including both consultants involved in modeling projects and developers of industrial modeling tools. During the workshop, we introduced declarative modeling techniques, performed two modeling assignments, and discussed the prospects of a declarative approach. The evaluation, both qualitative and quantitative, shows that practitioners see good opportunities for a *hybrid approach combining imperative and declarative techniques* while they are skeptical regarding a purely declarative approach. With the gained insight from the discussion, we present requirements on such a hybrid approach, use cases, limitations, and requirements concerning tool support. Shifting the focus from imperative and declarative modeling to a hybrid approach raises many research questions. Therefore, we propose a *research agenda* for the BPM community to make the hybrid approach work.

We continue with a brief introduction to Declare and DCR Graphs, two declarative approaches we used throughout the workshop. In Sect. 3, we describe the outline of the workshop and our evaluation method. The quantitative evaluation is described in Sect. 4, and Sect. 5 reports on the qualitative evaluation. In Sect. 6, we present our research agenda. We close with a conclusion and directions for future work.

2 Declare and DCR Graphs by Example

In this section, we briefly introduce two declarative modeling approaches, Declare [3,24] and DCR Graphs [12], using the following example of a document management system. To simplify the presentation, we restrict ourselves to the control flow dimension and do not consider data or resources.

Example 1. Every case of the document management system is initially *created* and eventually *closed*. For a created case, an arbitrary number of documents can

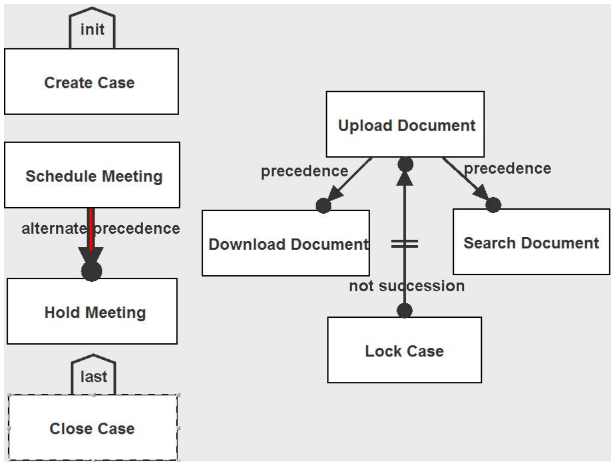


Fig. 1. Declare model of the document management system

be *uploaded*. An uploaded document can be *downloaded* or *searched*. At any time, a case can be *locked*. After locking a case, it is not possible to upload a document; still, uploaded documents can be downloaded and searched. Furthermore, in every case, meetings can be *held*. To hold a meeting, it has to be *(re-)scheduled*. Meetings can be rescheduled arbitrarily often, but it is not possible to schedule more than one meeting in advance.

2.1 Declare

A Declare model consists of activities and constraints. An activity is depicted as a rectangle and a constraint as a hyper-arc (i.e., a constraint connects one or more activities). From the specification, we identify eight activities which are highlighted in the description. Figure 1 shows the Declare model of the example. The *init* symbol on top of activity *Create Case* specifies that every case of the document management system starts with activity *Create Case*. Likewise, the *last* symbol on top of activity *Close Case* specifies that the final activity of every case of the document management system is *Close Case*.

There are three types of arcs in Fig. 1. Each arc type specifies one type of constraint. The precedence constraint, modeled as an arc from *Upload Document* to *Download Document* specifies that a document has to be uploaded before it can be downloaded. Likewise, we can only search a document once it has been uploaded (arc from *Upload Document* to *Search Document*). The second type of constraint is the not-succession constraint, which is modeled by an arc from *Lock Case* to *Upload Document*. It specifies that after a case has been locked, we cannot upload new documents. The third type of constraint, alternate precedence, is the arc from *Schedule Meeting* to *Hold Meeting*. It means that a meeting can only be held after it has been (re-)scheduled at least once. Moreover, after a meeting has been held, the next meeting has to be (re-)scheduled before

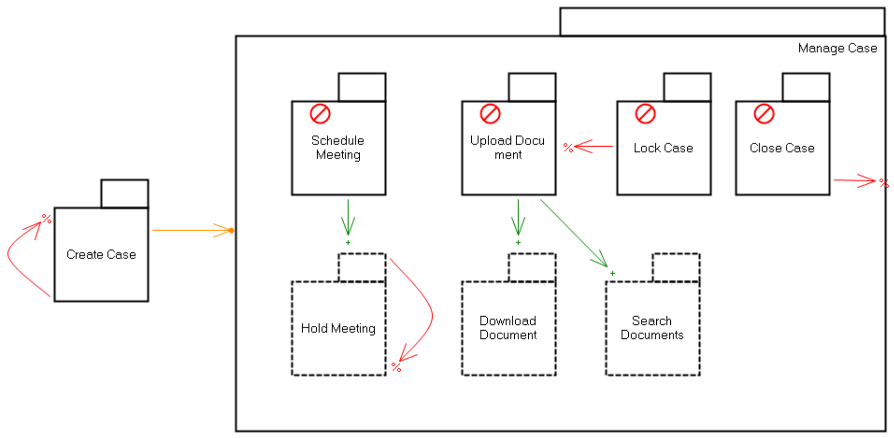


Fig. 2. DCR Graph model of the document management system

it can be held (i.e., activity *Hold Meeting* has to be followed by *Schedule Meeting* before *Hold Meeting* can be executed again).

As mentioned in the introduction, a declarative model only describes the essential characteristics of a process rather than how the process has to work exactly. For example, holding and (re-)scheduling meetings is independent from handling documents. Therefore the respective activities are not connected by arcs; that is, no constraint restricts their interplay. To execute the model in Fig. 1, one has to determine which activities are enabled by evaluating all constraints. Initially, it is the start activity, *Create Case*. After this activity is executed, any of the activities *Schedule Meeting*, *Upload Document*, *Lock Case* and *Close Case* can occur. A Declare model can be enacted and executed. The tool then computes the enabled transitions for every state [24].

2.2 DCR Graphs

A DCR Graph model consists of activities, relations, and a runtime marking. Activities are depicted as rectangles with an “ear” that can contain the roles which can execute the activity. Activities can be nested under super-activities, depicted by drawing an activity inside the rectangle of another activity, in which case any relation that applies to the super-activity, applies to all its sub-activities. Only the atomic activities (that do not contain any sub-activities of their own) are executable. The relations are drawn as arrows between activities.

Figure 2 shows the DCR model of the example. The first activity is *Create Case*, which should occur before all other activities can occur. We model this behaviour by the yellow *condition relation* from *Create Case* to the super-activity *Manage Case*, containing all other activities. The condition relation states that the second activity (in this case any sub-activity of *Manage Case*) can not occur before the first activity (in this case *Create Case*). We also require that *Create*

Case happens only once, which we model through the *dynamic exclusion relation* drawn as a red arrow with a percentage sign at the end. Through this relation *Create Case* excludes itself from the workflow when it is executed, meaning that it can not be executed anymore afterward. The next two activities are *Schedule Meeting* and *Hold Meeting*. We should always schedule a meeting before we can hold a meeting, but it might be the case that a meeting is rescheduled before it is held. We model this in the following way: *Hold Meeting* is *initially excluded*, meaning that at the start of the workflow it can not be executed before it is included. *Hold Meeting* is included by doing *Schedule Meeting*, modelled by the *dynamic inclusion relation*, drawn as a green arrow with a plus sign in the end. *Hold Meeting* excludes itself meaning that it can not be executed again before there has been a new occurrence of *Schedule Meeting*. The next three activities are *Upload Document*, *Download Document*, and *Search Documents*. We can not download or search documents before at least one document has been uploaded, therefore those activities are initially excluded and will be included by *Upload Document*. The case can also be locked through the activity *Lock Case*, which makes it impossible to upload further documents, therefore *Lock Case* excludes *Upload Document*. Finally we can close the case by executing the activity *Close Case*. We model this by having *Close Case* exclude the super-activity *Manage Case*. Because all activities are nested under *Manage Case*, *Close Case* will exclude all activities from the workflow.

The final two relations of DCR Graphs are not used in the example. First there is the response relation which states that one activity requires another activity to happen in the future, when this occurs we say that the second activity is a *pending response* and annotate it with an exclamation mark. A workflow is in an accepting state while there are no *included pending responses*, in case there are included pending responses these should be executed before the workflow can be closed. The second relation that is not shown is the milestone relation, it captures this accepting condition on the level of activities by stating that while some activity is a pending response, some other activity can not be executed.

We represent the runtime of a DCR Graph by showing which activities have been executed at least once before by drawing them with a green check-mark, showing which activities are pending responses by drawing them with a red exclamation mark and showing which activities are currently excluded by drawing them with a dashed line instead of a solid line. We call these three sets of activities the *marking* of the DCR Graph. Based on the marking we can determine which activities are enabled: Activities which are excluded (drawn with a dashed line) are not enabled and activities that are blocked by a condition and/or milestone relation are also not enabled. In the latter case, we show this by drawing a red stop-mark on the activity. In Fig. 2, one can see that the only initially enabled activity is *Create Case*. All other activities are either excluded (drawn with dashed lines) or blocked through the condition relation (drawn with a red stop-mark). We distinguish between being excluded and blocked by a condition/milestone relation because we consider these two as essentially different states of the activity: When it is blocked it is still a part of the workflow, but being stopped from executing. When it is excluded it is not considered as a part of the workflow at that time. This is also why only *included* pending responses will block the workflow from being closed.

