IT University of Copenhagen

Foundations of Communication-Centred Programming

Calculi, Logics & Types

Hugo-Andrés López-Acosta

Ph.D. Dissertation

IT University of Copenhagen January 2012

IT University of Copenhagen

Foundations of Communication-Centred Programming

Calculi, Logics & Types

Hugo-Andrés López-Acosta

A PhD Dissertation
Presented to the Faculty of the IT University of Copenhagen
in Partial Fulfillment of the Requirements of the PhD Degree

Advisor :	Thomas T. Hildebrandt	-	IT University of Copenhagen
Examinators :	Carsten Schürmann	-	IT University of Copenhagen
	Hans Hüttel	-	Ålborg University
	Nobuko Yoshida	-	Imperial College London

IT University of Copenhagen January 2012

Foundations of Communication-Centred Programming

Abstract: Service and process-oriented systems are relatively new technologies that promise effective business processes and flexible and adaptable enterprise IT systems. The key factor in service systems is the ability to decompose a business process into a distributed system, where each participant implements parts of the functions in a business process, and interactions between participants are performed via message passing. In recent years, service- and process-oriented applications have rapidly become the norm for distributed enterprises and they have led to a number of new programming languages and standards, collectively referred to as *communication-centred programming*. Despite their adoption, these languages and standards for communication-centred programming are still young and unstable, and not grounded on a solid theoretical foundation.

There are at least two significant dimensions when describing communication-centred programs: First, the global/local views used to describe interactions, and second, the imperative/declarative specifications styles used. With respect to views: a global view considers the system as a whole, describing specifications as sequences of message exchanges among participants, and a local view describes the system as a concurrent composition of processes, implementing each participant in the system. While the global view is what is usually provided as specification, the local view is a necessary step towards a distributed implementation. On specification styles: If processes are defined imperatively, then the control flow is defined explicitly (e.g.: as a flow graph of interactions/commands). In a declarative approach processes are described as a collection of conditions (e.g.: logical formulae) they should fulfill in order to be considered correct. Until now, research in these two dimensions have evolved rather independently from each other.

This dissertation collects works devoted to foundational studies in communication-centred programming. Specifically, the dissertation revolves around *process calculi* as the main analytical tool for service-oriented systems. Process calculi are formal languages conceived for the description and analysis of concurrent systems, providing a *rigorous framework* where distributed systems can be accurately analysed, by means of *reasoning techniques* to verify essential properties of a system. By means of process calculi, we provide formal relations between global and local views, and declarative/imperative specifications. This is achieved by extending previous works on the area with additional information in model specifications, like timing constraints and compensable behaviour. Finally, we provide process specifications with reasoning techniques (specification logics, type systems, simulation techniques) that allow one to *verify* the behaviour of a service specification with respect to trustworthy properties in the system.

Keywords: Process Calculi, Service-Oriented models, specification logics, type systems.

Acknowledgments

This thesis is seldom the result of the work of a single person. Since I started learning about Concurrency Theory back in 2004, many people have contributed, inspired and supported my work. The document you are holding now summarises some of the results achieved in these years, and it they would have clearly not existed if it was not because of the influence of wonderful people surrounding me.

I am extremely greateful to Thomas Hildebrandt and Marco Carbone for their patience and support during my studies in the IT University. Thomas gave me sound advice and had enough patience to let me work in my own research ideas, no matter how disparate they were. This freedom allowed me to start paths far from what it was originally the ideas basing my Ph.D., which later evolved in some of the articles here presented. I am most grateful to him for such liberty.

I met Marco Carbone back in 2004 in Paris at the LIX Colloquium of Concurency theory, and I would have never imagined how influential would he be in the selection of my research path. His forebearing, visions and support were definitive for finishing this thesis. Even though he was not my direct supervisor, Marco and I spent long hours having scientific (and other) discussions, and his way of approaching research is definitely one of the biggest assets I take from my stay at ITU. Above all, I thank Marco, and his family, for their invaluable friendship.

Special thanks go for Jorge A. Pérez. He has been my colleague and a supporting friend since I started doing research back in Colombia, and we have shared uncountable discusions on Concurrency Theory and life at a large. Jorge appears as coauthor in only two papers in this dissertation. His contribution was, however, greater than what that you have might perceived. Research with Jorge is exciting, as he can see both the "big picture" and work methodically on the details, a quality that few have.

My approach towards research is directly linked with friendship, and I have profound admiration for my coauthors along this time. They are all extremelly talented people, and I have learnt a great deal from the interaction with each of them. Thanks to Davide Grohmann, Carlos Olarte, Gian D. Perrone, Andzrej Wasowski, Nicola Zannone, Fabio Massacci, Jorge A. Pérez, Marco Carbone and Thomas Hildebrandt.

As part of my Ph.D. I made a short visit to the Laboratoire d'Informatique de l'École Polythecnique-Paris, and a longer visit to the Department of Informatics of the University of Lisbon. I wish to thank both teams and my hosts (Frank Valencia and Vasco Vasconcelos) for providing exciting research environments and constructive criticisms that nourished this research. LIX is clearly an inspiring place to do research, and my discussions with Frank, Carlos Olarte and Catuscia Palamidessi gave me a better understanding on Concurrent Constraint Programming languages. The group at the University of Lisbon, with Vasco, Dimitris Mostrous and Kohei Suenaga, is expert in type theories, and our many discussions inspired me in a great extent to explore the connections between logics and types. I owe special gratitude to Vasco, who had always time for doing research with me despite how busy he could

be.

My time as a Ph.D. student ends in Denmark, but it started in Italy back in 2006. From my time in the University of Trento I wish to thank Fabio Massacci, and Paola Quaglia. Fabio inpired my research in business processes and service-oriented technologies, which I had no idea would be so exciting. To Paola goes my deepest gratitude, for being an inspiring figure, a talented scientist and a supporting friend, who helped me with her advice on making the right decisions in the proper time.

I would like to express my appreciation to the excellent research environment at the IT University of Copenhagen, specially in the Programming, Logics and Semanics group. People in PLS helped me all along my period as a Ph.D. student, sharing their views about research, reading and correcting my manuscripts, giving me constructive critics, inspiring my work or even distressing the environment with a soccer match or a glass in the friday bar. Special thanks go to Espen Højsgaard, Troels Damgaard, Ebbe Elsborg, Fabrizio Montesi, Davide Grohmann, Gian D. Perrone, Søren Debois, Claus Brabrand, Jacob Thamsborg, Adam Poswolsky and Jeffrey Sarnat.

Many people expressed interest in the research I performed, and I am most greateful for their comments on each of the papers. I wish to express my special gratitude towards Kohei Honda, Nobuko Yoshida and Martin Berger, that read, commented or discussed with me different aspects of the work here presented.

I owe much to Camilo Rueda and Frank Valencia for inspiring me on doing research in the academia. They are both extremely humble people despite their outstanding research, and they both provide me advice on my career path, way before I decided to explore the field of concurrency theory. Frank played an important part in my life, and his interests in finding relations between logics and concurrent programs also shaped the motivations of this thesis. It is fair to say, that without them, this thesis would have not existed.

I am very proud of being part of a generation of people showing that research in (theoretical–but not only) computer science in Colombia not only *is possible*, but can provide fruitful results. Despite our different interests, this small group of colombians is working hard towards the construction of a better country by doing top research in their own specialities. Many thanks to Julian Gutierrez, Jorge A. Pérez, Alberto Delgado, Alejandro Arbelaez, Andrés A. Aristizábal, Joel Granados and Juan D. Hincapié for making such effort.

Doing research in academia is in general not an easy task, not only because it is a complex and highly-competitive environment, but also because it also involves living abroad and get dettached from your roots. My friends have helped me to endure such situations and allowed me continue along, even in moments I felt I would not have managed. They did not need to know anything about my research, but supported me as a human being, which is a great quality. My gratitude goes for the Segata-Gasperetti family, Gabriele Sarli and Francesco Nesta in Italy, Haideer Miranda in Costa Rica, Rodolfo Cartas in Mexico, Alejandro Fuentes across the Øresund bridge, Natalia Kuraszynska, Jon Springborg, Mikkel Hall in Denmark, and Franklin J. Valverde, Juan M. Casas, Mónica Larrahondo and Emma Sánchez in Colombia. Also in this "human department", I want to thank all those tango, salsa and kizomba

dancers in my life. It will be very honest to say, that dancing kept my sanity along these years, by connecting me back with the human factor that academia sometimes forgets.

Last, but by not means least –quite the contrary, actually– I want to dedicate this thesis to my loving family, who provide me unconditional support, trust and peace of mind in the moments I needed the most. To my father Victor Hugo, my mother Amparo and my sister Verónica Isabel, this is for, and because of you.

Hugo-Andrés López
Copenhagen, 31st January 2012.

Contents

List of Figures

Chapter 1

Introduction

What is the relationship between the ways a physician, a chemist, or a salesman organise their daily work practices? Their days are organised in terms of task and activities, normally involving some sort of collaboration with computational systems and other colleagues inside or outside the organisation, such as, generating quotes of a given product to a possible customer, performing a biological experiment and integrating the results with the database, or prescribing medicine according to patients' tests. Ultimately, these interactions must necessarily be structured in a meaningful way. This thesis studies the ways in which computational interactions underlying most of our daily work activities are structured. In particular, we concentrate on *Communication-Centred Programming*, or the programming language techniques used to describe such interactions. We provide contributions on *process calculi* for Communication-Centred Programming, and *reasoning techniques* to ensure properties of the reliable performance of distributed systems. The hypothesis is that specifications of Communication-Centred programs currently have different, interrelated visions, both in the level of abstraction used to define interactions as well as the programming language paradigms used when defining them. We believe it is of utmost importance to define exactly how these visions are interrelated.

1.1 Motivation

More than forty years separate us from the beginnings of the Internet, a medium that changed drastically the way we communicate, interact, and collaborate across the globe. From mechanisms to support democratic elections up to meshes of networks collaborating in the execution of biological experiments, it has reshaped the way we perceive the world. The impact has been so deep that in June 2011 the United Nations declared Internet access as an inalienable fundamental right to which a person is inherently entitled simply because she or he is a human being [La Rue 2011].

Such advances do not come alone, and considerations regarding how to provide dependable infrastructures must be examined when defining ways of interacting over the Internet. From the security of personal data to guarantees regarding the correct execution of a distributed transaction, the "future Internet" poses questions in terms of scalability, mobility, flexibility, security, trust, and robustness to the more than forty year old current Internet architecture. As outlined in [European Commission 2007], a vast landscape of application and ever-changing requirements and environments must be supported, and new means of interaction must be devised, coping with safety and reliability in their coordination methods.

1.1.1 Communication-Centred Programming

Given the loosely-coupled, highly distributed nature of systems over the Internet, this thesis revolves around the central theme of *Communication-Centred Programming* [Carbone *et al.* 2007]. In Communication-Centred programming, the focus lies in designing the communication protocols of distributed systems in such a way that, when deployed together, they will behave according to a specific control flow.

In this way, Communication-centred programming captures related concepts like (i) Business Processes, (ii) Workflow models, and (iii) Service Oriented Computing:

Business processes: Business processes describe a series of structured and automated interactions among business entities. It is predominantly inter-domain, regulation-based, and demands that involved stakeholders have a clear shared understanding of its meaning. Because of its inherent inter-organisational nature, a business process typically describes interactions at a *global level*, where different stakeholders may be involved.

Workflow models: The development of organisational-level business processes led to workflow management technologies. The important achievement of workflows is the explicit representation of process structures in process models and the controlled enactment of business processes according to these models. Workflow technologies provide a framework for the specification and automation of processes by means of activities respecting a business logic. Compared to business processes, process descriptions in the workflow are more concrete, and one can see workflows as automation of business processes in which tasks and information are contingent upon other for action, according to a set of procedural rules.

Service-Oriented Computing: Finally, Service-Oriented Computing (SOC) opens a new, different horizon by distributing the places in which the business logic is defined; now, small process units (services) can be shared between different organisations, so each of them can fulfil their business goals by reusing and outsourcing services. Compared to business processes, services are specified at a *local level*, describing only the viewpoint of one component in the system. Compared to workflows, services are flexible entities where the component providing a service can be substituted in a transparent way, without requiring any further adaptation.

An illustration of how these levels are related might help to understand their importance. Let us consider an on-line booking scenario. On one side, consider an on-line broker company which has deals with airlines, hotels and car rentals. On the other side, there is a customer looking for the best offers for his upcoming holiday trip. We can informally describe an excerpt of the sequence of allowed interactions as follows:

1. Customer establishes a communication with the on-line broker;

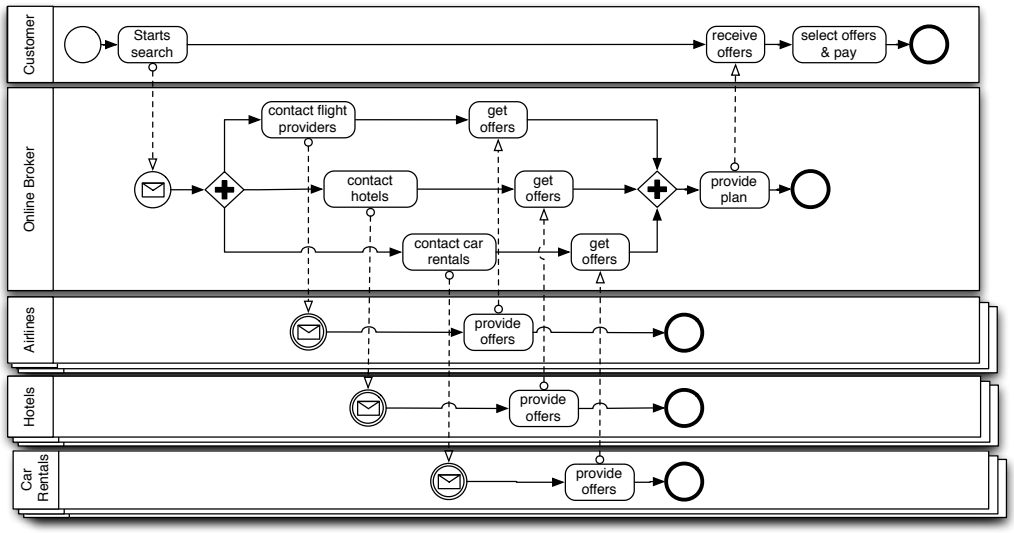


Figure 1.1: Example of a Business Process: an on-line booking scenario

2. Customer asks the broker for a holiday plan including flight tickets, and bookings for hotels and cars, given a set of constraints;

3. The on-line broker establishes a communication with his partners serving the destination and respecting the constraints imposed by the customer;

4. After receiving offers, the on-line broker compiles plans and forwards the plans to the customer;

5. Customer analyses the plans given, and selects the best offer.

A business process describing the given scenario will pay attention to the set of roles involved, and the sequence of interactions between such roles. Figure 1.1 describes the participants, activities, and control/message flows in this scenario[1]. Here, the description is rather abstract, and details about the communication primitives used and synchronisation methods are not specified. A business process then expresses the "global view", providing a specification of the whole system.

A workflow model will refine the specification of the business process, including the way in which each activity is described, adding more detail to the activities and communication flows. A workflow model is intended to be executable, and therefore its descriptions must be precise enough to be followed. Returning to the example,

[1]We use the Business Process Modelling Notation (BPMN) [Object Management Group 2011] just as an illustrative tool to outline the model in this process.

the workflow will specify precisely which communication protocols must be followed for each communication between the on-line broker and its associated airlines. The workflow hard-wires each activity, and if one needs to change the technologies supporting any of the activities, then it will be necessary to adapt the workflow accordingly.

In the last level, services provide a flexible, scalable, and technology-agnostic infrastructure. Every task of the business process is encapsulated by a service, deployed at the site of one of the participants. The "local view" refers to this deployment: located services posses only information about their local state, and connections to other services they can use. Each service is published in a *service repository*, that allows later linkage with other services, provided that they have compatible interfaces. Compared to workflow technologies, the use of service repositories adds a degree of flexibility: while in the workflow changes in technologies require adaptors that connect old and new components, services act *transparently*, that is, they do not require previous knowledge about the underlying technologies used to implement the service in order to use it (that is, as long as they conform to the same interface).

1.2 Specifying Communication-Centred Programming.

Giving the inherent complexity of analysing services in distributed environments, one normally uses different abstractions to describe and reason about them. One such abstraction deals with the the study of the *concurrent* nature of services. *Process calculi* are formal languages for the description and analysis of concurrent systems. As such, the goal of a process calculus is to provide a rigorous framework in which complex systems can be accurately analysed, including reasoning techniques (type systems, specification logics, behavioural relations) to verify essential properties of a system. The work of [Honda *et al.* 1998] provided a new view of Communication-Centred programming in light of a process algebraic approach. The term *structured communications* refers to the branch of process calculi devoted to the analysis of interactions between services. In a calculus for structured communications, one considers the computation within a service as an atomic activity, and focuses the core of the analysis on the interactions between services.

One of the most important aspects when modelling services relates to the notion of *trustworthiness*, or the extent to which users believe that the systems behave correctly. A *safe* system is one in which a property considered harmful for the proper functioning of the system will never happen, e.g., the disclosure of the private credentials of the manager to a thief.

Despite of being such a recent trend, different but related views for the analysis of service oriented systems have been proposed. We can characterise such approaches in two dimensions: *global*/*local* views of services, and *imperative*/*declarative* specifications.

1.2.1 Global vs. Local Views

The first dimension relates to the granularity of abstraction used to describe interactions; either one describes the system as the exchange of messages between different participants, or one considers the system as the composition of the local behaviours of each participant. In this first view, known as *choreography* [Kavantzas *et al.* 2004], one considers the system as a whole, taking care only of the interfaces that participants use when interacting with the outside world. Choreographies help to describe the scenario in which all actors are involved in the communication process, defining where and when a communication has to happen. A designer of orchestrations decides that e.g., there will be a message from the Customer to the On-line Broker without further considerations on how this will be implemented by the Customer (sending a message) or at the On-line Broker (expecting to receive a message). This level of abstraction has already proved useful from a software development perspective, and approaches coming from both the theoretical side [Brogi *et al.* 2004, Montangero & Semini 2006, Su *et al.* 2007, Carbone *et al.* 2009, Carbone *et al.* 2010] and the development of industrial standards [Kavantzas *et al.* 2004, Object Management Group 2011] for choreographies have been studied.

The second view, known as *orchestration* [Misra & Cook 2006], instantiates the global view described by a choreography to a distributed view where participants (also called *end-points*) implement the communication strategies dictated by the choreography. One can see the vision of an orchestration as one in which the system is perceived by the eyes of each participant, sending and receiving messages but not knowing which other actors are present during a communication. Both choreographies and orchestrations assume autonomous scenarios, that is, they do not require a single point of control. This assumption, although realistic, greatly complicates the correct mapping from choreographies to orchestrations. Flavours of orchestration models abound [Lapadula *et al.* 2007a, Lanese *et al.* 2007, Hongli *et al.* 2007, van Riemsdijk & Wirsing 2007, Bruni & Mezzina 2008, Vieira 2010], and we will discuss them in the next section. As previously stated [Carbone *et al.* 2007, Lanese *et al.* 2008, Hongli *et al.* 2007], choreographies and orchestrations can be operationally correspondent, and one can project a choreography to generate distributed orchestrations that implement it.

1.2.2 Imperative vs. Declarative Specifications

The second dimension refers to the approach used to construct the models. Descriptions can have imperative or declarative flavours: In an imperative approach, one explicitly defines the control and communication flow of activities. These kinds of specifications are the most commonly used nowadays, and typical representatives of this approach are based on Petri nets and process calculi, as well as diverse industry standards. An imperative specification describes "how the process should behave", defining the set of control structures, synchronisation methods, and activities involved in a specification. Imperative models require a *total* knowledge of the system being

specified, and the flexibility of imperative specifications is highly dependent on the constructs included in the modelling language used. Finally, changes to models derived from imperative specifications might be costly, as one needs to re-engineer the model from scratch.

By contrast, in a declarative approach the focus shifts to the specification of the set of constraints (causality relations, time constraints, quality of service) processes should fulfil in order to be considered correct [Nørgaard *et al.* 2005, Pesic & van der Aalst 2006, van der Aalst & Pesic 2006, Lyng *et al.* 2008, Rychkova *et al.* 2008]. The idea in a declarative approach lies in defining "what are the requirements?" instead of "how should we model them?". The idea of declarative specifications emerged from a generation with a need for flexible and rapidly-evolvable processes [Heinl *et al.* 1999], and require only a *partial* knowledge of the system specified. When working in a declarative language, the users are driven by the system to produce the required results, while the manner in which the results are produced depends on the preference of the users. Adapting the system to new requirements requires significantly less work than in imperative specifications: The constraints representing the new requirements are added to the original specification, and the system will be valid as long as the added constraints do not create inconsistencies with the original specification. Declarative specifications can also be seen as the definition of policies about the behaviour of the system, and they normally have a logical foundation, for instance, a formula in Linear Temporal Logic [Manna & Pnueli 1992].

Even if these two trends address similar concerns, we find it interesting that both styles have evolved independently of one another. Imperative and declarative specifications need not be mutually exclusive. For instance, consider the healthcare domain: a hospital specifies all possible treatments for different illnesses in special kinds of workflows, also called *clinical protocols*. Clinical protocols are quite specific regarding the actions taken by each of the actors involved in patient treatment, also describing the sequences in which each of the members involved in the patient treatment will engage in the protocol. On the other hand, *clinical guidelines* are the de-facto assessment tool regarding the compliance of a hospital with good practices in the medical sector. Clinical guidelines are, as defined by the American Institute of Medicine, "systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances"[Field & Kathleen N. Lohr 1990]. As a clinical protocol depends greatly on the infrastructure and personnel a health care institution has, clinical guidelines define only sets of actions that *should* be performed. We can observe here, that clinical protocols are imperative specifications, while clinical guidelines are their declarative counterparts.

1.3 This Dissertation

This thesis focuses on the development of foundations in which connections between the four different dimensions describing Communication-Centred Programming can be formalised. In the following we will discuss how this can be done. Our approach

relies on concurrency theory, in particular we specialise in *process calculi*.

1.3.1 Process Calculi

Process calculi (also known as *process algebras*) are formalisms devised for the description and analysis of the behaviour of *concurrent systems*. As such, the goal of a process calculus is to provide a *rigorous framework* where complex systems can be accurately analysed, including *reasoning techniques* to verify their essential properties. We first discuss some basic principles of process calculi, including several issues that distinguish them from other formal models for concurrency and the main approaches to give meanings to processes.

The nature and features of concurrent systems occurring in the real world makes difficult the task of finding a canonical formalism in which *every* system can be accurately represented. In fact, even in the context of the restricted field of Communication-centred Programming, a wide variety of different phenomena, occurring at different levels of abstraction, can be recognised. The goal then is to identify a common set of *underlying principles* in the systems of interest, and to define suitable operators that capture them in a precise way. In other words, a process of *abstraction* is required to define meaningful calculi in the simplest possible way.

Process calculi are then *abstract* modelling languages for concurrent systems. This implies that models of systems abstract from real but unimportant details that do not contribute to essential system interactions. Abstraction not only allows designers to better understand the core of a system, but it also turns out to be necessary for an effective use of reasoning techniques associated with the calculus.

In addition, process calculi follow a *compositional* approach for systems description. This implies that a process calculus model of a system is given in terms of models representing its subsystems. This favours an appropriate abstraction of the main components of the systems and, more importantly, allows one to explicitly reason about the *interactions* among the identified subsystems. As we will see later, each calculus assumes a particular abstraction criteria over systems, which will influence the level of compositionality that models exhibit.

Whilst industrial modelling languages provide a broad set of language constructs, process calculi pay special attention to *economy*. There are few process constructors, each with a distinct and fundamental rôle in capturing the behaviour of systems. A reduced number of constructors in the language helps to ensure that the theory underlying the calculus is tractable, and encourages a precise definition of the abstraction criteria that the calculus aims to express. Interestingly, research shows that when it comes to standards for Communication-centred Programming, a broader corpus of the modelling language can be encoded into core subsets [Højsgaard & Hallwyl 2012], and that current usage of industrial modelling standards tends to restrict the language constructs used to a small but expressive subset [Muehlen & Recker 2008], akin to the motivation behind having fewer constructs in process calculi.

Besides giving a solid framework for the description of process behaviours, one of the main advantages of process calculi lie in the provision of reasoning methods

where concurrent models can be studied. In this thesis we concentrate our studies in three reasoning techniques: behavioural relations, specification logics, and type systems.

Behavioural Relations: Generally speaking, behavioural relations formalise the set of criteria that must be respected when comparing two concurrent systems. When comparing a specification and different possible implementations of a system, for instance, we will require that the implementations considered have *at least* the same behaviour as that of the specification. At the same time, when comparing implementations, we would like them to have *an equivalent* set of behaviours. Relations between models at different levels of abstraction are normally captured by pre-orders, while equivalences capture the relations between models at the same level. The granularity used to define the criteria in a behavioural relation will determine how strong the relation is, and a number of different equivalences and pre-orders have been proposed based on which aspects of system behaviour should be observable.

Specification Logics: When describing the correctness of a system with respect to a specification using behavioural equivalences such as observational equivalences, we are in many cases forced to specify the overall behaviour of the system. A specification logic allows for verification of properties regarding *partial specifications* of the system. Most of the interesting properties in concurrent systems fall into two categories: *safety properties* ("something bad never happens in the system"), and *liveness properties* ("something good eventually happens") [Lamport 1994]. The verification of these kind of properties in a process specification is carried out by exploring the state space of the system, rather than by equivalence checking. A logical semantics gives the meaning of process interactions in terms of a language with well-defined and intuitively understandable semantics. Usually, this is done by providing an interpretation of the process constructors in terms of a logic, such as Modal Logics and Temporal Logics. A specification logic then allows a user to verify properties via exploration of the state space of the process in question (Model checking).

Type Systems: Another way of specifying and analysing the behaviour of a process specification is by means of *type systems*. A type system is a syntactic method that restricts the behaviour of specifications such that they fulfil a certain set of properties. In Communication-centred Programming, for instance, we use type systems to guarantee that processes will adhere to a certain protocol communication strategy. Another example is security protocols, where we can use type systems to limit the power of a language so it does not allow attackers to infer private information.

1.3.2 Approach

This research has as a main objective to build formal connections between the different dimensions (global vs. local and imperative vs. declarative) in specification

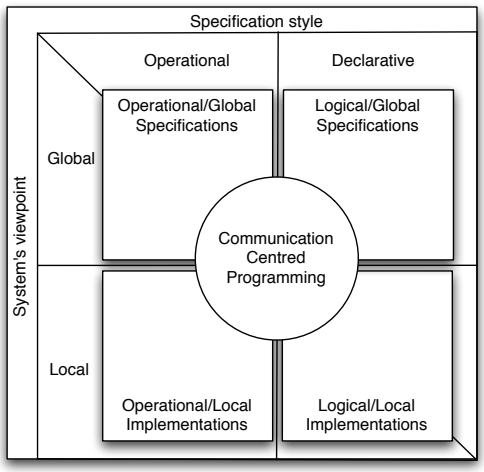


Figure 1.2: Visions of Communication-Centred Programming

and verification of Communication-centred Programs. Rather than proposing a novel modelling language featuring imperative/declarative visions, this thesis explores the connections between well-established approaches for the study of interactions in message-passing concurrency. We believe it is in this way that we can achieve a faster adoption in the techniques and methods presented here: A business process described in an existing modelling language such as the Web Services Choreography Description Language (WS-CDL) [Kavantzas *et al.* 2004] can be directly checked against its declarative counterpart without requiring the model to be transformed. Our goal is to present characterisations of processes, both at the *operational* and *logical* level. This is done by relating the way processes are specified, both from their global and local viewpoints. Figure 1.2 illustrates the approach for the specification and verification of Communication-centred programs. A specification of a process in its global view (choreography) can be projected to the distributed execution of its end points. Similarly, every global specification corresponds to a formula in a modal logic representing the interactions between agents. A modal logic for global specifications not only provides the logical characterisation of a process; it also allows for *partial specification*: Given a logical formula, one can see if there is a process in the global specification that can satisfy it.

A similar reasoning ability is provided for distributed implementations (end-points): given a set of end-points defining how services interact, one is interested in describing the behaviour of its composition. Moreover, a formula representing the global behaviour of a choreography can be projected to a corresponding formula describing the behaviour of a set of end-points.

In this thesis we mainly explore different process calculi for the analysis of Communication-Centred Programs. Here we proceed to list some of the languages which we build upon:

π-calculus for Session-Based communication:

In Communication-Centred Programming, a *session* is a logic unit of information exchange between two or multiple communicating agents [Dezani-Ciancaglini & De'Liguoro 2010]. Starting from Milner's π-calculus [Milner *et al.* 1992], we explore different variants of name-passing calculi with support for in-session communication. The calculus presented in [Honda *et al.* 1998] is probably the first attempt towards a rigorous approach in Communication-Centred Programming. The calculus (here referred as HVK after the initials of their authors) is a language for orchestrations that features primitive treatment for session-based communication, including session establishment, data communication and session delegation primitives, as well as a typing discipline that ensures that communicating processes always follow safe communication patterns. In further works, we use the duality between choreographies and orchestrations presented in [Carbone *et al.* 2007]. There are two typed calculi for interactions: a distilled version of the Web-Services Choreography Description Language WS-CDL (here referred as the Global Calculus) and an applied π-calculus with locations for end-points (so called the end-point calculus). The global calculus directly describes interactions among multiple participants involving sequencing, branching, and recursion, which differs from the end-point-based descriptions given in the π-calculus, that describe orchestrations and the causal relations between messages.

Conversation Calculus:

The Conversation Calculus (CC) [Vieira *et al.* 2008] corresponds to a π-calculus with *labelled* communication and extended with *conversation contexts*. A conversation context can be seen as a medium in which interactions take place. It is similar to π-calculus variants for session-based communication in the sense that every conversation context has an unique identifier (e.g., a session). Interactions in CC may be intuitively seen as communications in a pool of messages, where the pool is divided in areas identified by conversation contexts. Multiple participants can access many conversation contexts concurrently, provided they can get hold of the name identifying the context. Moreover, conversations can be nested multiple times (for instance, a private chat room within a multi-user chat application).