2.3 Comparison

Figures 1 and 2 clearly illustrate the idea behind declarative modeling. The main difference between DCR Graphs and Declare is that the DCR Graph approach allows to define any constraint using the five basic relations, while one has to define many more constraint for Declare (some of them are logical combinations of simpler constraints). Also, Declare represents the runtime of a workflow by showing the state of the individual constraints—that is, which constraints are (possibly) satisfied, and which constraints are (possibly) violated. DCR Graphs, by contrast, represents the runtime of a workflow by showing which tasks have been executed at least once before, which tasks are pending as a response and should be done some time in the future, and which tasks are currently included in the workflow. While on infinite traces DCR Graphs are strictly more expressive than Declare, this has no impact on practical business process modeling: The processes under consideration typically produce finite traces.

3 Method

For our evaluation, we worked together with Perceptive Software, a provider of enterprise content management and BPM technologies. We invited both consultants, who engage with clients to model their processes and implement BPM suites using such models, and professionals who contribute to the development of the toolsets. We planned a single workshop that went through the four phases, as depicted in Fig. 3.

In the first phase (*Introduction*), we provided the participants with the motivation for organizing the workshop and gave them a generic introduction to declarative principles. After this phase, we split the groups into two sub-groups of equal size. We first randomly assigned half of the consultants to group 1 and the other half to group 2. We did the same for the developers after that. In this way, we ensured an even distribution of consultants and developers over the groups.

The second phase was specific for each group and consisted of a tutorial on the techniques under consideration (*Explanation*). In other words, one group received the tutorial on DCR Graphs and the other on Declare. The tutorials were provided by separate moderators for each group. Each moderator had deep expertise in the technique that he explained. The tutorials were synchronized beforehand between the moderators to guarantee a similar level of depth and the same duration.

Following up on the tutorials, each group received two assignments. These assignments were the same for both groups and required the participants to translate the assignment material into process models (*Modeling*). Clearly, the

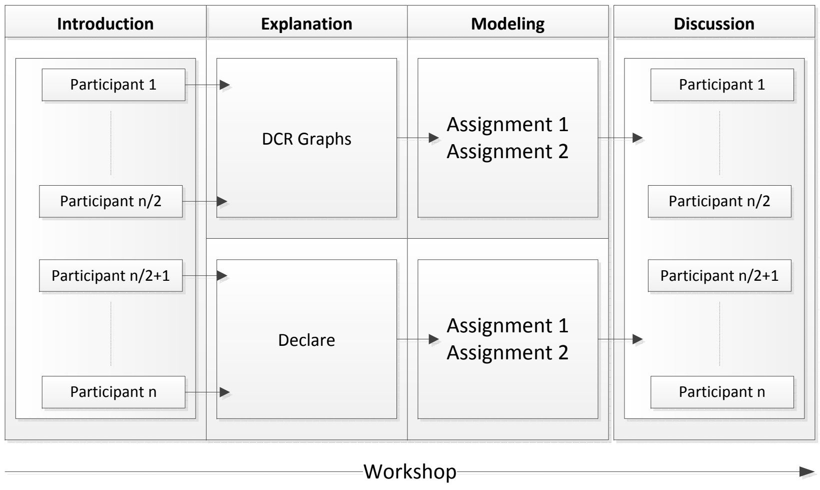


Fig. 3. Organization of the workshop

sub-group who received the tutorial on DCR Graphs used this technique; the other sub-group used Declare. The assignments can be found back at `http://www.win.tue.nl/ais/doku.php?id=research:declare:bpm2013`. As we were not so much interested in checking the correctness of the solutions but in transferring knowledge on the techniques to the participants, we encouraged them to work in pairs within each sub-group.

The final phase re-united the sub-groups (*Discussion*). During this phase, we first had the participants fill out a questionnaire on usefulness, ease of use, and intent to use as proposed by Moody [17]. The questionnaire can be used to get a broad-brush insight into the perceived quality of an IS design method, building on the concepts known from the Technology Acceptance Model as proposed by Davis [6]. We extended the questions with some more to gather demographic data on the group. The used questions can also be found at `http://www.win.tue.nl/ais/doku.php?id=research:declare:bpm2013`. After the questionnaire, we engaged in a semi-structured discussion with the group. This discussion was moderated by one of the authors, while the other authors took notes. The independently taken notes were used to reach consensus on how the participants reflected on the questions.

The insights that we gathered during the last phase of the evaluation with the questionnaire will be referred to as the *quantitative evaluation*, because the design of Davis' list allows for measuring the strength of the perceptions on ease of use, usefulness, and intent to use. Our insights on the modeling phase and the open part of the discussion phase will be dealt with as the *qualitative evaluation*, as they add a qualifying lens on the results. These respective evaluations will be discussed next.

4 Quantitative Evaluation

4.1 Demographics

Overall, ten professionals participated in the workshop. Of these, five are active as consultants, modeling processes at client sites and implementing process management software, while the other five are involved in different roles associated to the development of process modeling and workflow tools (product manager/architect/developer). For the entire group, the average number of years of experience in the BPM domain was more than 11 years. Of the ten participants, on a scale of 1 to 5, three considered themselves to have an intermediate expertise in process modeling (level=3), three to have an advanced level of expertise (level=4), and the remaining four people considered themselves to be experts (level=5). Finally, the participants indicated that on average they had each read close to 15 different process models in the preceding 12 months, while each had created or updated nearly four models on average in the same period. We are aware that the number of professionals is rather low. However, within a given time frame, we were choosing the day for which most professionals indicated their availability.

4.2 Validity and Reliability

Prior to performing an in-depth analysis of the data that had been gathered through the questionnaire, the validity and reliability of the empirical indicators were checked. We determined all correlations between the responses for questions that were used to measure to same construct (inter-item correlations) and identified no item that displayed a low convergent validity. In other words, the questions and their grouping to measure the constructs appeared valid. Next, we used Cronbach's alpha to test the reliability of the items to measure the various constructs. This is a test to check internal consistency of the questions. While there is no authoritative level for Cronbach's alpha, it is generally assumed that levels above 0.7 point at a good reliability of the items [18]. Adequate levels were established for *Perceived Usefulness* (0.743) and *Perceived Ease of Use* (0.826). However, *Intention to Use* scored too low (0.600). For this reason, we removed the latter construct from our main analysis and will only report on the mean scores of the items. Note that it was the only construct measured using just two items—an approach to be reconsidered in future applications of the questionnaire.

4.3 Results

Our main analysis then focused on this question: Are the considered techniques, DCR Graphs vs. Declare, perceived differently by the groups? To select the appropriate technique, we established with the Shapiro-Wilk test that the respondent answers were normally distributed. We could, therefore, proceed with applying a one-way ANOVA test with Perceived Usefulness and Perceived Ease

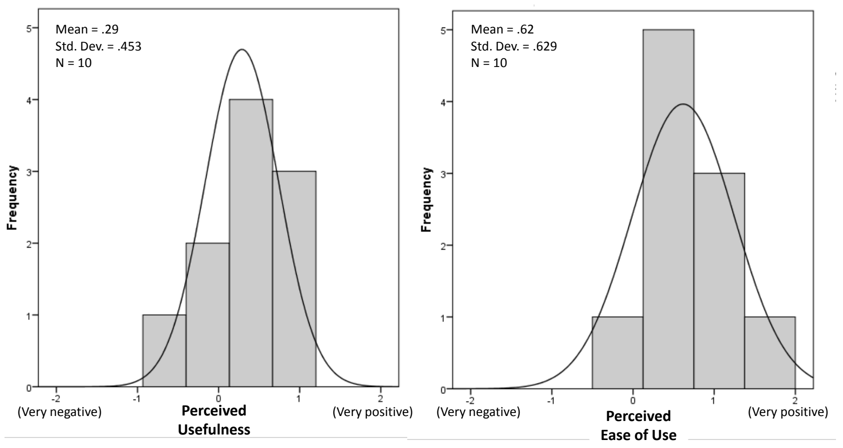


Fig. 4. Histograms for Perceived Usefulness and Perceived Ease of Use

of Use as dependent variables and the technique employed as factor. The test generated p-values of 0.116 and 0.939 for Perceived Usefulness and Perceived Ease of Use, respectively. By maintaining a confidence level of 95%, both of these values exceed the 0.05 threshold. In other words, any differences in perception between the used techniques are not statistically significant. Therefore, we must reject the idea that people perceive the techniques as different in either their usefulness or their ease of use.

This first important insight allows us to aggregate the responses received from both groups to determine a view on the usefulness and ease of use of declarative techniques on a more general level. Figure 4 shows the histograms for the two constructs under consideration, Perceived Usefulness and Perceived Ease of Use, aggregating the responses from all ten respondents. Also displayed is the fitted normal distribution for both constructs.