Universal Temporal Concurrent Constraint Programming:

Concurrent Constraint Programming (CCP) [Saraswat 1993] is a formalism for concurrency in which processes interact with one another by placing and reading information represented as constraints in a shared medium. We explore the use of a variant of CCP for Communication-Centred Programming. The Universal Temporal CCP (utcc) [Olarte & Valencia 2008a] is a variant of CCP for reactive synchronous programming, with the ability to express link mobility. It does so by including a universally quantified abstraction (*ask*) operation over the syntax of the timed version of CCP. This adds the ability to extend the scope

of local knowledge which is not possible in CCP. utcc is a declarative model for concurrency: it is shown that utcc processes can be seen, at the same time, as computing agents and as logic formulae in Pnueli's First-order Linear-time Temporal Logic (FLTL).

1.3.3 Contributions

We can outline the contributions of this thesis in four main areas: *process calculi extensions*, *logical characterisation of message-passing concurrency*, *definition of type systems*, and *definition of behavioural relations*.

Process Calculi Extensions: In the search for adequate models for Communication-centred programming, this thesis focuses on the use of derivatives of process calculi with message-passing capabilities. Most of the calculi here studied are derivatives of the π-calculus with support for session handling primitives, including HVK [Honda *et al.* 1998], the End-Point Calculus (EPC) [Carbone *et al.* 2007], and the Conversation Calculus [Vieira *et al.* 2008]. With respect to these calculi, we propose extensions for the study of timed and exceptional behaviour. In particular, we present HVK^{T} [López *et al.* 2010], a timed extension of HVK that explicitly includes information on session duration, allows for declarative preconditions within session establishment constructs, and features a construct for session abortion. Additionally, we introduce C3 [López & Pérez 2012], a variant of the Conversation Calculus in which conversation contexts are enriched with a time duration, a compensating activity, and designated signals for conversation abortion.

In a parallel but interrelated branch, we studied how the concurrent constraint family of languages could be suitable for models of Communication-Centred Programming. The Universal (Temporal) Concurrent Constraint Programming (utcc) [Olarte & Valencia 2008a] is a variant of CCP that introduces a universally quantified ask operation that makes it possible to infer knowledge which is local to other agents, modelling to a certain extend the mobility necessary in calculi with support of sessions. Here we present secure utcc (utcc_s), a variant of utcc that allows the assumption of local knowledge in abstractions, limiting the power of the abstraction operator so it can be used to encode mobility as in the π-calculus.

Logical Characterisations: We strive for logical correspondences between the calculi for Communication-Centred Programming and their logical meaning. To do so, we explored the relation between Choreographies and Orchestrations and Temporal/Modal logics. Starting with an extension of Hennessy-Milner Logic [Hennessy & Milner 1980], we introduced $\mathcal{GL}$ [Carbone *et al.* 2010], a global logic for the study of choreographies. The logic is equipped with a proof system that allows for verification of properties among participants in a choreography. With $\mathcal{GL}$, one can see the state of a choreography as a formula in the logic, and one

can check for satisfaction of desirable properties by relating a logical formula with respect to a choreographic specification. The correspondence between Orchestrations and logics is given in two ways: first, the relationship between the calculus of orchestrations in HVK and Linear Temporal Logic is provided in form of an operational encoding to utcc. This way, services can be analysed in a declarative framework where time is defined explicitly, and their behaviour compared to formulae in LTL. Second, We define $\mathcal{LL}$, a Henessy-Milner logic variant describing the behaviours of end-point specifications. $\mathcal{GL}$ and $\mathcal{LL}$ are closely tied, and one can project logical formulae describing properties for choreographies in $\mathcal{GL}$, to the realisation of end-point formulae in $\mathcal{LL}$.

Type Systems: While trying to make the mapping between Concurrent Constraint languages and name-passing calculi used in Communication-Centred programming, we identified that concepts such as the communication of mobile data and access control of information flow were not easily captured. The recently proposed *universal* tcc (utcc) introduces a universally quantified ask operation that makes it possible to infer knowledge which is local to other participants. However, it allows participants to guess knowledge even if it is encrypted or communicated on private channels, simply by quantifying over both the encryption key (or channel) and the message simultaneously. This complicates the adoption of utcc as a model for mobility and secure communications. We introduce a type system for constraints allowing to distinguish between restricted (secure) and non-restricted (universally quantifiable) variables in constraints [Hildebrandt & López 2009]. The adoption of such type systems allows for a correct communication model including the transmission of local names (as in the π-calculus) and applications in security protocols.

Behavioural Relations: A question derived from the definition of logical characterisations relates to the comparison of models at different levels of abstraction (choreographies & orchestrations) using behavioural relations. We search for a *refinement relation* that can capture whether an implementation meets the requirements imposed by a specification. Two types of refinements are studied in this thesis: Firstly, we build upon definitions of testing theories [De Nicola & Hennessy 1984] and simulation techniques in order to define a set of criteria where models of session-based communication featuring timed and exceptional behaviour could be compared [López & Pérez 2012]. Secondly, we search for refinement relations that can capture the flexibility presented in our logical specifications. Here we present initial ideas towards the refinement of *Open Specifications*. An open specification has two components: a system description that presents the sequence of activities that must be performed, and "open" activities: tasks that a system may do and still conform to the specification. Here we present short notes on two initial, independent ideas towards the definition of refinement relations for open specifications.

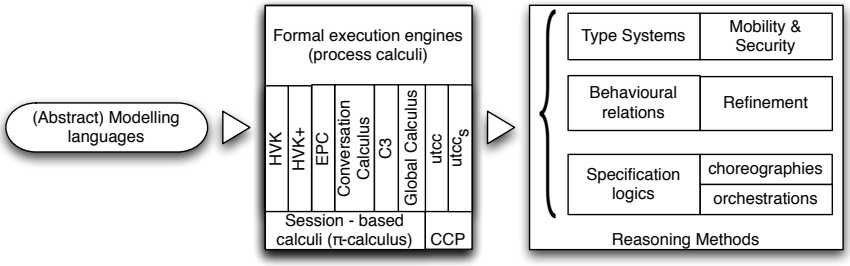


Figure 1.3: Overview of the contributions

Figure 1.3 illustrates the contributions of this thesis in terms of process-calculi variants and its associated reasoning techniques.

1.4 Related Work

Vast is the literature referring to models and techniques for Communication-Centred Programming, and they cover almost every aspect on the hierarchy of web-service modelling: From foundational calculi for web-services, such as Misra and Cook's Orc [Misra & Cook 2006], we can follow several lines of research, each of them extending basic ideas through the definition of languages equipped with service oriented primitives, dealing with Quality of Service aspects or defining security mechanisms for services, among others. In this section we aim to draw a picture describing recent advances in Communication-Centred Programming. The works here presented are influential in the whole course of the document, although each chapter has its own related work specific to that contribution. Subareas related to this research can be grouped into different categories: research in process calculi, declarative languages and type systems will be explored here.

1.4.1 Process Calculi

Works about behavioural descriptions of communications grow rapidly, and this small survey would not do justice to the diversity of calculi proposed with aims of capturing structured communications and web services[2]. We can see the research in process algebras for structured communications in three subcategories: foundational calculi for communication-centred programming, technologically-inspired calculi and, adaptations of general models of concurrency for Communication-Centred Programming.

1.4.1.1 Foundational calculi

In their seminal paper [Misra & Cook 2006], authors presented Orc, a basic programming model for structured orchestration of services. With simple primitives like

[2] A recent survey in [Bruni 2009] presents a broad view in this area.

sequential execution, symmetric and asymmetric execution, concepts of computations between different *sites* are modelled. It is not meant to be a computational model, but it provides important insights about properties of choreography between different processes. In fact, Orc has been compared with several of the standardised workflow patterns available at the moment, found it to be expressive enough despite its reduced syntactic set [Cook *et al.* 2006]. In later works [Kitchin *et al.* 2009], Orc has been introduced as a programming language, featuring constructs for concurrent threads, real-time observations [Wehrman *et al.* 2008], and mutable states. Orc has served as inspiration for further specialised calculi for service centred computations, extending the basic formalism with primitives for session handling, stream processing and pipe-lining, as occurred with the CaSPIS family of calculi [Boreale *et al.* 2006, Lanese *et al.* 2007, Boreale *et al.* 2008]. Also in a foundational approach, the authors in [Busi *et al.* 2006] explore the connections between choreographic and orchestration languages, in a minimal setting with no types.

1.4.1.2 Technology-inspired calculi

In a different approach, some calculi derived from the standards of business process languages, such as WS-BPEL [Andrews *et al.* 2003] and XLANG [Thatte 2001] where presented. Blite [Lapadula *et al.* 2008], COWS [Lapadula *et al.* 2007a] and SOCK [Guidi *et al.* 2006] represent some of these efforts. The motivation behind Blite constitutes a great example on the ambiguities one can find when modelling business processes without a formal semantics: In [Lapadula *et al.* 2008], a battery of business processes were tested in leading BPEL engines, and resulting executions exhibited quite different results with respect to their specifications and with respect to other engines. This problem not only undermines confidence in the implementations in question, but complicates the portability of business processes among different engines. Blite is designed as a simplification of WS-BPEL with structured activities, service definitions, correlation sets and compensation handlers, with a clear operational semantics amendable to formal verification. A similar approach is taken by the calculus for orchestration of web services (COWS) [Lapadula *et al.* 2007a]: While being foundational in its approach, COWS combines the notion of shared-states present on WS-BPEL with primitives of interaction already available in classic process calculi (e.g.: synchronous and asynchronous communication, pattern matching, scope extrusion) to capture complex service interactions that were not captured by Orc, such as session handling, and message passing between services. In a different view, SOCK also receives inspiration from web service specifications but conceives service oriented computing as the interaction of three different layers: the behaviour of different services, the deployment of different services in an execution environment (service engine), and a final level where the interactions between service engines can take place. Those levels are analysed with individual reasoning techniques, and the relations between different levels have been formally defined. These efforts can be considered as *technology-driven* approaches, in the sense that their conception is intrinsically linked to the available technology, and permits cross-checking sys-

tem functionalities with respect with process calculi constructions. The final goal is then to provide programming languages built on top of these calculi, such as Jolie [Montesi *et al.* 2007] or BliteC [Cesari *et al.* 2010]. Some efforts related to verification of technology-inspired calculi have been proposed, including quantitative analysis [Prandi & Quaglia 2007], and model checking [Fantechi *et al.* 2008].

1.4.1.3 Adaptation of Calculi

A different trend of research towards formal model of communication-centred programming have been the use and extension of existent formalisms in concurrency theory for the specification of service models: For example, the encoding of business processes and workflow patterns into the π-calculus [Puhlmann 2007], and long running transactions [Laneve & Zavattaro 2005] are constructed to solve reconfiguration and sessioning issues in service interactions, but since they allow unrestricted π-calculus communications the analysis is difficult, as expressed in [Lanese *et al.* 2007].

Recent works with motivations similar to ours include CC-Pi [Buscemi & Montanari 2007] and the calculus for structured communications in [Coppo & Dezani-Ciancaglini 2009]. CC-Pi combines synchronous communication and a restriction as in the π-calculus with operations for creating, removing, and making logical checks over the constraint store, as in Concurrent Constraint Programming [Saraswat 1993]. In CC-Pi, the reasoning techniques associated to CC-Pi are essentially operational, and used to reason about service-level agreement protocols. Additionally, as the authors note, important properties of the CCP paradigm (e.g.: the notion of consistency) not longer hold with the introduction of the new process constructors in CC-Pi. In [Coppo & Dezani-Ciancaglini 2009], the key for analysis is represented by a type system which provides consistency for session execution, much as in the original approach in [Honda *et al.* 1998].

Time and exceptional behaviour issues are in the scope of this thesis. Although there is a long history of timed extensions for (mobile) process calculi (see, e.g., [Berger & Honda 2000]) and the study of constructs for exceptional behaviour has received significant attention (see [Ferreira *et al.* 2010] for a recent overview), time and its interplay with forms of exceptional behaviour do not seem to have been jointly studied in the context of models for structured communication, but they have been considered only separately for orchestrations: With respect to time, Timed Orc [Wehrman *et al.* 2008] introduced real-time observations for orchestrations by introducing a delay operator. Timed COWS [Lapadula *et al.* 2007b] extends COWS (the Calculus for Orchestration of Web Services [Lapadula *et al.* 2007a]) with operators of delimitation, abortion, and delays for orchestrations; we are not aware of reasoning techniques for Timed COWS. With respect to exceptional behaviour, the work of [Dragoni & Mazzara 2010] presents a process algebraic view of compensation handling techniques in the WS-BPEL using a variant of the π-calculus for long running transactions. The work in [Hongli *et al.* 2007] presents a denotational semantics based on traces for a simple language for choreographies with exception handling and finalisation constructs, allowing a projection from exceptional behaviour of a choreography to

its endpoints. Two points are on consideration here: First, the language used for choreographies assumes that there is a principal taking the decisions about which branches to execute. Second, the semantics of the compensating blocks act much like exceptions in sequential languages, where exceptions are evaluations of expressions, and there is no treatment for nesting contexts.

1.4.2 Declarative Specifications

ConDec [Pesic & van der Aalst 2006, Pesic 2008] has been proposed as declarative language for the modelling and enacting of dynamic business processes based on LTL. In [van der Aalst & Pesic 2006], the authors have proposed Declarative Service Flow language (DecSerFlow) to specify, enact, and monitor service flows, which is a sister language for ConDec. Both languages share the same constructs and have tool support through the Declare tool [van der Aalst *et al.* 2009]. Both ConDec and DecSerFlow specify the constraints that need to be fulfilled, instead of specifying how flows should be programmed, giving more flexibility to users. The semantics of both languages is defined in terms of LTL formulae, and execution of the workflow/service flow is given by the execution of the referring Büchi automaton.

Dynamic Condition Response (DCR) graphs [Hildebrandt & Mukkamala 2010] have recently been proposed as an alternative formalism for declarative event-based workflows. Inspired by such works as the declarative process matrix [Lyng *et al.* 2008] and prime event structures, DCR graphs are conceived as a generalisation of prime event structures extended with notions of progress (condition-response relations), multiple executions dynamic inclusion and exclusion of nets. The declarative flavour of DCR graphs comes in the way a model is specified: Here, a process model is a directed graph where each node represents an event in the system, and the relation between events is given by preconditions (Condition relations), postconditions (Response relations) and dynamic inclusions and exclusions of events. In [Hildebrandt *et al.* 2011] the authors present a distribution technique for DCR graphs that projects a workflow specification in terms of their distributed end- points, demonstrating the soundness of their approach by evidencing bisimilarity between the global process and its distributed projections.

Finally, [Montangero & Semini 2006] present a logical view of choreography, formalising WS-CDL constructs in terms of a distributed-state temporal logic. Although its aim is in many respects similar to the logical characterisation we provide in Chapter 5, the characterisation provided in [Montangero & Semini 2006] is restricted, and the language does not provide session support.

1.4.3 Petri Nets

Petri nets [Peterson 1977] and Coloured Petri Nets [Jensen 1994] are one of the most popular techniques for specifying Workflows and Business Processes. One of the main selling points of Petri Nets over other formalisms (e.g. process calculi) is that it provides both a graphical notation and a formal semantics. While the graphical

notation allows for Business modellers to focus on the development of processes in a fast and intuitive way, the formal semantics allows for simulation and checking of properties of the models. A process model is then represented by the structure of a Petri net, and the execution of the process is given by the markings inside the net [van der Aalst 1998, van der Aalst 2003]. While Petri nets are feasible formalisms for the representation of simple type of workflows, more complex scenarios require specific modelling constructs related specifically to business processes. Van der Aalst proposed Workflow nets [van der Aalst 1998]. A Workflow net is a strongly-connected coloured Petri net with special places denoting start and end of procedures within a workflow, that can be organised hierarchically to abstract different levels in the specification of each process. Starting from the application of workflow nets in the industry, a guide to desirable workflow characteristics is provided by the well-known workflow patterns which are derived from a comprehensive survey of contemporary tools and modelling formalisms [van Der Aalst *et al.* 2003]. Such patterns are technology-agnostic, and they have been also used in relation to process-calculi [Puhlmann & Weske 2005] and declarative languages [Pesic 2008]. Although related to this thesis, the approach based on Petri nets places emphasis on the control flow, rather than the communicating nature of workflows, and focuses the analysis on a central (global) view rather than on distributed end points.

Recently, a language derived from Workflow nets has been conceived as an effort towards a common, comprehensive modelling language for business processes based on formal foundations. The *Yet Another Workflow Language* (YAWL for short) [Van Der Aalst & Ter Hofstede 2005, Russell *et al.* 2009] is an adaptation of the Petri net model used to describe workflow patterns, with a definition of its execution semantics in terms of state transition systems. The main aim is to define a minimal set of language constructs that could support the description of workflow patterns already identified, and can coexists inside the workflow solutions already present in the market, giving in this way a common understanding of the activities in a workflow management system, and eliminating some of the ambiguities present when each workflow management system defines its own execution semantics. This motivation is closer to our approach (and in general, with the vision of process calculi), as it defines a minimal set of language constructs with a well defined semantics where workflows can be studied.

1.4.4 Type Systems

Type disciplines to control the way interactions are performed date back to the works on sortings for the polyadic π-calculus [Milner 1991], and later in [Pierce & Sangiorgi 1993], [Takeuchi *et al.* 1994] and [Honda *et al.* 1998]. Here, π-calculus processes denote communication flows (interactions) between different parties. The communication is seen at a local level, and features constructs for message passing, labelled selection and delegation of responsibilities. Names (*sessions*) are used to ensure that interactions are different among themselves, and that processes involved in different interactions can keep a coherent track of them by the use of different

names. The type discipline (*session types*) guarantee that processes are engaged in interactions in a *structured* and complementary way. That is, if the specification of a scenario involves two participants sending and receiving information, there is a balance between dual communication primitives (inputs and outputs) and the causality relation between messages is respected.

The works on session types in [Honda *et al.* 1998] have been recently evolved in a methodology for Communication-Centred Programming. In [Carbone *et al.* 2007], a correspondence between *global types* (describing interactions at a choreographical level) and *local types* (describing interactions of end points) is presented. As described before, global descriptions can be realised by end-point specifications, in a sound and complete way that ensures that all (and only) the communication behaviour described by the global types is present in their end points.

The connections between logics and session types have been explored in different works. Here we comment on some of the most representative exponents, namely [Coppo & Dezani-Ciancaglini 2009, Caires & Pfenning 2010, Bocchi *et al.* 2010, Gordon & Fournet 2009, Berger *et al.* 2008]. In [Coppo & Dezani-Ciancaglini 2009], a calculus combining notions of concurrent constraint programming and name passing is proposed. The resulting calculus treats sessions as constraint formulae representing the requirements to be satisfied in a client-server communication, in a similar approach to the CC-Pi calculus explained above. As communications are represented as constraints, the type discipline takes account of how processes and constraints are related, guaranteeing that communications follow an structured order as in [Honda *et al.* 1998].

The relationship between session types and linear logics has been explored in [Caires & Pfenning 2010], where the authors establish a bidirectional correspondence between the session types and (dual) intuitionistic linear logic formulae. The correspondence is tight, and relates the existence of a simulation between reductions in session types and proof reductions in dual intuitionistic linear logic, and vice versa. In [Pérez *et al.* 2012], the authors make use of the linear logic interpretation of session types to describe a theory of logical relations for session types, allowing one to study properties like termination of well-typed interactions, and behavioural characterisations of session-typed isomorphisms as linear logic equivalences.

Type and effect systems have been used to study structured communications. In [Gordon & Jeffrey 2003], the π- calculus is extended with labelled assertions describing progress in their communication steps. Assertions have complementary operations, and one can ensure that the communication is safe if all specified assertions have their correspondent begin-end operations present in the run of a protocol. In [Bonelli *et al.* 2005], the theory of session types with corresponding assertions is studied, providing stronger guarantees for session types, in the sense that correspondence assertions allow one to keep track of the changes to the data transmitted over sessions, and the way data is propagated across multiple parties.

Relations between types and logics can also give more information about the nature of structured communications. In [Bocchi *et al.* 2010], authors proposed the integration of typed-based signatures with logical predicates as a method to guarantee

finer grained properties about the information in transit in structured interactions. The proposed a methodology (*Design by contract*), constitutes an extension of multiparty session types [Honda *et al.* 2008] with global assertions, describing global constraints on processes' interactions in terms of predicate logic formulae. In this way, types not only describe causal relations between the inter-process communications, but they also fulfil constraints regarding the values in transit.

In [Berger *et al.* 2008], a proof systems characterising May/Must testing pre-orders and bisimilarities over typed π-calculus processes is presented. The connection between types and logics in such system comes in handy to restrict the shape of the processes one might be interested in, allowing us to consider such work as a suitable proof system for calculi describing the communication of end points.

In the context of security, the work on F7 [Gordon & Fournet 2009] has explored the integration of dependent and refinement types in a suite of functional programming languages, with the aim of statically checking assertions about data and state, in order to enforce security policies.

It is important to relate this research to related approaches involving types and timed constraints and exceptional behaviour. With respect to exceptional behaviour, [Carbone *et al.* 2008] proposed a language for *interactional exceptions*, in which exceptions in a protocol generate coordinated actions between all peers involved. Associated type systems ensure communication safety and termination among protocols with normal and compensating executions. In [Capecchi *et al.* 2010], the language is enriched further with multiparty session and global escape primitives, allowing nested exceptions to occur at any point in an orchestration. As for choreographies, [Carbone 2008] introduced an extension of a language of choreographies with try/catch blocks, guaranteeing that embedded compensating parts in a choreography are not arbitrarily killed as a result of an abort signal. With respect to timing behaviour, [Berger & Yoshida 2007] proposes typing analysis techniques for a variant of the asynchronous π-calculus with locations and time windows. A linear/affine type discipline presents a way to integrate time and linearity conditions in the analysis of interactions: by typing timed processes, one is able to provide further guarantees about the liveness conditions of the systems in consideration, allowing us to consider cases where messages are lost or participants involved do not reply.

Part of the research presented in Chapter 7 is based on the conversation calculus. Types in the conversation calculus are flexible structures that allow us to describe global and local views of the systems in a uniform manner [Caires & Vieira 2010]. Compared to session types, the type discipline for the conversation calculus is a generalisation of the theory of end-point projections where global and end-point types are expressed at the same level. Moreover, global types do not correspond to fixed participants but can be instantiated at runtime, allowing for scenarios where dynamic reconfigurations of multiparty sessions (delegation for instance) is captured by the types.

1.4.5 Behavioural Contracts

Parallel to session types, a theory of contracts has emerged in recent years. Contracts are behavioural descriptions of web services [Castagna *et al.* 2008, Bravetti & Zavattaro 2008], and formalise notions of compatibility between services and safe replacement of services. In particular, a contract can be seen as the description of the external, observable behaviour of a service, or as behavioural types of processes that do not represent the internal structure nor parallelism. Contracts are endowed with subcontract relations, describing pre-orders between implementations of services and the general specifications given in the contracts, in a way similar to semantic subtyping in [Castagna *et al.* 2005]. In this way, services can be exchanged as long as they are both subcontracts of the same contract specification. Authors in [Laneve & Padovani 2008] explore the relations between contracts and session types, and present a series of encodings from session types to contracts and vice versa. Related to such encodings, this work shows that while the encoding from session types to contracts is straightforward and almost homomorphic with respect to the operations in contracts, the converse relation (from contract specifications to session types) is far more complex, and manifests an exponential blow up of the encoded contract in session types. That is due to the level of abstraction used in the description of contracts, which is suggested to be higher than the one in session types. In [Castagna & Padovani 2009] the correspondence between contracts and session types is extended further, by adding contract specifications with explicit channel names that allows one to describe properties about message exchanges, delegation, and dynamic reconfiguration in terms of contracts. A logic for contracts has been proposed in [Bartoletti & Zunino 2010], where elements of the π-calculus, concurrent constraint programming and the fusion calculus [Victor & Parrow 1998] are combined in a language describing contract relations between multiple participants. Evolution in such calculus depends on the satisfaction of constraints regarding the contract specification given by each participant. In this way, the calculus describe satisfaction of "contractual specifications" much in the way of Service-Level Agreements [Buscemi & Montanari 2007]. Although this research gives a different outlook on contracts in service specifications, they can be seen as a declarative underspecified specification of which behavioural contracts are an implementation.

1.4.6 Modelling Standards

We want to relate this work to the modelling standards used in the industry. The Business Process Modelling Notation (BPMN) [Object Management Group 2011] is an effort towards a unified modelling language for business processes, and has emerged as the de-facto modelling notation in industry. BPMN was designed to provide a graphical notation for XML-based business process languages, such as WS-BPEL [Andrews *et al.* 2003]. With it, business analysts can take advantage of the use of BPMN since they can exploit facilities for generating executable WS-BPEL code from BPMN graphical models [Ouyang *et al.* 2006]. However, the specification of

BPMN as a language is rather informal and leaves ambiguity about its semantics [Recker & Mendling 2006, Dijkman *et al.* 2007b, Dijkman *et al.* 2007a], making BPMN models unsuitable for formal verification. Only recently, encodings of BPMN into different process calculi such as CSP [Hoare 1983] and COWS have been provided as an way to guarantee a formal account of BPMN [Wong 2010, Prandi *et al.* 2008]. Finally, Message Sequence Charts (MSCs) [Harel & Thiagarajan 2004] appeared as a technique for describing patterns of interaction between participants in distributed systems, and it has been used in the industry (e.g., telecommunication) to describe the global behaviour from a global point of view. Albeit theoretical, some efforts suggest that global descriptions in MSCs can have an equivalent of the end-point projection. A MSC is realisable if the interfaces generated from the it guarantee causality relations between inputs and outputs [Broy 2007].

1.5 Organisation and Structure

Chapters 3–8 of this dissertation each consists of a single paper. Each chapter can be read independently, and each concludes with a summary of its content and a discussion about related work. Albeit independent, some relations on the reading order of each chapter are presented in Section 1.5.2. The contribution of each chapter is described as follows:

Chapter 2 [Technical Background] This chapter provides the theoretical background for the thesis. We introduce fundamental concepts on process calculi for Communication -Centred Programming, including the π-calculus, Universal Temporal Concurrent Constraint Programming (utcc), Languages for structured communication and the Conversation Calculus. Moreover, we recall concepts of session types, and behavioural relations between processes.

Chapter 3 [A Unified Framework for Structured Communications] This chapter is a first endeavour towards a unified framework for the declarative analysis of structured communications. Starting from Universal Temporal Concurrent Constraint Programming (utcc), we provide relations between temporal logics and orchestration languages. This is realised by using utcc to give a declarative interpretation to the language of orchestrations of [Honda *et al.* 1998]. This way, services can be analysed in a declarative framework where time is defined explicitly, and their behaviour compared to formulae in LTL. Moreover, the selected language is prone to timed extensions: an orchestration language can be benefited from the inclusion of timed information on the duration of sessions, declarative preconditions within session establishment constructs, and session abortion primitives.

Chapter 4 [Types for Security and Mobility in Universal CCP] In this chapter, we dive deep into aspects regarding mobile and secure communications in Concurrent Constraint languages, which are prime concerns in Communication-Centred Programming. As described before, Universal Temporal Concurrent

Constraint Programming (utcc) is a recent addition to the family of Concurrent Constraint languages that introduces the possibility of universally quantify over predicates in the constraint store. We present how utcc can capture mobility and access control of information flow. To do so, we proposed utcc_s , an extension of utcc with a type system for constraints used as patterns in process abstractions, which essentially allows us to distinguish between universally abstractable information and secure (non-leakable information) in predicates. We also proposed a novel notion of abstraction under local knowledge, which gives a general way to model that a process (principal) knows a key and can use it to decrypt a message encrypted with this key without revealing the key.

Chapter 5 [A logic for Choreographies] We explore logical reasoning for the global calculus, a coordination model based on the notion of choreography, with the aim to provide a methodology for specification and verification of structured communications. Starting with an extension of Hennessy-Milner logic, we present the global logic ($\mathcal{GL}$), a modal logic describing possible interactions among participants in a choreography. We illustrate its use by giving examples of properties of service specifications. Finally, we show that, despite $\mathcal{GL}$ is being undecidable, there is a significant decidable fragment which we provide with a sound and complete proof system for checking validity of formulae.

Chapter 6 [Modal Logics for Structured Communications] We build upon the results presented in Chapter 5 to develop a full a framework integrating imperative and declarative views for structured communications. Starting from languages for the specification of services, we provide a modal logic characterisation of the interactions occurring in a system, both from a global standpoint and from the views of each participant. The framework copes with two aims: exhibiting logical guarantees about the presence of an interaction, and model generation from logical specifications.

Chapter 7 [Time and Exceptional Behaviour in Multiparty Structured Interactions] The Conversation Calculus (CC) is an extension of the π-calculus, intended as a model of multiparty interactions. The CC is built upon the notion of *conversation*—a possibly distributed medium in which participants may communicate. We study the interplay of time and exceptional behaviour for models of structured communications based on conversations. We propose C3, a *timed* variant of the CC in which conversations feature both standard and exceptional behaviour. The exceptional behaviour may be triggered either by the passing of time (a timeout) or by an explicit signal for conversation abortion. We argue that the combination of time and exceptional behaviour greatly enhances the significance and level of detail of specifications of structured communications.

Chapter 8 [Towards Refinement Relations in Open Specifications] Here we present initial ideas on *Open Specifications*. An open specification describes the behaviour of a system in terms of activities that can be refined. We comment

on different notions of refinement for open specifications. First, we propose a new denotational behavioural model called open mixed trees which generalises standard model of labelled trees (where labels are marked as negative, positive or both) by annotating each state with a set of so-called open actions and a flag indicating if termination is allowed in the state or not. The definition of refinement is then a generalisation of covariant-contravariant simulation that also takes account of termination and allows intermediate open parts of the specification. Second, we explore transition systems with responses for the specification of open systems. A transition system with responses is a new generalisation of modal transition systems that allows for natural of deadlock freedom and liveness for infinite computations. Here we present a definition of refinement that fits transition systems with responses.