The histograms display the frequencies of the scores on a scale of -2.0 (very negative) to +2.0 (very positive). The 0 value indicates the neutral stance (not negative, not positive). What can be seen is that the averages of the distributions for both constructs are positive, hinting at a receptive mood toward declarative techniques in terms of both constructs. Note that the mean values for the two items under consideration for Intention to Use are 0.00 and 1.00. Second, Perceived Ease of Use seems to be more positively evaluated than Perceived Usefulness, with respective mean values of 0.62 and 0.29.

To determine whether the optically favorable outcomes are indeed statistically significant, we applied one sample t-tests. Like in our previous test, we used a confidence level of 95%, which means that we will only treat p-values below 0.05 as statistically significant. The outcomes of the t-tests are that the positive mean score for Perceived Ease of Use is significantly different from zero (p=0.013), but that this is—just—not the case for Perceived Usefulness (p=0.076). In other words, one can trust that the positive stance toward the ease of use is not a matter of chance. However, this cannot be ruled out for usefulness, despite its closeness to the cut-off value. Apparently, the involved respondents can easily use the method, despite the limited amount of training received. They were not similarly outspoken about the usefulness of a declarative technique, albeit certainly not negative either.

We finally checked whether the years of experience, the level of expertise, the type of role (consultant vs. non-consultant), or the modeling intensity in terms of models read or created had any relation to the outcomes. Interestingly, we could see that the most negative responses on Perceived Usefulness came from those respondents who assessed their own level of process modeling expertise as relatively low. While on average the three respondents with an intermediate expertise assessed the usefulness of the declarative techniques as negative (-0.208), the advanced modelers and the experts were positive (0.417 and 0.563, respectively). Tukey's HSD (Honestly Significant Difference) confirmed that the self-assessed level of expertise was a significant factor to explain differences in scores on Perceived Usefulness (p=0.042). In other words, the higher the level of expertise, the more merit a participant saw in the declarative techniques. The other factors had no noticeable effects on the scores.

5 Qualitative Evaluation

In this section, we present the qualitative evaluation of the workshop. In particular, we report on the results of the modeling assignment and of the discussion with the professionals.

5.1 Modeling Assignment

As reported in Sect. 3, we split the ten professionals into two groups of five. One group got an introduction to Declare and the other group to DCR Graphs. After this introduction of about 30 minutes, each group was asked to work on two small modeling assignments. One assignment was the document management system, which we used to illustrate Declare and DCR Graphs in Sect. 2. The second assignment was a hospital process of similar size and level of difficulty. The professionals worked in groups of two and three on the two assignments. Each assignment took less than 15 minutes, after which we presented and discussed our solution. All four groups came up with a correct solution for each of the two assignments.

The way we organized the assignment does not allow us to derive overly strong conclusions. Still, we gained two interesting insights. First, the result of the assignment shows that it is possible to teach declarative modeling to practitioners. Although it was difficult for the professionals to get used to the declarative way of modeling and to the graphical notations of the techniques, they came up with correct models in reasonable time. Second, we were told that the graphical notation of Declare and DCR Graphs are too academic and for practitioners neither convincing nor intuitive. Moreover, also the informal description, which we provided for each introduced constraint, did not help them to easily identify the constraint they needed. These comments hold for both techniques, DCR Graphs and Declare. This comes, indeed, not as a surprise as both formalisms have an academic background. However, we expected DCR Graphs to be more comprehensible and easier to use than Declare because DCR Graphs only consist of five relations, whereas Declare requires to learn a larger set of constraints.

5.2 Opportunities for a Declarative Approach

In the subsequent discussion with the professionals, we tried to figure out whether they see opportunities for a declarative modeling approach. Clearly, such a question is difficult to answer given the the short tutorials and only taking two assignments. The participants did indicate that there are probably processes that can be modeled most naturally using the imperative approach, while others would fit better with the declarative approach. For example, a clearly well-structured process of registering a newborn at a townhall can be modeled most naturally in an imperative way whereas the document management system of Sect. 2 is an example of a process that can be modeled most natural in a declarative way. In addition, in almost all processes the professionals came across, there were always at least parts or subprocesses where an imperative approach seems most natural. So, the conclusion to this question is that a purely declarative approach seems less attractive than a *hybrid approach*, which combines imperative and declarative modeling aspects.

5.3 Requirements Concerning a Declarative/Hybrid Approach

In the previous section, we showed that practitioners see opportunities for a hybrid approach. Next, we report on the practitioners' requirements concerning the specification, the constraints, the process, and tool support.

The consensus was that the efficient design of a declarative model (or of the declarative part of a hybrid model) will require a declarative specification. The reason is that it can be nontrivial to derive constraints from an imperative specification. We received one comment that it might be difficult to get a declarative specification at all, but we are not that pessimistic. Based on our experience, it depends on how one formulates questions to domain experts; that is, asking about the relationship between two activities (i.e., declarative) rather than which steps can be performed in a certain state (i.e., imperative) will allow one to come up with a declarative specification.

Other requirements concern the constraints. The involved professionals brought forward that too many constraints may negatively influence the quality of a declarative model. For example, many constraints affecting few activities may result in an unreadable model. Furthermore, they assumed the completeness of the constraints to be crucial, although that is similar to the completeness of the branching conditions in an imperative model.

Looking at the process model or the specification to identify "candidate" parts that may benefit from a declarative modeling approach, the professionals suggested to identify parts that have many dependencies (e.g., spaghetti-like parts). Although such parts seem good candidates, it is unclear whether a declarative way of modeling results in a better model. Another suggestion was to identify those parts where much modeling freedom is; for example, a set of concurrent activities (that preferably occur more than once) with only few dependencies may result in a simpler declarative model than their imperative counterpart.

Finally, also proper tool support is a hard requirement. Here, in particular, the professionals saw deficits in the usability of the academic techniques when used in a pen-and-paper fashion. We shall discuss tool support in Sect. 5.6 in more detail.

5.4 Use Cases for a Declarative/Hybrid Approach

In this section, we list use cases for the declarative approach as identified during the discussion.

Process evolution [21] was mentioned as the main use case by the involved professionals—that is, manage processes along the various changes it encounters. Having a set of constraints rather than a graph-based model seems to be beneficial to visualize changes over time, on the one hand and to actually change a process model, on the other hand. This is, in fact, one of the claimed advantages of declarative modeling [3,15]. The "outside-in" approach of declarative models allows for a higher level of abstraction than an imperative process model. Therefore, it is often simpler to add or remove constraints than changing a BPMN model, for instance.

The discussion also suggested that the use case for declarative models is tied to model purpose. Process models serve different purposes—for example, as a medium to communicate with stakeholders (i.e., communication model) or to execute a process (i.e., executable model). Especially with respect to the communication aspect, the professionals saw good opportunities for using declarative techniques. Communication models are rather imprecise (e.g., exceptions may be left out), and business analysts do not tend to stick to model conventions. Instead, they may prefer to use short hands to illustrate behavior in a simplified way, for instance. Here, a hybrid approach looks promising as the business analyst is provided with a lot of different ways to present the model. Again, this follows from the higher level of abstraction of declarative models. In contrast, there was no common agreement on a declarative approach being useful for specifying executable models. As an executable model contains all behavior, a hybrid approach will only be beneficial if it allows for designing more readable or simpler models.

Another interesting aspect mentioned was that a hybrid approach may result in fewer errors in the model than using a purely imperative approach. This may lead to shorter development cycles. We think that this is also a consequence of the higher level of abstraction in declarative modeling. A modeler has to identify the constraints rather than encode it in terms of control flow. However, no experience report or empirical results exist that confirm this assumption.

5.5 Limitations of a Declarative/Hybrid Approach

In this section, we report on limitations of a declarative/hybrid approach concerning the specification, the modeling paradigm, and the usability.

The main concern regarding the specification is that currently all specifications are imperative (e.g., "we first do this, then that"), and it seems to be very difficult to produce a declarative model for such a specification. As discussed in Sect. 5.3, we think that it is possible to receive declarative specifications.

There has been a paradigm shift in system development from monolithic systems to component-based systems that are distributed within and across organizational boundaries. One prominent computing paradigm that implements this trend is service-oriented computing (SOC) [19]. We received concerns that in this setting declarative modeling techniques may be less applicable compared to imperative techniques. The reason for this concern lies in the fact that certain constraints affect activities of an individual component, whereas other constraints affect activities of different components. Declarative techniques have, however, been successfully applied in the service-oriented setting [14,15], and it has been studied how a declarative cross-organizational workflow containing global constraints can be projected to its individual localized components [11], so we are convinced that this concern is unsubstantiated.

Another limitation concerns the usability of existing techniques and tools. Current tool support is mainly academic by nature and seems, therefore, not overly concerned with usability issues. Moreover, the declarative paradigm also requires a different way of thinking, making it perhaps difficult for practitioners to understand declarative models. Here, more research is required to make declarative techniques more comprehensible.

5.6 Requirements Concerning Tool Support

In this section, we report on feedback we received concerning Declare and DCR Graphs and general requirements concerning tool support.