1.5.1 Publication list

This thesis compiles the results published in the following articles:

[Hildebrandt & López 2009] Thomas Hildebrandt and Hugo A. López. *Types for Secure Pattern Matching with Local Knowledge in Universal Concurrent Constraint Programming* . In 25th International Conference on Logic Programming (ICLP), volume 5649 of *Lecture Notes in Computer Science*, pages 417–431. Springer, Berlin Heidelberg, 2009

[López *et al.* 2010] Hugo A. López, Carlos Olarte and Jorge A. Pérez. *Towards a Unified Framework for Declarative Structured Communications*. In 2nd Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), volume 17 of *EPTCS*, pages 1–15, 2010

[López 2010] Hugo A. López. *Models for Trustworthy Service and Process Oriented Systems*. In 26th International Conference on Logic Programming (ICLP), volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 270–276, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

[Carbone *et al.* 2010] Marco Carbone, Thomas Hildebrandt, Davide Grohmann and Hugo A. López. *A logic for Choreographies*. In 3rd Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), 2010

[López & Pérez 2011] Hugo A. López and Jorge A. Pérez. *Timed, Compensable Conversations*. In 4th Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), 2011

[López & Pérez 2012] Hugo A. López and Jorge A. Pérez. *Time and Exceptional Behavior in Multiparty Structured Communications*. In Marco Carbone and Jean-Marc Petit, editors, Web Services and Formal Methods (WS-FM), volume (To appear) of *Lecture Notes in Computer Science*. Springer, 2012

[Carbone *et al.* 2011] Marco Carbone, Thomas Hildebrandt and Hugo A. López. *Open Mixed Refinement*. In Nordic Workshop of Programming Theory (NWPT), Västerås, Sweden, November 2011

[Carbone *et al.* 2012] Marco Carbone, Thomas T. Hildebrandt, Hugo A. López, Gian Perrone and Andrzej Wasowski. *Refinement for Transition Systems with Responses*. In 4th International Workshop on Foundation of Interface Technologies (FIT), 2012. Accepted for publication

The following paper is not part of this dissertation, but it contributed by providing some details on the case studies here explored.

[López *et al.* 2009] Hugo A. López, Fabio Massacci and Nicola Zannone. *Goal-Equivalent Secure Business Process Re-engineering*. In E. Di Nitto and M. Ripeanu, editors, Service-Oriented Computing - ICSOC 2007 Workshops, volume 4907 of *LNCS*, pages 212—223, Berlin, Heidelberg, January 2009. Springer - Verlag

1.5.2 Document Structure

> *"All disorder had meaning if it seemed to come out of itself, perhaps through madness one could arrive at that reason which is not the reason whose weakness is madness. «To go from disorder to order», thought Oliveira."* Julio Cortázar, *Hopscotch*, Chapter 18.

As described at the beginning of this section, each chapter can be considered as self-contained document per se, and therefore the reader is free to choose a convenient reading order according to his/her preferences. To provide some hints, some clear connections between the chapters will be outlined below.

Figure 1.4 describes the links between each chapter. The ordering between the first two chapters (Introduction and Preliminaries) is pretty obvious. After Chapter 2 one might chose to read any paper between Chapter 3 and Chapter 8. Chapter 9 concludes and discusses possible strands for future work. Some extra links are provided, and can be of help for the lost reader:

- Chapters 3 and 4 are both contributions on Temporal Concurrent Constraint Programming. They share the same language definition and therefore a combined reading is suggested.
- Chapter 5 and Chapter 6 relate to the connection between logics and structured communications, where Chapter 5 presents a logical formalisation for choreographies and Chapter 6 extends such work towards full logical characterisation for the Global and End-Point Calculus. Therefore, a linear reading between both chapters is suggested.
- The notions of time and exceptional behaviour are present in Chapter 3 and Chapter 7. In fact, one can consider the work in Chapter 3 as the main source

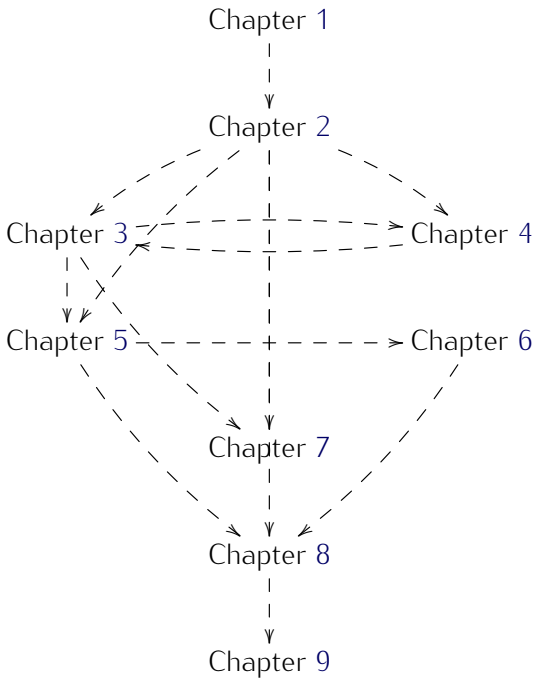


Figure 1.4: The structure of this thesis

of inspiration for timing analysis of structured communications, that will be explored later in Chapter 7. Therefore, an ascending order between these two chapters is suggested.

- The idea of partial specifications in structured communications that have been put forward in Chapter 5 and Chapter 6 is later explored in terms of behavioural (refinement) relations between processes, which appears in chapter 8. A linear reading between them might be useful.

Chapter 2

Technical Background

Contents

2.1 Process Calculi

Let us illustrate the interplay of the above issues by introducing one of the most representative process calculus for mobility.

2.1.1 A Process Calculus for Mobile Systems

The π-calculus [Milner 1999, Sangiorgi & Walker 2001], was proposed by Milner, Parrow and Walker in the early 90's for the analysis of mobile, distributed systems. The ability of representing *link mobility* is one of the main advances of the π-calculus with respect CCS (Calculus for Communicating Systems) [Milner 1995], its immediate predecessor. In the π-calculus, the description of mobile systems and their interactions is based on the notion of *name*. In principle, a process (an abstraction of a mobile agent) should be capable of evolving in many different ways, but always maintaining its identity during the whole computation. In addition, a process should be capable of identifying other related processes. In the π-calculus a name also denotes a *communication channel*, in such a way that communication among two processes is possible provided that they share the same channel. As a consequence,

in the π-calculus a name abstracts the identity of processes in an interaction by considering the communication channel each process is related to.

In the π-calculus, process capabilities are abstracted as *atomic actions*. They come in two main flavours:

- $x(z)$, representing the *reception* (or reading) of the datum z on the channel x. z is then ready for any subsequent computations.
- $\overline{x}\langle d\rangle$, denoting the *transmission* of a datum d over the channel x.

Actions (denoted by α) are used in the context of *processes* that are constructed by the following syntax:

$$\begin{aligned} P, Q, \ldots \ ::= \ & \mathbf{0} \\ & \mid \sum_{i \in I} \alpha_i.P_i \\ & \mid P \mid Q \\ & \mid !P \\ & \mid (\nu x)\ P \end{aligned}$$

Some intuitions underlying the behaviour of these processes follow.

- Process $\mathbf{0}$ represents the process that does *nothing*. It is meant to be the basis of more complex processes.
- The *interaction* of processes P and Q is represented by their *parallel composition* $P \mid Q$. In addition to the individual actions of each process, their communication is possible, provided that they *synchronise* on a channel, as illustrated in the following example.

  $$R = x(y).\overline{y}\langle z\rangle.\mathbf{0} \mid \overline{x}\langle w\rangle.\ \mathbf{0}$$

  Here, R represents the interaction of two processes sharing a channel x. The transmission of w through x is complemented by its reception, which involves recognising w as y. This is regarded as an atomic computational step. Afterwards, a datum z is sent, using the received name w as communication channel. Notice that in the context of R, there is no partner for w in its attempt of transmitting z.
- $\sum_{i \in I} \alpha_i.P_i$, usually known as a *summation* process, represents a choice on the involved P_i's, depending on the capabilities represented by each α_i. Only when any such processes is ready to interact with another one, a *choice* among all the possible interaction options takes place. For instance, in the process

  $$(x(y).\ \overline{z}\langle y\rangle.\ \mathbf{0} + z(y).\ \mathbf{0} + x(w).\ w(z).\ \mathbf{0}) \mid \overline{x}\langle r\rangle.\ \mathbf{0}$$

the first and third components of the sum are ready to interact with $\overline{x}\langle r\rangle.\ \mathbf{0}$. Depending on the choice, different resulting processes are possible. For instance, if the third component is selected, the resulting interaction would lead to the process $r(z).\ \mathbf{0}$.

- Process $!P$ represents the *infinite execution* (or *replication*) of process P. There will be an infinite number of copies of P executing: $!P = P \mid P \mid P \mid \ldots$.
- Process $(\nu x)\ P$ is meant to describe *restricted* names. Name x is said to be *local* to P and is only visible to it. We often write $(\nu \tilde{x})\ P$ to stand for the process $(\nu x_1)\ (\nu x_2)\ \ldots (\nu x_n)\ P$. A disciplined use of restricted names is crucial in delimiting communication.

The π-calculus is thus a language based in a few simple, yet powerful, abstractions. In addition to the above-mentioned abstraction of name as communication channels that can be transmitted, in the π-calculus the behaviour of mobile systems is reduced to a few representative phenomena: synchronisation on shared channels, infinite behaviour and restricted communication. The compositional nature of the calculus is elegantly defined by the parallel composition operator, which is the basis for representing interactions among processes and the construction of models.

Meaning of Processes Endowing process terms with a formal meaning is crucial in order to analyse process behaviour. A process language can have several semantic interpretations. In fact, the combination of two or more approaches is a common practice, since for instance, an approach can be more appropriate for intuitive understanding of processes whereas others can be more suitable for mathematical proofs. This is usually the case of Operational Semantics and Denotational/Algebraic ones. The use of several semantics motivates a legitimate question, that of determining whether different semantics are equivalent to each other. Lets illustrate the use of a semantics with the introduction of an operational semantics for the π-calculus.

Operational Semantics An operational semantics interprets a process term by using *transitions* that define computational steps [Plotkin 1981]. A common practice is to capture the state of the system by means of *configurations*: succinct structures including a process term and other relevant information to describe the state of the system (for instance, one could include the state of the variables in the configuration).

First we define transition systems as in [Joyal *et al.* 1993].

Definition 2.1.1 (Transition Systems [Joyal *et al.* 1993]). *A transition system T is a quadruple $(S, I, \longrightarrow, \mathrm{Act})$ where S is a set of states, $I \subseteq S$ is the set of initial states, $\longrightarrow \subseteq S \times \mathrm{Act} \times S$ is the transition relation and* Act *is a set of labels.*

Sometimes it is useful to consider transition systems without initial states, as introduced in [Keller 1976]. These special kind or transition systems are referred as Labelled Transition Systems (LTS), and can be thought of as an automaton without a start state or accepting states. Transitions are usually *labelled* by the actions that

$$\frac{}{x(y).\ P \xrightarrow{x(y)} P}\ \mathrm{P_{input}} \qquad \frac{}{\overline{x}\langle z\rangle.\ P \xrightarrow{\overline{x}\langle z\rangle} P}\ \mathrm{P_{output}}$$

$$\frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}\langle z\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{z/y\}}\ \mathrm{P_{synch}} \qquad \frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}\ \mathrm{P_{rep}}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \mathrm{P_{sum}} \qquad \frac{P \xrightarrow{\alpha} P' \quad x \notin n(\alpha)}{(\nu x)\ P \xrightarrow{\alpha} P'}\ \mathrm{P_{res}}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{P_{par}} \qquad \frac{P \xrightarrow{\overline{x}\langle y\rangle} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)\ (P' \mid Q')}\ \mathrm{P_{close}}$$

$$\frac{P \xrightarrow{\overline{x}\langle y\rangle} P'}{(\nu y)\ P \xrightarrow{\overline{x}(w)} P'[w/y]}\ y \neq x, w \notin fn((\nu y)\ P')\ \mathrm{P_{open}}$$

Figure 2.1: Operational semantics of the (late) π-calculus with no matching, symmetric rules for $+$ and $\mid$ are elided

originate evolution between configurations. This is commonly denoted as $P \xrightarrow{a} Q$, meaning that process P performs action a and then behaves as process Q.

Definition 2.1.2 (Labelled Transition Systems). *A labelled transition system T is a triple $(S, \mathrm{Act}, \xrightarrow{a})$ where S is a set of states,* Act *is a set of transition labels, and $\xrightarrow{a} \subseteq S \times a \times S$ is the labelled transition relation such that $a \in \mathrm{Act}$.*

Operational semantics are then defined by a set of *(transition) rules* that formally define the features of the relation $\xrightarrow{a}$. The set of transition rules that constitute the operational behaviour of a calculus is also known as its *LTS*.

As an example, consider the rule that formalises the communication of interacting processes in the π-calculus:

$$x(y).P \mid \overline{x}\langle z\rangle.\ Q \xrightarrow{\tau} P\{z/y\} \mid Q.$$

In this (labelled) rule, $P\{z/y\}$ denotes the syntactic replacement of all occurrences of the name y with the name z in the context of process P. We use $fn(\alpha)$ and $bn(\alpha)$ to denote the set of free and bound names in action (α). The set of names in α is defined as $n(\alpha) = fn(\alpha) \cup bn(\alpha)$. The set of transition rules for the late π-calculus with no matching is presented in Figure 2.1.

We will use variations of operational semantics in Chapters 3,4, 5,7.

Reduction Semantics The LTS describes not only the autonomous evolution of processes, captured by τ transitions, but also the interactions with the external en-

vironment. When we focus on closed systems, not subject to interaction with the environment, we are only interested on autonomous behaviour (τ transitions), which we capture by a notion of reduction.

Definition 2.1.3 (Reduction Relation $\rightarrow$). *The relation of reduction between processes, noted $P \rightarrow Q$, is defined as $P \xrightarrow{\tau} Q$. Also, we denote by $\rightarrow^*$ the reflexive transitive closure of the reduction relation.*

Using the behavioural descriptions captured by the LTS we define a behavioural semantics, which then precisely characterises when two systems have the same behaviour, and thus correspond to the same specification from a behavioural point of view.

We will use a variation of the reduction semantics of the π-calculus in Chapter 3.

Assertion Semantics An assertion (or logical) semantics gives meaning of process interactions in terms of a language with well-defined and intuitively understanding semantics. Usually, the language selected for an assertion semantics has a logical flavour, and allows a user to describe properties about the evolution of a process specification (Model checking).

In what follows, we hall present a property language that was introduced in process theory by [Hennessy & Milner 1985]. We are going to refer to this language as the Hennessy-Milner Logic (HML).