Several requirements on tools that were discussed deal with the specification and visualization of *constraints*. As mentioned earlier in Sect. 5.1, the professionals mentioned that working with constraints was relatively difficult for them. The graphical notations used in Declare and DCR Graphs were not always that intuitive. Moreover, specifying constraints in plain English is not always helpful either, because it is often nontrivial to identify the differences between two constraints. Therefore, the professionals proposed that constraints should be automatically derived from an informal textual specification. This problem has indeed been investigated in the field of computer-aided verification, for instance. Different approaches have been proposed, for example [7,5], but none of them could solve the problem entirely.

A given set of constraints makes it necessary to check for conflicting constraints. This is a feature which has been implemented in most declarative modeling tools [24], but has also been investigated in the context of compliance rules [4], for instance. Another important feature is to generate a model from a given set of constraints and to identify missing features. This problems is related to scenario-based programming [10]. In case an implementation and recorded event logs exist, process mining techniques to automatically derive missing constraints from the logs are required. First attempts at dealing with this topic exist, see [13].

Besides modeling support, tools should preferably also provide operational support. For example, event logs may be exploited to provide at runtime the best possible next step. Such features are implemented in recommender systems.

Finally, usability plays an important role. Specification of constraints, their graphical representation and the complete interplay between the tool and an end user must be on an abstraction level that is adequate to the task at hand.

6 Research Agenda

In this section, we pick up the results from the discussion with the professionals as presented in the previous section. We propose a research agenda for the development of a hybrid modeling approach that combines imperative and declarative techniques. The aim is thereby to point out necessary steps for developing and actually using a hybrid technique rather than a complete research agenda.

Model guidelines In order to apply the hybrid approach, a modeler has to know when to model in an imperative and when in a declarative way. In other words, we need to identify modeling guidelines to guide modelers through the modeling process. This requires rules for identifying imperative and declarative "candidate" parts on the level of an existing (imperative) process model (e.g., for process redesign), on the level of event logs (e.g., for process discovery), and on the level of (informal) specifications (e.g., for designing a new model).

Identify the hybrid technique Modeling in a hybrid way requires a well-suited modeling language. It needs to be investigated whether we can combine existing imperative and declarative languages or whether a new language has to be designed. For instance, we can integrate a declarative part as a subprocess into an imperative model (e.g., as a hierarchical transition in a Petri net or subprocess task in BPMN) or we can allow declarative and imperative constructs to coexist within a single subprocess. The modeling language must in any case support hierarchy. In the latest version of CPN Tools [23], Westergaard integrated DCR Graphs and Declare into Colored Petri nets. It turned out that defining the semantics of such models is nontrivial.

Beside that, it needs to be settled which constraints are relevant for practise and, thus, what the expressiveness of the declarative part of the language is. Empirical research has shown [8,9] that certain declarative constructs may be more difficult to understand. Thus, we think the language should not contain too many declarative constructs, but this needs further empirical investigation.

Also, the graphical representation of hybrid models must be investigated. Different graphical notations exist, for example, compare DCR Graphs and Declare. Insights from [16] may aid the design of a hybrid notation.

Analysis of hybrid models The novel modeling approach needs analysis techniques including the verification of models, performance analysis, and property-preserving abstraction and refinement techniques. Also, process mining techniques [1] are needed—for example, checking the conformance of an event log and a hybrid model and discovering a hybrid model from a given event log.

Tool support To show the applicability of the hybrid modeling technique, tool support is a *sine qua no*. As reported in Sect. 5.6, research has to be performed to simplify the use of declarative techniques, for example, finding a way to derive constraints from informal specifications that can be used by business analysts without requiring knowledge about temporal logics.

7 Conclusion

We reported on a workshop on declarative modeling given to professionals from industry. The goal of this workshop was to gain insight into what practitioners think about declarative modeling and what opportunities they see to use this technique. Our quantitative evaluation showed that they were mostly positive and open to this modeling paradigm. In particular, the techniques were rather easy to learn. The qualitative evaluation showed that the practitioners did single out the use of declarative techniques in the context of a hybrid approach, which combines imperative and declarative modeling. Although our study is only based on a small group of practitioners, we are convinced that practise can benefit from such a hybrid modeling approach. To arrive at such an approach, we proposed a research agenda for the development of a hybrid approach.

In our ongoing research, we plan to work on the development of modeling guidelines. We will investigate techniques to identify "candidate" parts of a model for which a declarative way of modeling seems most natural. Also, we plan to study event logs and process models and try to use the results to identify constraints that frequently occur. In a second branch of research, we will investigate what a hybrid technique may look like, thereby using Declare, DCR Graphs and CPN Tools as starting points for our studies.

References

1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Aalst, W.M.P.v.d.: Business process management: A comprehensive survey. ISRN Software Engineering 2013 (2013)

3. Aalst, W.M.P.v.d., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D 23(2), 99–113 (2009)
4. Awad, A., Weidlich, M., Weske, M.: Consistency checking of compliance rules. In: BIS 2010. LNBIP, vol. 47, pp. 106–118 (2010)
5. Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User guidance for creating precise and accessible property specifications. In: SIGSOFT FSE. pp. 208–218. ACM (2006)
6. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. 13(3), 319–340 (1989)
7. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE 1999. pp. 411–420. ACM (1999)
8. Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: BMMDS. LNBIP, vol. 29, pp. 353–366. Springer (2009)
9. Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of maintainability. In: BPM Workshops. LNBIP, vol. 43, pp. 477–488. Springer (2010)
10. Harel, D.: Come, let's play - scenario-based programming using LSCs and the play-engine. Springer (2003)
11. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: SEFM 2011. pp. 237–252. Springer (2011)
12. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES 2010. EPTCS, vol. 69, pp. 59–73 (2010)
13. Maggi, F.M., Bose, R.P.J.C., Aalst, W.M.P.v.d.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer (2012)
14. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, LNBIP, vol. 56. Springer (2010)
15. Montali, M., Pesic, M., Aalst, W.M.P.v.d., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. TWEB 4(1) (2010)
16. Moody, D.: The physics of notations: toward a scientific basis for constructing visual notations in software engineering. Software Engineering, IEEE Transactions on 35(6), 756–779 (2009)
17. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. In: ECIS 2003. pp. 1327–1336 (2003)
18. Nunnally, J.: C.(1978). Psychometric theory. New York: McGraw-Hill (1978)
19. Papazoglou, M.P.: Web Services - Principles and Technology. Prentice Hall (2008)
20. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 383–394. Springer (2012)
21. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems. Springer (2012)
22. Weber, B., Reijers, H.A., Zugal, S., Wild, W.: The declarative approach to business process execution: An empirical test. In: CAiSE 2009. LNCS, vol. 5565, pp. 470–485. Springer (2009)
23. Westergaard, M.: CPN Tools 4: Multi-formalism and extensibility. In: Petri Nets 2013. LNCS, Springer (2013), accepted for publication
24. Westergaard, M., Maggi, F.M.: Declare: A tool suite for declarative workflow modeling and enactment. In: BPM (Demos) 2011. CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)

Mixing Paradigms for More Comprehensible Models

Michael Westergaard[1,2*] and Tijs Slaats[3,4]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
[2] National Research University Higher School of Economics,
Moscow, 101000, Russia
[3] IT University of Copenhagen
Rued Langgaardsvej 7, 2300 Copenhagen, Denmark
[4] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark
m.westergaard@tue.nl, tslaats@itu.dk

Abstract. Petri nets efficiently model both data- and control-flow. Control-flow is either modeled explicitly as flow of a specific kind of data, or implicit based on the data-flow. Explicit modeling of control-flow is useful for well-known and highly structured processes, but may make modeling of abstract features of models, or processes which are highly dynamic, overly complex. Declarative modeling, such as is supported by Declare and DCR graphs, focus on control-flow, but does not specify it explicitly; instead specifications come in the form of constraints on the order or appearance of tasks. In this paper we propose a combination of the two, using colored Petri nets instead of plain Petri nets to provide full data support. The combined approach makes it possible to add a focus on data to declarative languages, and to remove focus from the explicit control-flow from Petri nets for dynamic or abstract processes. In addition to enriching both procedural processes in the form of Petri nets and declarative processes, we also support a flow from modeling only abstract data- and control-flow of a model towards a more explicit control-flow model if so desired. We define our combined approach, and provide considerations necessary for enactment. Our approach has been implemented in CPN Tools 4.

1 Introduction

Petri nets provide a powerful formalism for specifying many real-life systems, including business processes. Petri nets excel by having a duality between data and events, yielding a very powerful tool for specifying how data flows though a system. Control-flow of a Petri net model is often modeled explicitly as flow of a specific kind of data, similar to a program counter in traditional programming. Alternatively, the control-flow is not modeled at all, and just manifests as a consequence of the data-flow. As such, we call a Petri net model a procedural model

* Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.

as the control-flow when disregarding data is close to procedural programming languages: modelers specify *how* to solve a problem. An example where such a language is useful, is classical filling of forms, such as a patient registration process at a hospital.

Declarative specification of processes is an emerging trend for specifying especially business processes, but it has not seen massive use in practice. Declarative models often focus primarily on flow of control, but instead of explicitly modeling control-flow as a program counter, constraints between the different events are described. Declarative languages resemble declarative programming languages: modelers specify what the *intention* of the control-flow is, but not how to achieve that. An example where such languages are useful, is a patient treatment process at a hospital; here, many tests need to be run and many treatments are possible. There is no strict order of tests and treatments, but some treatments are incompatible with each other, and some treatments need follow-up treatments.

Declarative processes are typically better at describing highly dynamic environments, where actions can take place in many different orders, or early in the design, where the exact order of events is unknown. On the other hand, Petri nets are far better at modeling data-flow, and the strict control-flow model makes it easier to model processes with a strict and well-understood control-flow, which also makes it much easier to extract experiences from the model to an eventual implementation [10]. In this paper, we propose to merge two declarative approaches, Declare [15,19] and DCR graphs [7,13], with a high-level Petri nets formalism, colored Petri nets [9], to obtain a formalism that offers the best of both worlds. We aim to do so in a manner that makes it possible to use all three formalisms completely independently of each other or to mix all three formalisms in a single model. This makes it also possible to initially construct a purely declarative model, optionally with data, and during refinement make it more procedural as applicable. If we consider a hospital, this allows us to make a single model comprising both patient registration, diagnosis, and treatment. The reason for using both DCR graphs and Declare for the declarative parts is that the languages have different focus areas: Declare provides higher level primitives, often resulting in more comprehensible models, but DCR graphs do not suffer from the computational overhead of detecting conflicts necessary to ensure correct execution of Declare models.

We introduce our combined approach in Sect. 2, including pointers on how to allow analysis of combined models and our implementation in CPN Tools 4 [17]. In Sect. 3, we sum up our conclusions and compare with related work.

2 Combined Models

In this section we informally introduce our hybrid model and its semantics, and provide analysis considerations important for implementation. Actual implementation details are deferred to the next section.

The idea behind the hybrid approach is to identify transitions of CP-nets with tasks of Declare models and events of DCR graphs and then allow places and arcs (with annotations) from CP-nets, constraints from Declare models, and the relations from DCR graphs to be added to the model to constrain the possible executions.

The reason for including these three languages is that CP-nets is a widely used procedural formalism with a strong theoretical background. It provides great support for data flow, both theoretically and practically in the form of tool support. We also prefer to use both Declare and DCR graphs for specifying the declarative parts of the model. We choose these two languages because they are on the surface very similar, yet they have different focus areas.

Declare offers a large set of contraints which have been identified as commonly used in business processes, making it well-suited to the BPM domain. DCR Graphs on the other hand aim to provide a formal language for describing processes in general, containing only 4 basic constrains while still being formally more expressive than LTL.

By providing both languages, we can use pre-existing tools and techniques to analyze our combined models, automatically switching from one kind of analysis to the other as needed.

An execution is considered accepting if it is accepting for all three underlying models. In other words, the execution should be accepting for the CP-net that one gets when removing all Declare constraints and DCR Graph relations, it should be accepting for the Declare model one gets by removing all places, arcs and DCR Graph relations and it should also be accepting for the DCR Graph model that one gets by removing all places, arcs and Declare constraints. Formally, we can define the semantics of all three languages in terms of transition systems, and the semantics of the combined language is just the synchronization of the three transition systems we get from the individual semantics by projecting the combined model onto each of the three languages.

2.1 Analysis

We would like to provide a step-wise semantics for combined models. This is necessary for efficient simulation. For CP-nets isolated, this is easy because every state is accepting, so if a binding element sequence is enabled, the execution will inevitably end in an accepting state. For Declare models and DCR graphs, this is possible using a preprocessing step: we simply compute the prefix automaton, which is possible as they both have a semantics yielding finite automata, and only allow a transition if it leads to an accepting state in the prefix automaton. For the combined models, however, this is not in general decidable. While we can construct the transition system product of the 3 automata on the fly, we cannot employ any of the techniques to ensure we can end up in an accepting state: as not all states of Declare models and DCR graphs are accepting, not all states of the product are necessarily accepting, so we cannot just execute any enabled binding element sequence and be sure to end in an accepting state. As the transition system we get from a CP-net is not necessarily finite, neither is the product, so we cannot compute the prefix automaton. If either the CP-net model is bounded (yielding a finite state space), or the Declare model automaton and the DCR graph automaton only have accepting states, we can use the fact that these properties are preserved by transition system product and use the appropriate technique. Otherwise, we must settle for weaker guarantees.

When talking about runtime verification of Declare models [12], each constraint can be in not just the two states *satisfied* and *violated*, but also in two weaker states, where a constraint is only temporarily satisfied or violated, but future execution may violate or satisfy it. Only when the execution is terminated, is it possible to collapse possible satisfied/violated constraints into their (permanently) satisfied/violated counterparts.

For DCR Graphs we do not keep track of the state of individual constraints, instead we have a current marking and on execution check that the executed event is enabled and calculate the new marking. If a marking contains no pending included responses, the DCR Graph is in an accepting state and the process can terminate. Feedback to the user consists of showing which events have occured before, are enabled and need to occcur.

Simple simulation. As demonstrated in [18], even if Declare is decidable, constructing the automaton for the full system can be very time and memory consuming – it is exponential in the number of constraints. To avoid this overhead, we can instead create an automaton for each individual constraint. If we do so, we can avoid ever violating individual constraints, while retaining fast simulation (we can update the model in the initial state in constant time). CPN Tools offers a mixed mode, where simulation and editing are interleaved. This is useful for testing and debugging, but requires that the simulation can resume very quickly, so performing an operation that is exponential in the size of the model may be undesirable (at least for large models). By constructing the individual automata, we can avoid ever (permanently) violating constraints, and for some constraints, e.g., init, this is sufficient. For other constraints, this provides a best-effort but fast simulation mode (we can update the model in constant time in the initial state). We call this the *simple simulation* approach. The simple simulation approach for DCR Graphs comes down to doing basic runtime verification as described previously: checking that an event is enabled in the current marking and calculating a new marking can be done in constant time.

Smart simulation. As shown in [11] some Declare constraints can be in a conflicted state: they are not violated, but also cannot all be (possibly) satisfied at the same time. We can only catch this if we construct the automaton for the full Declare model. By making the product explicit, we can compute the prefix automaton. Unfortunately, the product of the prefix automata is not sufficient. As demonstrated in [18], this can still be fairly fast for moderately-sized models (in the order of seconds for models with 30-50 constraints). We call this *smart simulation*: we avoid executing any transition that would lead to a conflicted state. We can also do *smart simulation* of the DCR Graph constraints by building the finite automaton that corresponds to the graph and only allowing the execution of events that can lead to an accepting state (i.e., the DCR Graph does not contain any deadlocks). The efficiency of this approach has not been investigated structurally yet, but for the models that we have considered to date the state space tends to be modest. We can compute the product of the automata from the underlying Declare model and the underlying DCR Graph model to ensure that the two kinds of declarative constraints cannot conflict with other as well.

Data-aware simulation. When combining declarative models with CP-nets, we get an additional type of conflicts: a declarative constraint might require some task to be executed, while the the CP-net model blocks its execution (f.e. because of a missing token). Smart simulation cannot catch this on its own as it only looks at the declarative (and computable) parts of the model. To handle such situations we need constraints that yield automata which only have accepting states, which severely limits the usability, or that the state-space is finite. Thus, *data-aware simulation* is as hard as state space analysis.

For simple examples with small domains, we can just generate the state-space and perform the synchronization, typically in minutes or hours. If the state-space is larger but still finite, we can perform many simulations using smart simulation and discard any not ending in an accepting state, similar to how simulation is used for bug-finding until a final verification often is used. After computing the synchronization, it can be stored efficiently (often only few states are conflicted). If we deal with large domains, it is sufficient if we can generate an equivalence-reduced state-space. This is for example the case if all types are integers or reals, and we only compare all tokens with integers, similarly to region or zone reduction for timed automata.

2.2 Implementation

We have implemented our combined models in CPN Tools 4 [5,17]. CPN Tools 4 adds support for *simulator extensions* [17], a mechanism which makes it possible to extend CPN Tools using Java code. Each extension can add operations to CPN Tools and also modify existing operations. The integration comprises 4 parts: GUI extension, syntax check extension, enabling restriction, and analysis. In Fig. 1, we see how combined models look and are constructed in CPN Tools.