Definition 2.1.4 (Hennessy-Milner Logic). *The set $\mathcal{M}$ of Henessy-Milner formulae over a set of actions* Act *is given by the following abstract syntax:*

$$
\begin{aligned}
F, G ::= & \mathsf{tt} \\
& \mid \mathsf{ff} \\
& \mid F \wedge G \\
& \mid F \vee G \\
& \mid \langle a \rangle F \\
& \mid [a] F
\end{aligned}
$$

where $a \in$ Act and tt, ff *denote logical true and false, respectively.*

The satisfaction relation $\models$ relates processes to HML formulae by structural induction on formulae such that:

$$
\begin{aligned}
P &\models \mathsf{tt} \text{ for each } P \\
P &\models \mathsf{ff} \text{ for no } P \\
P &\models \phi \wedge \chi \text{ iff } P \models \phi \text{ and } P \models \chi \\
P &\models \phi \vee \chi \text{ iff } P \models \phi \text{ or } P \models \chi \\
P &\models \langle a \rangle \phi \text{ iff } P \xrightarrow{a} P' \text{ and } P' \models \phi \\
P &\models [a] \phi \text{ iff } P \xrightarrow{a} P' \text{ and } P' \models \phi \text{ for all } P'
\end{aligned}
$$

We will explore the uses of process characterisations in terms of variants of the HML logic and Linear Temporal Logic (LTL) further in Chapters 3,4 and Chapter 5.

2.1.2 (Temporal) Concurrent Constraint Programming

This section provides the interested reader the main concepts of Concurrent Constraint Programming (CCP), Temporal Concurrent Constraint Programming (tcc) and its universal extension (utcc), following the presentation of [Olarte & Valencia 2008a].

Concurrent Constraint Programming (CCP) was first introduced by Vijay Saraswat in [Saraswat 1993] as a rich family of programming languages where (partial) information plays a fundamental in the computation and control of concurrent programs. In CCP-based calculi all the (partial) information is *monotonically* accumulated in a so-called *store*. The store keeps the knowledge about the system in terms of *constraints*, or statements defining the possible values a variable can take (e.g., $x+y \geq 42$). It also defines an *entailment* relation "$\Vdash$" specifying interdependencies among constraints. Intuitively, $c \Vdash d$ means that the information in d can be deduced from that in c (as in, e.g., $x \geq 42 \Vdash x \geq 0$). Concurrent agents (i.e., processes) that are part of the system interact with each other using the store as a shared communication medium. They have two basic capabilities over the store, represented by *tell* and *ask* operations. While the former *adds* a piece of information about the system, the latter *queries* the store to determine if some piece of information can be inferred from its current content. Tell operations can act concurrently refining the information in the store while asks can serve as a general synchronisation mechanism, that will be blocked if there is not enough information into the store to answer its query.

A fundamental notion in CCP-based calculi is that of a *constraint system*. Basically, a constraint system provides a signature from which syntactically denotable objects in the language called *constraints* can be constructed, and an entailment relation ($\Vdash$) specifying interdependencies among such constraints. More precisely,

Definition 2.1.5 (Constraint System). *A constraint system is a pair* $CS = (\Sigma, \Delta)$ *where* Σ *is a signature of function (*$\mathbb{F}$*) and predicate (*$\mathbb{P}$*) symbols, and* Δ *is a decidable theory over* Σ *(i.e., a decidable set of sentences over* Σ *with at least one model). The underlying language* $\mathcal{L}$ *of* (Σ, Δ) *contains the symbols* $\neg, \wedge, \Rightarrow, \exists$ *denoting logical negation, conjunction, implication, existential quantification.* Constants*, such as* tt *and* ff *denote the usual always true and always false values, respectively.* Constraints*, denoted by* $c, d, \ldots$ *are first-order formulae over* $\mathcal{L}$*. We say that* c entails d *in* Δ*, written* $c \Vdash_\Delta d$ *(or just* $c \Vdash d$ *when no confusion arises), if* $c \Rightarrow d$ *is true in all models of* Δ*. For operational reasons we shall require* $\Vdash$ *to be decidable.*

Timed concurrent constraint programming (tcc) [Saraswat *et al.* 1994, de Boer *et al.* 2000] extends CCP for modelling reactive systems. In tcc, time is conceptually divided into *time units* (or discrete *time intervals*). In a particular time unit, a tcc process P gets an input (i.e. a constraint) c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines also a residual

process Q which is then executed in the next time unit. Here it is where one of the most important differences between CCP and tcc resides, as whilst the refinement of c during the execution of P at interval i is monotonic, d is not necessarily a refinement of c (that is, constraints can be forgotten).

Definition 2.1.6 (tcc **process syntax).** *Processes $P, Q, \ldots \in Proc$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by the following syntax.*

$$
\begin{aligned}
P, Q \ldots ::= & \textbf{skip} \\
& \mid \textbf{tell}\,(\mathrm{c}) \\
& \mid \textbf{when}\ \mathrm{c}\ \textbf{do}\ P \\
& \mid P \,\big|\big|\, Q \\
& \mid (\textbf{local}\ \vec{x}; c) P \\
& \mid \textbf{next}\,(P) \\
& \mid \textbf{unless}\ c\ \textbf{next}\,(P) \\
& \mid\ !P
\end{aligned}
$$

Intuitively, the process **skip** does nothing, **tell** (c) adds a new constraint c into the store, while **when** c **do** P asks if c is present into the store in order to execute P. A process (**local** $\vec{x}$; c) P *binds* the variables $\vec{x}$ in P by declaring them private to P under a constraint c. If c = tt, we write (**local** $\vec{x}$) P instead of (**local** $\vec{x}$; tt) P. The operators associated with time allow the process to go one time unit in the future (**next** (P)) or to define time-outs: if at the current time unit it is not possible to entail the constraint c then the process **unless** c **next** P will execute P at the next time unit. We will often use $\textbf{next}^n\,(P)$ as a shorter version of **next** (**next** (. . . **next** (P)))) n-times. Finally, $P \,||\, Q$ denotes the usual parallel execution and $!P$ denotes timed replication; that is, $!P = P \,||\, \textbf{next}\,(!P)$ executes P at the current time and replicates its behaviour over the next time period.

utcc [Olarte & Valencia 2008a] is an extension of the tcc calculus with a general *ask* defining a model of synchronisation. While in tcc an ask **when** c **do** P is blocked if there is not enough information to entail c from the store, utcc inspires its synchronisation mechanism on the notion of abstraction in functional programming languages.

Definition 2.1.7 (utcc **processes).** *The syntax of* utcc *processes result from replacing process* **when** c **do** P *in Definition 2.1.6 with the abstraction process* $(\lambda\, \vec{x}; \mathrm{c})\ P$.

$(\lambda\, \vec{x}; \mathrm{c})\ P$ can be seen as the dual version of (**local** $\vec{x}$; c) P in which the variables are *abstracted* with respect to the constraint c and the process P. A process $Q = (\lambda\, \vec{x}; \mathrm{c})\ P$ binds the variables $\vec{x}$ in P and c. It executes $P[\vec{t}/\vec{x}]$ for every term $\vec{t}$ s.t. the current store entails an admissible substitution over $\mathrm{c}[\tilde{\mathrm{t}}/\tilde{\mathrm{x}}]$. The substitution $[\vec{t}/\vec{x}]$ is admissible if $|\vec{x}| = |\vec{t}|$ and no x_i in $\vec{x}$ occurs in $\vec{t}$. Furthermore, Q evolves into **skip** at the end of

$$\mathrm{R_T}\ \frac{}{\langle \mathbf{tell}\,(\mathrm{d}),\mathrm{c}\rangle \longrightarrow \langle \mathbf{skip},\mathrm{c}\wedge\mathrm{d}\rangle} \qquad \mathrm{R_A}\ \frac{d \Vdash c[\vec{t}/\vec{x}] \quad [\vec{t}/\vec{x}] \text{ is admissible}}{\langle(\lambda\ \vec{x};\mathrm{c})\ P,\mathrm{d}\rangle \longrightarrow \langle P[\vec{t}/\vec{x}]\,||(\lambda\ \vec{x};\mathrm{c}\wedge(\tilde{\mathrm{x}}\neq\tilde{\mathrm{t}}))\ P,\mathrm{d}\rangle}$$

$$\mathrm{R_P}\ \frac{\langle P,\mathrm{c}\rangle \longrightarrow \langle P',\mathrm{c}'\rangle}{\langle P\,||\,Q,\mathrm{c}\rangle \longrightarrow \langle P'\,||\,Q,\mathrm{c}'\rangle} \qquad \mathrm{R_U}\ \frac{d \Vdash c}{\langle \mathbf{unless}\,\mathrm{c}\,\mathbf{next}\,P,\mathrm{d}\rangle \longrightarrow \langle \mathbf{skip},\mathrm{d}\rangle}$$

$$\mathrm{R_R}\ \frac{}{\langle !P,\mathrm{c}\rangle \longrightarrow \langle P\,||\,\mathbf{next}\,(!P),\mathrm{c}\rangle} \qquad \mathrm{R_L}\ \frac{\langle P,(\exists\tilde{\mathrm{x}}\mathrm{d})\wedge\mathrm{c}\rangle \longrightarrow \langle P',(\exists\tilde{\mathrm{x}}\mathrm{d})\wedge\mathrm{c}'\rangle}{\langle(\mathbf{local}\ \vec{x};\mathrm{c})\ P,\mathrm{d}\rangle \longrightarrow \langle(\mathbf{local}\ \vec{x};\mathrm{c}')\ P',(\exists\tilde{\mathrm{x}}\mathrm{c}')\wedge\mathrm{d}\rangle}$$

$$\mathrm{R_O}\ \frac{\langle P,\mathrm{c}\rangle \longrightarrow^{*} \langle Q,\mathrm{d}\rangle \not\longrightarrow}{P \overset{(c,d)}{\Longrightarrow} F(Q)}$$

$$\text{Where } F(Q) = \begin{cases} \mathbf{skip} & \text{if } Q = \mathbf{skip} \\ F(Q_1)\,||\,F(Q_2) & \text{if } Q = Q_1\,||\,Q_2 \\ R & \text{if } Q = \mathbf{next}\,(R) \\ \mathbf{skip} & \text{if } Q = (\lambda\ \vec{x};\mathrm{c})\ R \\ (\mathbf{local}\ \vec{x})\ F(R) & \text{if } Q = (\mathbf{local}\ \vec{x};\mathrm{c})\ R \\ R & \text{if } Q = \mathbf{unless}\,\mathrm{c}\,\mathbf{next}\,R \end{cases}$$

Figure 2.2: Transition System for utcc: Internal and Observable transitions

the time unit, i.e., abstractions are not persistent when passing from one time unit to the next one.

The operational semantics provides the intuitions on how utcc processes interact. In principle, a configuration is represented by the tuple $\langle P,\mathrm{c}\rangle$ where P denotes a set of processes and c a constraint store. P can evolve to a further process P' during an *internal transition* ($\longrightarrow$) where the constraint store c is monotonically refined, or can execute an *observable transition* ($\Longrightarrow$), producing the result of the future function of P and the constraint store d. The set of operational rules is presented in Figure 2.2, where $\langle P,\mathrm{c}\rangle$ denotes a configuration, and $F(Q)$ denotes the *future function of process* Q.

Intuitively, the operational rules of utcc behaves almost in the same way as its counterpart in tcc, excepting by the general treatment of asks in utcc. Here we will describe the operational consequence of this change, we refer to [Olarte & Valencia 2008a] for further details on the operational semantics. Rule $\mathrm{R_A}$ describes the behaviour of the abstraction $(\lambda\ \vec{x};\mathrm{c})\ P$: a configuration here considers two stores, being c and d *local* and *global* stores respectively. If d entails $\mathrm{c}[\tilde{\mathrm{t}}/\tilde{\mathrm{x}}]$ then $P[\vec{t}/\vec{x}]$

is executed. Moreover, the abstraction persists in time, allowing any other process to match with $\vec{x}$ in P while no other replacements of $\vec{x}$ with $\vec{t}$ will occur, as d is augmented with a constraint disallowing this.

The notion of *local information* can be evidenced in R_L, considering a process $P = (\textbf{local}\ \vec{x}; c)\ Q$, we have to consider: (i) that the information about $\vec{x}$ locally for P subsumes any other information present for the same set of variables in the global store; therefore, $\vec{x}$ is hidden by the use of an existential quantifier over $\tilde{x}$ in d. (ii) that the information about $\vec{x}$ that P can produce after the reduction is still local, so we hide it by existentially quantifying $\vec{x}$ in c' before publishing it to the global store. After the reduction, c' will be the new local store of the evolution of internal processes.

Finally, observable behaviour is described by R_o: after having used the internal transitions in a process P to evolve to a process Q with a quiescent-point (in which no more information can be added/inferred), the reduction will continue by executing the future function of Q with the resulting constraint store.

Assertion Semantics utcc provides a number of reasoning techniques: First, for a significant fragment of the calculus, the input-output behaviour of a process P can be retrieved from the set of fixed points of its associated closure operator [Olarte & Valencia 2008b]. Second, utcc processes can be characterised as First-order Linear-time Temporal Logic (FLTL) formulae [Manna & Pnueli 1992] (See section 2.2.1). This declarative view of the processes allows for the use of the well-established verification techniques from FLTL to reason about utcc processes.

Definition 2.1.8 (Output Behaviour). *Let $s = c_1.c_2....c_n$ be a sequence of constraints. If $P = P_1 \xrightarrow{(\mathrm{tt},c_1)} P_2 \xrightarrow{(\mathrm{tt},c_2)} \dots P_n \xrightarrow{(\mathrm{tt},c_n)} P_{n+1} \equiv_u Q$ we shall write $P \xrightarrow{s}{}^{*} Q$. If $s = c_1.c_2.c_3...$ is an infinite sequence, we omit Q in $P \xrightarrow{s}{}^{*} Q$. The* output behaviour *of P is defined as $o(P) = \{s \mid P \xrightarrow{s}{}^{*}\}$. If $o(P) = o(Q)$ we shall write $P \sim^o Q$. Furthermore, if $P \xrightarrow{s} Q$ and s is unimportant we simply write $P \longrightarrow^{*} Q$.*

Definition 2.1.9. *Let* TL[·] *a map from* utcc *processes to FLTL formulae given by:*

$$
\begin{array}{lcllcl}
\mathrm{TL}[\textbf{skip}] & = & \mathrm{tt} & \mathrm{TL}[\textbf{tell}(c)] & = & c \\
\mathrm{TL}[P \,||\, Q] & = & \mathrm{TL}[P] \wedge \mathrm{TL}[Q] & \mathrm{TL}[(\lambda\ \vec{y}; c)\ P] & = & \forall \vec{y}(c \Rightarrow \mathrm{TL}[P]) \\
\mathrm{TL}[(\textbf{local}\ \vec{x}; c)\ P] & = & \exists \vec{x}(c \wedge \mathrm{TL}[P]) & \mathrm{TL}[\textbf{next}\ P] & = & \circ \mathrm{TL}[P] \\
\mathrm{TL}[\textbf{unless}\ c\ \textbf{next}\ P] & = & c \vee \circ \mathrm{TL}[P] & \mathrm{TL}[!\ P] & = & \Box \mathrm{TL}[P]
\end{array}
$$

We use the *eventual* modality $\Diamond F$ as an abbreviation of $\neg\Box\neg F$.