3 Conclusion

In this paper we have introduced a new approach to modelling workflows combining the procedural formalism colored Petri nets, and the two declarative formalisms, Declare and DCR graphs. The combined formalism can be seen as adding declarative control-flow to CP-nets or as adding data-flow to declarative formalisms. Declarative approaches are typically better for abstract descriptions or highly dynamic processes, where procedural approaches are better for well-known and structured processes. Combining the two allows us the best of both

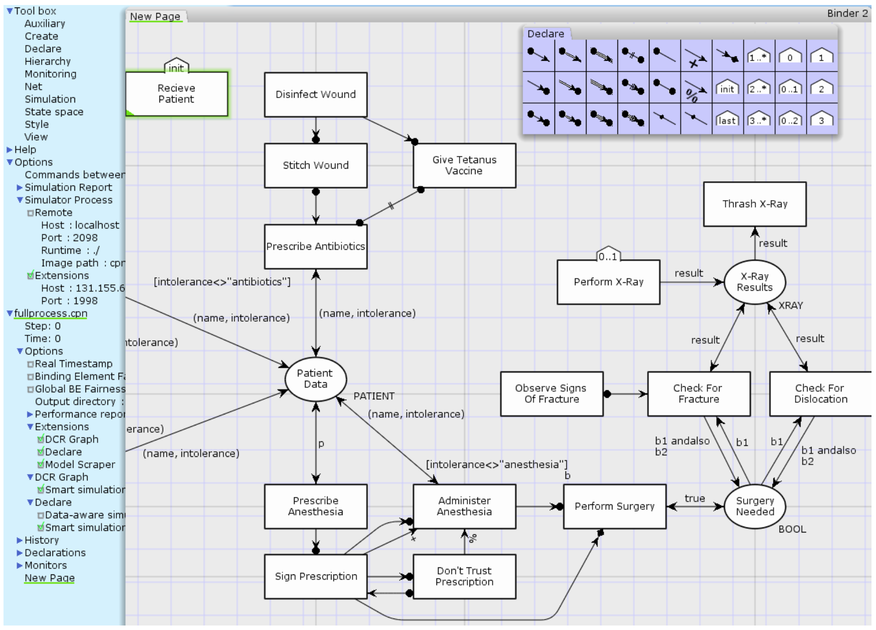


Fig. 1: Declare and DCR graphs in CPN Tools 4.

worlds and allows declarative processes to also deal with data. We have considered what is needed to provide simulation that avoids future conflicts in efficient ways, and introduce three modes of simulation: simple, smart, and data-aware, where the simple mode only avoids individual conflicts, smart avoids conflicts not related to data, and data-aware makes sure that even in the presence of data, all executions can terminate successfully. We have briefly introduced our implementation in CPN Tools.

3.1 Related Work

The *Guard-Stage-Milestone* (GSM) model [8] by Hull et al, which originated from the work on artifact-centric business processes [3], takes an approach to modelling business processes where a process consists of a number of (possibly nested) stages, which in turn contain a number of tasks. A stage also has guards and milestones; it is activated by satisfying its guards and through performing the tasks in the stage its milestones can become enabled, which can then in turn satisfy the guards of other stages. We see the GSM model as a hybrid model combining procedural and declarative structures in a single language, whereas our approach is based on combining existing procedural and declarative languages. In [14] the authors introduce a declarative version of the Computer-Interpretable Guidelines (CIG) language for modelling clinical guidelines, they conclude that because both the procedural and declarative languages have their disadvantages it would be best to combine them into a single model, but leave this for future work. In [16] the authors have examined the understanding of procedural and declarative process models by users. In their conclusions they note that while it appears that procedural models are more comprehensible, it remains uncertain to what extent this is caused by participants being skewed towards procedural models because of their general acceptance and availability. They do not consider a hybrid approach using both procedural and declarative concepts. In [6] Fahland bridges the gap between declarative and procedural workflow approaches by proposing a general compositional mechanism for translating declarative workflow models to procedural workflow models. He exemplifies the general approach by giving a translation from Declare to Petri nets. The main difference to our approach is that while in [6] a declarative model is translated to a procedural version to facilitate using existing modeling, analysis and management techniques, we aim to combine the declarative and procedural approaches and provide tools and techniques for the hybrid approach.

3.2 Future Work

Here we have assumed that a user creates and refines a model. Another approach is to have the tool do that. For example, a precedence(A, B) constraint is trivially modeled using a single place of type boolean, initially marked by false. Then A changes the value indiscriminately to true and B checks that the value on the place is true. not co-existence(A, B) can be implemented using a place with three possible values: {A,B,UNDECIDED}. init(A) is less elegant, but can be implemented using, e.g., inhibitor arcs. This will not catch conflicts, but we can do that (and translate all constraints in a uniform way) by constructing the finite automaton either just equate the states of the automaton with new places or use a (not data-aware) process mining algorithm [1] or the theory of regions [4] to construct a Petri net for the control flow. Future work includes investigating the best way to make such a translation (semi-)automatically.

The current approach works as long as we describe a single run of a single process. That is, our example in the figures handles one treatment of one patient. It would be interesting to investigate multiple instances, which would essentially be a folding of the current model. We could also have multiple processes communicate, e.g., like Proclets [2], resulting in a more artifact-driven result. Using a translation from declarative constraints would essentially result in process-partitioned CP-nets in the sense of [10], which means it would be possible to automatically derive code from the resulting model. It would also be interesting to look into making a proper notion of a process in CP-net models, where instances cannot just terminate in any state. This would also include adding a notion of instances to the declarative languages.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow Modeling using Proclets. In: Proc. of CoopIS'00. pp. 198–209. LNCS, Springer (2000)
3. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Proc. of BPM'07. pp. 288–304 (2007)
4. Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Proc. of BPM. LNCS, Springer (2008)
5. CPN Tools webpage. Online: `cpntools.org`
6. Fahland, D.: Towards analyzing declarative workflows. In: Autonomous and Adaptive Web Services. Dagstuhl Seminar Proceedings, vol. 07061, p. 6. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI) (2007)
7. Hildebrandt, T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Post-proc.of PLACES 2010 (2010)
8. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of WS-FM'10. pp. 1–24. Springer-Verlag, Berlin, Heidelberg (2011)
9. Jensen, K., Kristensen, L.: Coloured Petri Nets – Modelling and Validation of Concurrent Systems. Springer (2009)
10. Kristensen, L., Westergaard, M.: Automatic Structure-Based Code Generation from Coloured Petri Nets: A Proof of Concept. In: Proc. of FMICS. pp. 215–230. LNCS, Springer (2010)
11. Maggi, F., Montali, M., Westergaard, M., Montali, M., van der Aalst, W.: Runtime Verification of LTL-Based Declarative Process Models. In: Proc. of RV. LNCS, vol. 7186, pp. 131–146. Springer (2011)
12. Maggi, F., Montali, M., Westergaard, M., van der Aalst, W.: Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In: Proc. of BPM. LNCS, vol. 6896, pp. 132–147. Springer (2011)
13. Mukkamala, R.R.: A Formal Model For Declarative Workflows - Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (March 2012)
14. Mulyar, N., Pesic, M., van der Aalst, W.M.P., Peleg, M.: Declarative and procedural approaches for modelling clinical guidelines: Addressing flexibility issues. In: Proc. of BPM'07. pp. 335–346 (2007)
15. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
16. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: Proc. of ER-BPM '11. pp. 383–394 (2011)
17. Westergaard, M.: CPN Tools 4: Multi-formalism and Extensibility. Submitted to Petri Nets 2013
18. Westergaard, M.: Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In: Proc. of BPM. LNCS, vol. 6896, pp. 83–98. Springer (2011)
19. Westergaard, M., Maggi, F.: Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In: Business Process Management Demonstration Track (BPMDemos 2011). CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)

The Automated Discovery of Hybrid Processes

Fabrizio Maria Maggi[1], Tijs Slaats[2,3], and Hajo A. Reijers[4,5]

[1] University of Tartu, Estonia
[2] IT University of Copenhagen, Denmark
[3] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark
[4] Eindhoven University of Technology, The Netherlands
[5] Perceptive Software, The Netherlands
f.m.maggi@ut.ee, tslaats@itu.dk, h.a.reijers@tue.nl

Abstract. The declarative-procedural dichotomy is highly relevant when choosing the most suitable process modeling language to represent a discovered process. Less-structured processes with a high level of variability can be described in a more compact way using a declarative language. By contrast, procedural process modeling languages seem more suitable to describe structured and stable processes. However, in various cases, a process may incorporate parts that are better captured in a declarative fashion, while other parts are more suitable to be described procedurally. In this paper, we present a technique for discovering from an event log a so-called *hybrid* process model. A hybrid process model is hierarchical, where each of its sub-processes may be specified in a declarative or procedural fashion. We have implemented the proposed approach as a plug-in of the ProM platform. To evaluate the approach, we used our plug-in to mine a real-life log from a financial context.

1 Introduction

Process models are an important aid to capture how business operations are organized. One direction to simplify the tasks of creating, maintaining, and reading such models involves the use of declarative techniques for process modeling. In contrast to the procedural approach, which is dominant for modeling business processes, a declarative approach leaves implicit in what exact sequences activities must be carried out. Instead, the emphasis is on the constraints that must be respected in carrying out the process – any behavior that respects these goes. In contexts where activities can be executed in highly different combinations, a declarative approach arguably produces simpler representations of the involved process logic. Examples of concrete declarative modeling techniques are Declare, DCR Graphs [3], and SCIFF.

In [10], we reported that a *hybrid* process modeling technique was considered by practitioners as more attractive than a completely declarative or procedural one. Hybrid, in this context, refers to the potential use of both procedural and declarative model elements in the same model. The rationale is that the two types of modeling paradigms allow for a natural fit with different types of process behavior. In places where the process is highly flexible, a declarative modeling approach leads to a compact and simple description of such a "pocket of flexibility" [11]. Instead of describing all the different types of feasible behavior, the focus is then on ruling out what is not allowed (if anything). By contrast, for parts of the process that are highly structured, a procedural description may be the way to go: It is then simpler to describe what is allowed than what is to be ruled out. For processes that both incorporate structured and unstructured pockets, a hybrid model delivers a compact and simple description.

This paper should be seen as a direct follow-up to our earlier work. Specifically, we developed a technique *to automatically generate a hybrid model from an event log*. This is a novel contribution, since existing techniques can only generate a process model that is either procedural or declarative. By contrast, our technique flexibly alternates between employing a procedural or declarative mining approach in accordance with the nature of the traces it processes. By doing so we are able to avoid the "spaghetti"-like process models that are commonly generated by traditional process mining techniques.

Against this background, the paper is structured as follows. In Section 2, we will outline the notion of a hybrid model and pinpoint its semantics. Section 3 describes our core contribution, the discovery approach. We will will evaluate this approach in Section 4 on a real-life log. After a discussion of related work, we conclude this paper with a reflection on the presented work and future steps in Section 6.

2 Semantics of a Hybrid Model

Our interest in this paper is with hybrid models where the procedural and declarative parts are contained in separate sub-processes. In this sense, there is a resemblance with the *pockets of flexibility* concept [11]. A hybrid process consists of a procedural or declarative top-level process, which may contain a number of atomic activities as well as sub-processes. Each sub-process can be either procedural or declarative and may contain sub-processes of its own. Our approach is applicable to any combination of procedural and declarative languages, but in this paper we will apply Petri nets for our procedural models and Declare [9] for our declarative models. Sub-processes are considered atomic, meaning that once a sub-processes is started the control is passed from the parent process to that sub-process. No other activities can be executed until the sub-process has completed. A sub-process can only complete while it is *accepting*. When exactly it is accepting depends on the language used. In the case of a Petri net this means reaching a final marking, while for a Declare model it means having no violated constraints. For the language of a hybrid model we consider the start and completion of sub-processes as silent transitions, which means that there will be no start- and complete-events for the sub-processes in the log. This underlines the fact that the sub-processes are really just a tool for improving the understanding of the process and not a part of the actual enactment of the process.

3 Discovering Hybrid Process Models

Fig. 1 gives an overview of our approach. In the following paragraphs we describe our approach step by step.

Distinguishing Structured and Unstructured Events. We start by separating the events of the log into two distinct sets: one containing those events that occur in a structured

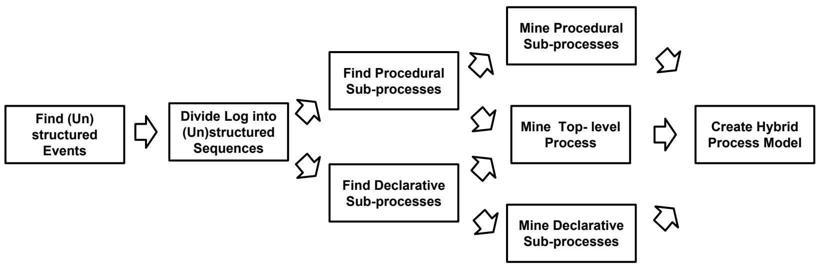

Find (Un) structured Events
Divide Log into (Un)structured Sequences
Find Procedural Sub-processes
Find Declarative Sub-processes
Mine Procedural Sub-processes
Mine Top- level Process
Mine Declarative Sub-processes
Create Hybrid Process Model

Fig. 1. Overview of our approach

context and one containing those events that occur in an unstructured context. To distinguish structured from unstructured events we use a novel technique, which we refer to as *context analysis*. Our first step is to determine for each event the number of unique predecessors and successors to that event. We then consider an event with a large number of both predecessors and successors to be unstructured (according to a user-defined threshold, in our experimentation we used 4), while an event with a small number of predecessors or a small number of successors is considered to be structured. The reasoning behind these cases is as follows: if an event has a high number of predecessors and a high number of successors, then there are few rules constraining when exactly the event can occur. We then consider it likely to fit well into a declarative model. Similarly, if an event has only a small number of predecessors and a small number of successors, then it is probably more easily modeled procedurally, for example, as a sequence or a choice from a low number of options. In the case that an event has a small number of predecessors, but a large number of successors, we consider it likely that the event is either the last element in a structured sequence, which is followed by an unstructured sequence, or that the event is followed by a choice from a large number of options. In both cases it makes sense to consider this as a structured event and model it procedurally. Similarly, in the case that an event has a small number of successors, but a large number of predecessors we consider it likely that the event is either the first element in an structured sequence, which was preceded by an unstructured sequence, or that the event joins a choice from a large number of options. In both cases it seems fitting to consider this as a structured event and model it procedurally.

Dividing the Log into Structured and Unstructured Sequences. The context analysis gives us two sets: one that contains structured events and one that contains unstructured events. In the following step, we use these events to identify structured and unstructured sequences by parsing the log and starting a new sequence whenever an event does not belong to the same set as its preceding event. After this step, our approach splits into two branches, one handling the structured sub-logs and the other handling the unstructured sub-logs.

Finding and Mining procedural Sub-processes. By grouping together each structured event with all the direct successors and all the direct predecessors, we obtain a set of disjoint clusters of the structured sequences. We then mine procedural sub-processes for each of these clusters. Finally, we abstract the main log by replacing each sequence with the identifier of the sub-process that it belongs to. It should be noted that the clusters of procedural sequences that are discovered could be further split up using existing clustering techniques. This did not seem necessary on basis of the examples we used in our experimentation with the technique.

Finding and Mining Declarative Sub-processes. For finding declarative sub-processes, we first use an indirect association rule mining algorithm on the set of unstructured sequences to find declarative patterns. Recall that an indirect association rule can be used to find events that rarely occur together, yet there are other "mediator events" with which they appear relatively frequently. We use this type of algorithms since it gives us the opportunity to not only discover Declare constraints that express positive relations, but also constraints like, for example, not coexistence constraints, which are more likely to be satisfied when the events involved do not occur together in the same trace. In a second stage, we use a mining algorithm for standard association rules on the remaining sequences. These rules reflect relationships that exist between events that often co-occur in common transactions. For this reason, these rules allow us to group together events that are very likely connected with each other through positive relations in Declare. The patterns are abstracted in the main log, and any remaining event that is at this point not identified as belonging to a declarative pattern is left as an atomic event.

Mining the Top-level Process. When we are done finding (but not necessarily mining) procedural and declarative sub-processes and have all sub-processes abstracted in the main log, we can then either choose to apply the approach iteratively on the abstracted log, starting from the first step where we distinguish structured and unstructured activities based on their context, or we can choose to finalize the approach by mining the main log. In the latter case, we compute the average string edit distance for all traces in the abstract main log and in case of a high similarity among the traces (>50%) we mine the log procedurally using a procedural miner. In case of a lower similarity, we mine it using a declarative miner. The use of the string edit distance is a simple way to distinguish between structured and unstructured logs. We validated this approach based on experiments on synthetic logs. The results of these experiments have shown that traces in structured logs are more similar to each other with respect to traces in an unstructured log. Of course, more sophisticated techniques can be used for discriminating between them.

Creating a Hybrid Process Model. When all mining tasks have finished, we can combine the resulting process models into a single hybrid model, based on which abstract activities in the top-level model correspond to which sub-process. The exact method will depend on the miners used and the languages that they use to generate models. In our implementation, we simply generated separate Petri nets and Declare models. However, to improve usability, a tool that supports the visualization and management of such hybrid models would be needed (for example, to graphically represent a Declare sub-process within a top-level Petri net model). At this point, this is left for future work.

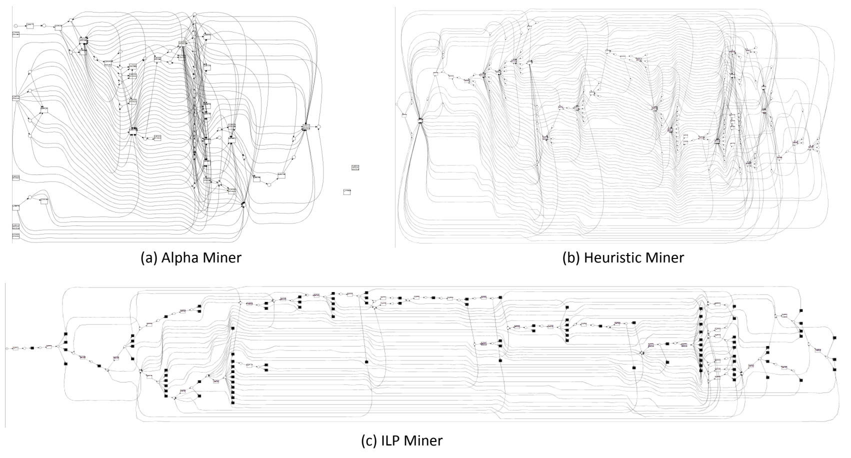


Fig. 2. Procedural Models

4 Evaluation

To evaluate our approach, we have implemented it as a plug-in of the process mining tool ProM.[1] For the evaluation, we turned to the real-life event log, which was made available as part of the BPI Challenge 2012.[2] The process represented in the event log is an application process for a personal loan within a global financing organization. The log itself contains some 262.200 events in 13.087 cases.