The following theorem relates the operational view of processes with their logic interpretation.

Theorem 2.1.10 (Logic correspondence [Olarte & Valencia 2008a]). *Let* TL[·] *be as in Definition 2.1.9, P a* utcc *process and $s = c_1.c_2.c_3...$ an infinite sequence of*

constraints s.t. $P \stackrel{s}{\Longrightarrow}^{*}$. *For every constraint* d, *it holds that:* $\mathrm{TL}[P] \Vdash \Diamond d$ *iff there exists* $i \geq 1$ *s.t.* $c_i \Vdash d$.

Recall that an observable transition $P \stackrel{(c,c')}{\Longrightarrow} Q$ is obtained from a finite sequence of internal transitions (rule R_{O}). We notice that there exist processes that may produce infinitely many internal transitions and as such, they cannot exhibit an observable transition; an example is $(\lambda\ x; \mathrm{c}(\mathrm{x}))\ \mathbf{tell}(c(x+1))$. The utcc processes considered in this paper are *well-terminated*, i.e., they never produce an infinite number of internal transitions during a time unit. Notice also that in the Theorem 2.1.10 the process P is assumed to be able to output a constraint c_i for all time-unit $i \geq 1$. Therefore, P must be a well-terminated process.

2.1.3 Languages for Structured Communications

Here, we will mention the approach of the Global Calculus and the End Point Calculus

2.1.3.1 A session type language: HVK

We begin by introducing HVK, a language for structured communication proposed in [Honda *et al.* 1998]. We assume the following conventions: *names* are ranged over by $a, b, \ldots$; *channels* are ranged over by k, k'; *variables* are ranged over by $x, y, \ldots$; *constants* (names, integers, booleans) are ranged over by $c, c', \ldots$; *expressions* (including constants) are ranged over by $e, e', \ldots$; *labels* are ranged over by $l, l', \ldots$; *process variables* are ranged over by $X, Y, \ldots$. Finally, $u, u', \ldots$ denote names and channels. We shall use $\vec{x}$ to denote a sequence (tuple) of variables $x_1 \ldots x_n$ of length $n = |\vec{x}|$. Notation $\vec{x}$ will be similarly applied to other syntactic entities. The sets of free names/channels/variables/process variables of P, is defined in the standard way, and are respectively denoted by $\mathbf{fn}(\cdot)$, $fc(\cdot)$, $\mathrm{fv}(\cdot)$, and $fpv(\cdot)$. Processes without free variables or free channels are called *programs*.

Definition 2.1.11 (The HVK language [Honda *et al.* 1998]). *Processes in* HVK *are built from:*

P,Q	$::=$	**request** $a(k)$ **in** P	*Session Request*
	$\mid$	**accept** $a(k)$ **in** P	*Session Acceptance*
	$\mid$	$k![\vec{e}];\ P$	*Data Sending*
	$\mid$	$k?(\vec{x})\ .\ P$	*Data Reception*
	$\mid$	$k \lhd l; P$	*Label Selection*
	$\mid$	$k \rhd \{l_1 : P_1 \,\|\|\, \cdots \,\|\|\, l_n : P_n\}$	*Label Branching*
	$\mid$	**throw** $k[k'];\ P$	*Channel Sending*
	$\mid$	**catch** $k(k')$ **in** P	*Channel Reception*
	$\mid$	**if** e **then** P **else** Q	*Conditional Statement*
	$\mid$	$P \mid Q$	*Parallel Composition*
	$\mid$	**inact**	*Inaction*
	$\mid$	$(\nu u)P$	*Hiding*

$$\textbf{request}\ a(k)\ \textbf{in}\ Q \mid \textbf{accept}\ a(k)\ \textbf{in}\ P \longrightarrow_h (\nu k)(P \mid Q) \quad \text{(Link)}$$

$$(k![\vec{e}]; P) \mid (k?(\vec{x})\,.\,Q) \longrightarrow_h P \mid Q[\vec{c}/\vec{x}] \quad \text{if } e \downarrow \vec{c} \quad \text{(Com)}$$

$$k \lhd l_i; P \mid k \rhd \left\{ l_1 : P_1 \,\middle|\middle|\, \cdots \,\middle|\middle|\, l_n : P_n \right\} \longrightarrow_h P \mid Pi\ (1 \leq i \leq n) \quad \text{(Label)}$$

$$\textbf{throw}\ k[k']; P \mid \textbf{catch}\ k(k')\ \textbf{in}\ Q \longrightarrow_h P \mid Q \quad \text{(Pass)}$$

$$\textbf{if}\ e\ \textbf{then}\ P\ \textbf{else}\ Q \longrightarrow_h P\ (e \downarrow \mathrm{tt}) \quad \text{(If1)}$$

$$\textbf{if}\ e\ \textbf{then}\ P\ \textbf{else}\ Q \longrightarrow_h Q\ (e \downarrow \mathrm{ff}) \quad \text{(If2)}$$

$$\textbf{def}\ D\ \textbf{in}\ (X[\vec{e}\,\vec{k}] \mid Q) \longrightarrow_h \textbf{def}\ D\ \textbf{in}\ (P[\vec{c}/\vec{x}] \mid Q)\ (e \downarrow \vec{c}, X(\vec{x}\vec{k}) = P \in D) \quad \text{(Def)}$$

$$P \longrightarrow_h P' \text{ implies } (\nu u)P \longrightarrow_h (\nu u)P' \quad \text{(Scop)}$$

$$P \longrightarrow_h P' \text{ implies } P \mid Q \longrightarrow_h P' \mid Q \quad \text{(Par)}$$

Figure 2.3: Reduction Semantics of HVK ($\longrightarrow_h$)[Honda *et al.* 1998].

$$\begin{array}{rcll} & \mid & \textbf{def}\ D\ \textbf{in}\ P & \textit{Recursion} \\ & \mid & X[\vec{e}\,\vec{k}] & \textit{Process Variables} \\ D & ::= & X_1(x_1k_1) = P_1\ \textbf{and}\ \cdots\ \textbf{and}\ X_n(x_nk_n) = P_n & \\ & & & \textit{Declaration for Recursion} \end{array}$$

Operational Semantics of HVK. The operational semantics of HVK is given by the reduction relation $\longrightarrow_h$ which is the smallest relation on processes generated by the rules in Figure 2.3.

Sessions The central idea in HVK is the notion of a *session*, i.e., a series of reciprocal interactions in which two interacting parties first establish a private connection via some public channel and then interact through it, possibly interleaved with other sessions. More concretely, an interaction between two parties starts by the creation of a fresh session identifier, that later will be used as a private channel where meaningful interactions take place. Each session is fresh and unique, so each communication activity will be clearly separated from other interactions; thus, sessions serve as an abstraction unit for describing structured communication.

More precisely, sessions are initialised by a process of the form **request** $a(k)$ **in** Q | **accept** $a(k)$ **in** P. In this case, there is a request, on name a, for the initiation of a session and the generation of a fresh channel. This request is matched by an accepting process on a, which generates a new channel k, thus allowing P and Q to communicate each other. This is the intuition behind rule Link. Three kinds of

atomic interactions are available in the language: sending (including name passing), branching, and channel passing (also referred to as delegation). Those actions are described by rules Com, Label, and Pass, respectively. In the case of Com, the expression $\vec{e}$ is sent on the port (session channel) k. Process $k?(\vec{x})\,.\,Q$ then receives such a data and executes $Q[\vec{c}/\vec{x}]$, where $\vec{c}$ is the result of evaluating the expression $\vec{e}$. The case of Pass is similar but considering that in the constructs **throw** $k[k']; P$ and **catch** $k(k')$ **in** Q, only session names can be transmitted. In the case of Label, the process $k \lhd l_i; P$ selects one label and then the corresponding process P_i is executed. The other rules are self-explanatory.

2.1.3.2 A global view of communications: The Global Calculus

The Global Calculus (GC) [Carbone *et al.* 2006, Carbone *et al.* 2007] originates from the Web Service Choreography Description Language (WS-CDL) [Kavantzas *et al.* 2004], a description language for web services developed by W3C. Terms in GC describe choreographies as interactions between participants by means of message exchanges. As in HVK, the description of such interactions is centred on the notion of a *session*. In this section, we present the basic elements of GC and its reduction semantics. Terms in GC describe how how global descriptions evolve, and relate to the type discipline that describes the structured sequence of message exchanges between participants.

Syntax Let $\mathcal{C}, \mathcal{C}', \ldots$ denote *terms* of the calculus, often called *interactions* or *choreographies*; Terms describe a course of information exchange among two ore more parties from a global viewpoint. The syntax of the Global Calculus is given below.

Definition 2.1.12. *The syntax of the global calculus is given by the following grammar:*

$$
\begin{array}{rll}
\mathcal{C} ::= & \mathbf{0} & \text{(inaction)} \\
\mid & A \to B{:}a(k).\ \mathcal{C} & \text{(init)} \\
\mid & A \to B : k\langle \mathrm{l}, e, y \rangle & \text{(com)} \\
\mid & \mathcal{C}_1 \mid \mathcal{C}_2 & \text{(par)} \\
\mid & \textbf{if } e@A \textbf{ then } \mathcal{C}_1 \textbf{ else } \mathcal{C}_2 & \text{(cond)} \\
\mid & X & \text{(recvar)} \\
\mid & \mu X.\ C & \text{(recursion)}
\end{array}
$$

Some conventions follow:

- $A, B, C, \ldots$ range over a collection $\mathcal{P}$ of *participants*;
- $k, k', \ldots$ range over a collection $\mathcal{K}$ of *linear channels* (also called *session channels*). Session channels designate communication channels freshly generated for each session. In service technologies, they are realised by sending a freshly generated identifier as part of the message.

- $a, b, c, \ldots$ range over a collection $\mathcal{S}$ of *shared channels*, also called session initiating channels.
- $v, w, \ldots$ range over a collection Var of variables; $X, Y, \ldots$ are process variables, and $l, l_i, \ldots$ labels for branching;
- finally $e, e', \ldots$ range over unspecified arithmetic and other first-order expressions.

We write $e@A$ to mean that the expression e is evaluated using the variable related to participant A in the store.
Intuitively, the term (inaction) denotes a system where no interactions take place. (init) denotes a session initiation by A via B's service channel a, with a fresh session channel k and continuation $\mathcal{C}$. Note that k is bound in $\mathcal{C}$. (com) captures both the selection of an operation l as well as in-session communication of the expression e (at A's) over a session channel k. In this case, y does not bind in $\mathcal{C}$ (the semantics will treat y as a variable in the store of B). In (par), $\mathcal{C}_1 \mid \mathcal{C}_2$ denotes the parallel product between $\mathcal{C}_1$ and $\mathcal{C}_2$. (cond) denotes the standard conditional operator where $e@A$ indicates that the expression e has to be evaluated in the store of participant A. In (recursion), $\mu X.\ \mathcal{C}$ is the minimal fix point operation for recursion, where the variable X of (recvar) is bound in $\mathcal{C}$. The free and bound session channels and term variables are defined in the usual way. The calculus is equipped with a standard structural congruence $\equiv$, defined as the minimal congruence relation on interactions $\mathcal{C}$, such that $\equiv$ is a commutative monoid with respect to $\mid$ and $\mathbf{0}$, it is closed under alpha equivalence $\equiv_\alpha$ of terms, and it is closed under the recursion unfolding, i.e., $\mu X.\mathcal{C} \equiv \mathcal{C}[\mu X.\mathcal{C}/X]$.

Remark 2.1.13 (Differences with respect to [Carbone *et al.* 2007]). The syntax in Definition 2.1.12 presents a simplified version of the global calculus without restriction, summation and local assignments. In its original presentation [Carbone *et al.* 2006], restriction is used only during session initiation. We capture the requirement of fresh identifiers by using the operational rules in Figure 2.4. Excluding the lack of local assignment, we argue that our version of GC is, to some extent, as expressive as the one originally reported in [Carbone *et al.* 2007].

Semantics Terms in the global calculus are considered modulo structural congruence ($\equiv$) as the least congruence relation on $\mathcal{C}$ such that: (i) $(\mathcal{C}, \mathbf{0}, \mid)$ is a commutative monoid, and (ii) $\mathcal{C} \equiv_\alpha \mathcal{C}'$ if $\mathcal{C} \equiv_\alpha \mathcal{C}'$.

The semantics of the global calculus is defined as a reduction relation $(\sigma, \mathcal{C}) \rightarrow (\sigma', \mathcal{C}')$ which says that the choreography $\mathcal{C}$ with state σ performs an interaction and evolves into $\mathcal{C}'$ with state σ'. The state σ contains a set of variables labelled by participants. As described in the previous subsection, a variable x located at participant A is written as $x@A$. The same variable name labelled with different participant names denotes different variables (hence $\sigma(x@A)$ and $\sigma(x@B)$ may differ). The reduction relation $\rightarrow$ is defined as the least relation on state/choreography pairs satisfying the rules in Figure 2.4.

$$\frac{h \text{ is fresh}}{(\sigma, A{\to}B{:}a(k).\ \mathcal{C}) \longrightarrow (\sigma, \mathcal{C}[h/k])}\ \mathsf{G}-\mathsf{RInit} \qquad \frac{\mathcal{C} \equiv \mathcal{C}'' \quad (\sigma, \mathcal{C}) \longrightarrow (\sigma', \mathcal{C}') \quad \mathcal{C}' \equiv \mathcal{C}'''}{(\sigma, \mathcal{C}'') \longrightarrow (\sigma', \mathcal{C}''')}\ \mathsf{G}-\mathsf{RStruct}$$

$$\frac{(\sigma, \mathcal{C}[\mu X.\mathcal{C}/\mathcal{C}]) \longrightarrow (\sigma', \mathcal{C}')}{(\sigma, \mu X.\mathcal{C}) \longrightarrow (\sigma', \mathcal{C}')}\ \mathsf{G}-\mathsf{RRec} \qquad \frac{(\sigma, \mathcal{C}_1) \longrightarrow (\sigma', \mathcal{C}_1')}{(\sigma, \mathcal{C}_1 \mid \mathcal{C}_2) \longrightarrow (\sigma', \mathcal{C}_1' \mid \mathcal{C}_2)}\ \mathsf{G}-\mathsf{RPar}$$

$$\frac{\sigma(e@A) \Downarrow \mathsf{tt}}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2) \longrightarrow (\sigma, \mathcal{C}_1)}\ \mathsf{G}-\mathsf{RIfT} \qquad \frac{\sigma(e@A) \Downarrow \mathsf{ff}}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2) \longrightarrow (\sigma, \mathcal{C}_2)}\ \mathsf{G}-\mathsf{RIfF}$$

$$\frac{\sigma(e@A) \Downarrow v}{(\sigma, A{\to}B : k\langle \mathsf{l}, e, x\rangle.\ \mathcal{C}) \longrightarrow (\sigma[x@B \mapsto v], \mathcal{C})}\ \mathsf{G}-\mathsf{RCom}$$

Figure 2.4: Reduction Semantics for the Global Calculus

Intuitively, transition (G-RInit) describes the evolution of a session initiation: after A initiates a session with B on service channel a, A and B share the fresh channel h locally. (G-RCom) describes the main interaction rule of the calculus: the expression e is evaluated into v in the A-portion of the state σ and then assigned to the variable x located at B resulting in the new state $\sigma[x@B \mapsto v]$. (G-RIfT) and (G-RIfF) show the possible paths that a deterministic evolution of a choreography can produce. (G-RPar) and (G-RStruct) behave as the standard rules for parallel product and structural congruence, respectively.