Our evaluation took on the following form. We set out to compare a model that would result from a traditional mining approach on the selected log with a hybrid model that is generated as proposed in this paper. We will refer to these as the *procedural* and the *hybrid* models. The aim then is to compare these models specifically with respect to the understandability of the generated models. We decided to create three procedural models of the event log by using the Alpha, Heuristic, and ILP miner, respectively. The resulting procedural models are shown in Fig. 2.

We created the hybrid model by using the Declare miner on the clusters of unstructured sequences, while using the Heuristic miner on the clusters of structured sequences. Since the root model in this case also could be classified as structured, it was mined with the Heuristic miner as well. The hybrid model can be seen in Fig. 3. In this figure, the procedural root net is shown, as well as links to its sub-nets. Note that two sub-nets are of a declarative nature (D1.1 and D2.1); the other sub-nets are procedural.

To make sure that a comparison with respect to the simplicity of the various models is fair, we first reflect on their *fitness* [13]. This expresses how well the model is able to “replay” the observed behavior in the log. The values are provided in Table 1. As can be seen, the fitness values for the procedural models range from 0.01 for the ILP

[1] `http://www.promtools.org/prom6/HybridMiner`
[2] `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`

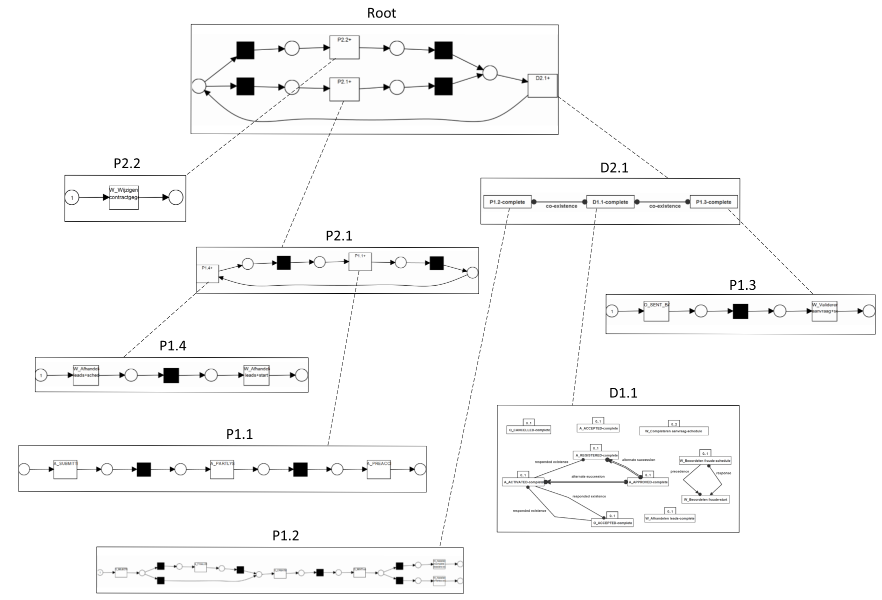


Fig. 3. Hybrid Model

miner to 0.58 for the Alpha miner. For the hybrid model, the fitness values are provided for each of the sub-nets. These values vary from 0.69 to 1.00 (perfect fitness). Without an integrated fitness measure available for hierarchical nets, we propose to take the minimum value as a conservative approximation for the fitness of the hybrid net. On this basis, the replay fitness of the hybrid net can be seen to be at least as good as that of the procedural models. Also, it is not particularly "flower-like", which can be a drawback of aiming at a well-fitting model [13].

A visual inspection of the models seems to indicate that the hybrid model is vastly simpler than the procedural models. All procedural models can be characterized as "spaghetti-like". The hybrid model, by contrast, is composed of 9 different sub-nets, each of which having a fairly simple structure. Arguably the most difficult of these sub-nets are P1.2 and D1.1, which are the largest procedural and declarative sub-nets, respectively. While the modularity of the hybrid model to some extent seems to help the understanding of the process, the overall lack of visual clutter is apparent. The proposed

Procedural			**Hybrid**								
Alpha	**Heuristic**	**ILP**									
			Root	P1.1	P1.2	P1.3	P1.4	P2.1	P2.2	D1.1	D1.2
0.58	0.40	0.01	1.00	0.84	0.73	0.69	0.81	1.00	0.86	1.00	1.00

Table 1. Fitness values for the generated models

approach, therefore, seems to have the potential to automatically generate behaviorally accurate process models that are simple to read.

5 Related Work

Several approaches in the literature focus on the discovery of declarative process models [1,4,2,5,6,7,8]. The algorithms proposed in [4,2,6,8] are tailored to discover Declare specifications. In particular, the technique proposed in [4,2] is based on a two-step approach. First, the input event log is parsed to generate a knowledge base containing information useful to discover a Declare model. Then, in a second phase, the knowledge base is queried to find the set of constraints that hold on the input log. The work proposed in [6] is based on an Apriori algorithm for association rule mining and has been used in this paper for the discovery of the declarative sub-processes of a hybrid model. The approaches proposed in [1,5] are more general and allow for the specification of rules that go beyond the traditional Declare templates. However, these approaches can be hardly used in real-life settings since they are based on supervised learning techniques thus requiring negative examples that are difficult to be derived from real data. In the work proposed in [7], a first-order variant of LTL is used to specify a set of data-aware patterns. Such extended patterns are used as the target language for a process discovery algorithm to produce data-aware Declare constraints from raw event logs.

A recent implementation of a hybrid process modeling technique is made available in CPN Tools 4.0 [15]. Different than what is proposed in the paper at hand, a hybrid CPN net allows for the use of procedural and declarative modeling elements within the same sub-process. Already at an earlier stage, modeling approaches have been proposed that embrace “pockets of flexibility”. Specifically, in [11] it is proposed to define at build-time in a workflow process pockets in a way that is highly similar to a declarative style to match their highly flexible behavior; at runtime one has to pick a specific procedural instantiation of the workflow that fits the definition. Two other approaches that combine procedural and declarative elements worth noting are Flexibility-as-a-Service (FAAS) [14] and the Guard-Stage-Milestone model [12]. It should be noted that for none of these approaches automated discovery techniques exist.

6 Conclusion

In this paper, we presented an automated discovery technique for hybrid process models. By analyzing the traces that are available in an event log and clustering them together according to their structure (or lack thereof), we are able to mine the structured and less structured pockets within a process with procedural and declarative mining algorithms, respectively. The result is a hierarchical process model with both procedural and declarative sub-processes. Our evaluation on a real-life event log suggests that the proposed technique is indeed capable of producing a much simpler representation of a process than traditional, purely procedural approaches can.

The proposed approach could be improved along theoretical, technical, and empirical angles. On a theoretical side, there is a need to establish proper metrics that tie to the established quality dimensions of fitness, precision, generalization and simplicity [13] for hybrid, hierarchical process models. At this point, it is not entirely clear how a quality measure for a subprocess propagates to the quality of the overall model. Establishing this will pave the way for a more thorough insight into the strengths and weaknesses of the proposed discovery technique. Technically, a step ahead would be to allow for *duplicate events*, i.e. the same event can be part of a procedural as well as a declarative sub-process. We did not allow for this at this point, but this could be done by identifying “recurrent" predecessors/successors even if these appear only in a certain percentage of cases. From an empirical angle, end users need to be confronted with hybrid models for a thorough evaluation of their usefulness and ease of use.

As to stimulate the uptake of hybrid process models, a number of other developments are called for as well. As we pointed out in our earlier work [10], modeling guidelines and tool support will be essential to allow for the manual creation and maintenance of hybrid process models. We are currently experimenting with such guidelines and our initial insights are that modelers with an intermediate experience with procedural modeling approaches do not find the composition of hybrid models all that difficult. We hope to report on more substantial insights in the near future.

References

1. F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Exploiting inductive logic programming techniques for declarative process mining. *ToPNoC*, 2009.
2. Claudio Di Ciccio and Massimo Mecella. A two-step fast algorithm for the automated discovery of declarative workflows. In *CIDM*, 2013.
3. Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *BPM*, 2014.
4. C. Di Ciccio and M. Mecella. Mining constraints for artful processes. In *BIS*, 2012.
5. E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *Inductive Logic Programming*, volume 4894. 2008.
6. F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative models from event Logs. In *CAiSE*, pages 270–285, 2012.
7. F.M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali. Discovering data-aware declarative process models from event logs. In *BPM*, pages 81–96, 2013.
8. F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-guided discovery of declarative process models. In *CIDM*, pages 192–199, 2011.
9. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC 2007*, pages 287–298, 2007.
10. H.A. Reijers, T. Slaats, and C. Stahl. Declarative modeling – An academic dream or the future for BPM? In *BPM*, pages 307–322, 2013.
11. S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378, 2005.
12. R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160, 2011.
13. W.M.P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
14. W.M.P. van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg. Flexibility as a service. In *Database Systems for Advanced Applications*, 2009.
15. M. Westergaard and T. Slaats. Mixing paradigms for more comprehensible models. In *Business Process Management*, pages 283–290. 2013.