Remark 2.1.14 (Global Parallel). Parallel composition in the global calculus differs from the notion of parallel found in standard concurrency models based on input/output primitives [Milner 1999]. In the latter, a term $P_1 \mid P_2$ may allow *interactions* between P_1 and P_2. However, in the global calculus, the parallel composition of two choreographies $\mathcal{C}_1 \mid \mathcal{C}_2$ concerns two parts of the described system where *interactions* may occur in $\mathcal{C}_1$ and $\mathcal{C}_2$ but never across the parallel operator $\mid$. This is because an interaction $A \to B \ldots$ abstracts from the actual end-point behaviour, i.e., how A sends and B receives. In this model, dependencies between two choreographies can be expressed by using variables in the state σ.

2.1.3.3 Structured Communications: The end-point Calculus

The end-point calculus (EPC) [Carbone *et al.* 2007] is the π-calculus [Milner 1999] extended with sessions [Honda *et al.* 1998] as well as locations [Hennessy 2007] and

store [Carbone *et al.* 2004]. Below, $P, Q, \ldots$ denote *processes*, $M, N, \ldots$ *networks*.

$$
\begin{array}{lllll}
P ::= & ! \, a(\tilde{k}).\ P & \text{(initin)} & \mid \ \overline{a}\langle \tilde{k} \rangle.\ P & \text{(initout)} \\
\mid & k \lhd \mathrm{l}\langle e \rangle.\ P & \text{(selection)} & \mid \ k \rhd \Sigma_i \mathrm{l}_i(y_i).\ P_i & \text{(branch)} \\
\mid & P_1 \oplus P_2 & \text{(plus)} & \mid \ P_1 \mid P_2 & \text{(par)} \\
\mid & \mu X.\ P & \text{(rec)} & \mid \ X & \text{(recvar)} \\
\mid & \textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2 & \text{(cond)} & & \\
\mid & \mathbf{0} & \text{(inact)} & &
\end{array}
$$

$$
\begin{array}{lll}
N ::= & A[\, P \,]_\sigma & \text{(participant)} \\
\mid & N_1 \mid N_2 & \text{(parnet)} \\
\mid & \varepsilon & \text{(inactnet)}
\end{array}
$$

(initin) and (initout) are dual operations for describing session initiation: $! \, a(\tilde{k}).\ P$ denotes a process offering a replicated (available in many copies) service a with session channels $\tilde{k}$ while $\overline{a}\langle \tilde{k} \rangle.\ P$ denotes a process requesting a service a with session channels $\tilde{k}$. In both cases, P is the continuation. The next two processes denote standard in-session communications (where y_i in the second construct, the branching input, is not bound in P_i, and $\{l_i\}$ are pairwise distinct). The term (PLUS) denotes internal choice. The rest is standard. Networks are parallel composition of participants, where a participant has the shape $A[\, P \,]_\sigma$, with A being the name of the participant, P its behaviour, and σ its local state, now interpreted as a local function from variables to values. We often omit σ when irrelevant. The free session channels, free term variables and service channels are defined as usual over processes and networks and, similarly to the global calculus, are denoted by $fsc(P/N), fv(P/N)$ and $channels(P/N)$ respectively. The syntax here presented differs from its original presentation in the absence of the local assignments, conditional operators, and restriction of networks and processes.

Semantics We give a semantics for EPC in terms of reductions of networks $N \to N'$. The reduction semantics follows the π-calculus and is defined by the rules given in Figure 2.5. Note that symmetric rules are omitted, and rules for restriction and variable assignment in the original version are omitted.

Rules in the reduction semantics for EPC treat processes and networks differently. Rules (E-RINIT) describe the interaction given by two end-points willing to establish a new session. Here, $! \, a(k).P$ denotes a replicated service. Reactions involving In-session communication and label selection are described by(E-RCOM), where $e \Downarrow v$ describes the evaluation of expression and $\sigma[x \mapsto v]$ the mapping variables x by the value v in σ. Rules (E-RPARP) and (E-RPARN) describe reductions over parallel composition of threads and parallel composition of networks. Rule (E-RSUM) is the standard rule for internal choice. Finally, (E-RSTRUCT) relates structural congruent networks.

Definition 2.1.15. *The structural congruence over end-point terms* ($\equiv$) *is defined as the least congruence on processes such that* $(\equiv, \mathbf{0}, \mid)$, $(\equiv, \mathbf{0}, \oplus)$, $(\equiv, \varepsilon, \mid)$ *are*

$$\text{E}-\text{RInit}\quad \frac{k_i \notin \text{fsc}(P') \cup \text{fsc}(Q') \quad \tilde{h} \text{ is fresh}}{A[\,!\,a(\tilde{k}).\ P \mid P'\,]_\sigma \mid B[\,\overline{a}\langle\tilde{k}\rangle.\ Q \mid Q'\,]_{\sigma'} \rightarrow (A[\,!\,a(\tilde{k}).\ P \mid P \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'})[\tilde{h}/\tilde{k}]}$$

$$\text{E}-\text{RCom}\quad \frac{\sigma \vdash e \Downarrow v \quad j \in I}{A[\,k \rhd \Sigma_i l_i(x_i).\ P_i \mid P'\,]_\sigma \mid B[\,k \lhd l_j\langle e\rangle.\ Q \mid Q'\,]_{\sigma'} \rightarrow A[\,P_j \mid P'\,]_{\sigma[x_j \mapsto v]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

$$\text{E}-\text{RIfT}\quad \frac{\sigma \vdash e \Downarrow \text{tt}}{A[\,\textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2 \mid P'\,]_\sigma \rightarrow A[\,P_1 \mid P'\,]_\sigma} \qquad \text{E}-\text{RParN}\quad \frac{M \rightarrow M'}{M|N \rightarrow M'|N}$$

$$\text{E}-\text{RIfF}\quad \frac{\sigma \vdash e \Downarrow \text{ff}}{A[\,\textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2 \mid P'\,]_\sigma \rightarrow A[\,P_2 \mid P'\,]_\sigma} \qquad \text{E}-\text{RSum}\quad \frac{i \in \{1,2\}}{A[P_1 \oplus P_2|R]_\sigma \rightarrow A[P_i|R]_\sigma}$$

$$\text{E}-\text{RRec}\quad \frac{A[\,P[\mu X.P/X] \mid Q\,]_\sigma \mid N \rightarrow N'}{A[\,\mu X.P \mid Q\,]_\sigma \mid N \rightarrow N'} \qquad \text{E}-\text{RStruct}\quad \frac{M \equiv M' \quad M' \rightarrow N' \quad N' \equiv N}{M \rightarrow N}$$

$$\text{E}-\text{RParP}\quad \frac{A[\,P_1 \mid R\,]_\sigma \rightarrow A[\,P_1' \mid R\,]_{\sigma'}}{A[\,P_1 \mid P_2 \mid R\,]_\sigma \rightarrow A[\,P_1' \mid P_2 \mid R\,]_{\sigma'}}$$

Figure 2.5: Reduction Relation for the End-Point Calculus

commutative monoids, and (i) $P \equiv Q$ *if* $P \equiv_\alpha Q$*, (ii)* $A[\,P\,]_\sigma \equiv A[\,Q\,]_\sigma$ *if* $P \equiv Q$*.*

2.1.4 The Conversation Calculus

Here we briefly introduce the Conversation Calculus (CC, in the following). Further details can be found at [Vieira *et al.* 2008, Vieira 2010].

The CC corresponds to a π-calculus with *labeled* communication and extended with *conversation contexts*. A conversation context can be seen as a medium in which interactions take place. It is similar to sessions in service-oriented calculi (see [Honda *et al.* 1998]) in the sense that every conversation context has an unique identifier (e.g.: an URI). Interactions in CC may be intuitively seen as communications in a pool of messages, where the pool is divided in areas identified by conversation contexts. Multiple participants can access many conversation contexts concurrently, provided they can get hold of the name identifying the context. Moreover, conversations can be nested multiple times (for instance, a private chat room within a multi-user chat application).

Definition 2.1.16 (CC **Syntax).** *Let $\mathcal{N}$ be an infinite set of names. Also, let $\mathcal{L}$, $\mathcal{V}$, and χ be infinite sets of labels, variables, and recursion variables, respectively. Using d to range over $\uparrow$ and $\downarrow$, the set of actions α and processes P is given below:*

$$\alpha ::= \mathsf{l}^d!(\vec{n}) \mid \mathsf{l}^d?(\vec{x}) \mid \mathbf{this}(x) \quad P, Q ::= n \blacktriangleleft [P] \mid \sum_{i \in I} \alpha_i.\ P_i \mid P \mid Q \mid (\nu n)\, P \mid \mu X.\ P \mid X$$

Above, $\vec{n}$ and $\vec{x}$ denote tuples of names and variables in $\mathcal{N}$ and $\mathcal{V}$, respectively. Actions can be an output $\mathsf{l}^d!(\vec{n})$ or an input $\mathsf{l}^d?(\vec{x})$, as in the π-calculus, with $\mathsf{l} \in \mathcal{L}$ in both cases. The *message direction* $\downarrow$ (read "here") decrees that the action it is associated to should take place in the *current* conversation context, while $\uparrow$ (read "up") decrees that the action should take place in the *enclosing* one. We often omit the "here" direction, and write $\mathsf{l}?(y).P$ and $\mathsf{l}!(\vec{n}).P$ rather than $\mathsf{l}^{\downarrow}?(y).P$ and $\mathsf{l}^{\downarrow}!(\vec{n}).P$. The context-aware prefix $\mathbf{this}(x)$ binds the name of the enclosing conversation context to x. The syntax of processes includes the conversation context $n \blacktriangleleft [P]$, where $n \in \mathcal{N}$. We follow the standard π-calculus interpretation for guarded choice, parallelism, restriction, and recursion (for which we assume $X \in \chi$). As usual, given $\sum_{i \in I} \alpha_i.\ P_i$, we write $\mathbf{0}$ when $|I| = 0$, and $\alpha_1.\ P_1 + \alpha_2.\ P_2$ when $|I| = 2$. We assume the usual definitions of free/bound variables and free/bound names for a process P, noted $fv(P), bv(P)$ and $fn(P), bn(P)$, respectively. The set of names of a process is defined as $n(P) = fn(P) \cup bn(P)$. Finally, notice that labels in $\mathcal{L}$ are not subject to restriction or binding.

The semantics of the CC is given as a labeled transition system (LTS). As customary, a transition $P \xrightarrow{\lambda} P'$ represents the evolution from P to P' through action λ. We write $P \xrightarrow{\lambda}$ if $P \xrightarrow{\lambda} P'$, for some P'. We define $P \longrightarrow P'$ as $P \xrightarrow{\tau} P'$. We use $P \longrightarrow^* P'$ to denote the transitive closure of $P \longrightarrow P'$, and write $P \stackrel{\lambda}{\Rightarrow} P'$ when $P \longrightarrow^* \xrightarrow{\lambda} \longrightarrow^* P'$.

Definition 2.1.17. *Transition* labels λ *are defined in terms of* actions σ*, as defined by the following grammar:*

$$\sigma ::= \tau \mid \mathsf{l}^d?(\vec{x}) \mid \mathsf{l}^d!(\vec{n}) \mid \mathsf{this} \qquad \lambda ::= \sigma \mid c\,\sigma \mid (\nu n)\,\lambda$$

Action τ denotes internal communication, while $\mathsf{l}^d?(\vec{x})$ and $\mathsf{l}^d!(\vec{n})$ represent an input and output to the environment, respectively. Action this represents a conversation identity access. A transition label λ can be either the (unlocated) action σ, an action σ *located at* conversation c (written $c\,\sigma$), or a transition label in which n is bound with scope λ. This is the case of bounded output actions. $out(\lambda)$ denotes the names produced by a transition, so $out(\lambda) = a$ if $\lambda = \mathsf{l}^d!(a)$ or $\lambda = c\mathsf{l}^d!(a)$ and $c \neq a$. A transition label λ denoting communication, such as $\mathsf{l}^d?(\vec{x})$ or $\mathsf{l}^d!(\vec{n})$ is subject to *duality* $\overline{\lambda}$. We write $\overline{\mathsf{l}^d?(\vec{x})} = \mathsf{l}^d!(\vec{n})$ and $\overline{\mathsf{l}^d!(\vec{n})} = \{\mathsf{l}^d?(\vec{x}) \mid \vec{x} \in \mathcal{V}\}$.

Figure 2.6 presents the LTS. The rules in the upper part of Fig. 2.6 follow the transition rules for a π-calculus with recursion. For instance, rule (CC-Open) corresponds to the usual scope extrusion rule in the π-calculus. The rest of the rules are specific to the CC. Rule (CC-This) captures the name of an enclosing conversation context. Rule (CC-LocL) locates an action to a particular conversation context,

$$\text{(CC-In)}\quad \frac{}{\lfloor^d?(\vec{x}).\ P \xrightarrow{\lfloor^d?(\vec{n})} P[\vec{n}/\vec{x}]} \qquad \text{(CC-Out)}\quad \frac{}{\lfloor^d!(\vec{n}).\ P \xrightarrow{\lfloor^d!(\vec{n})} P} \qquad \text{(CC-This)}\quad \frac{}{\mathbf{this}(x).\ P \xrightarrow{c\ \mathtt{this}} P[c/x]}$$

$$\text{(CC-Open)}\quad \frac{P \xrightarrow{\lambda} Q \quad n \in out(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} Q} \qquad \text{(CC-Res)}\quad \frac{P \xrightarrow{\lambda} Q \quad n \notin n(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} (\nu n)\ Q} \qquad \text{(CC-Sum)}\quad \frac{\alpha_j.\ P_j \xrightarrow{\lambda} P'_j \quad j \in I}{\sum_{i\in I} \alpha_i.\ P_i \xrightarrow{\lambda} P'_j}$$

$$\text{(CC-Par1)}\quad \frac{P \xrightarrow{\lambda} P' \quad bn(\lambda)\#fn(Q)}{P \mid Q \xrightarrow{\lambda} P' \mid Q} \qquad \text{(CC-Comm1)}\quad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{(CC-Rec)}\quad \frac{P[X/\mu X.\ P] \xrightarrow{\lambda} Q}{\mu X.\ P \xrightarrow{\lambda} Q}$$

$$\text{(CC-Close1)}\quad \frac{P \xrightarrow{(\nu \vec{n})\ \overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad \vec{n}\#fn(Q)}{P \mid Q \xrightarrow{\tau} (\nu \vec{n})\ (P' \mid Q')} \qquad \text{(CC-LocL)}\quad \frac{P \xrightarrow{\lambda^{\downarrow}} P'}{c \blacktriangleleft [P] \xrightarrow{c\ \lambda^{\downarrow}} c \blacktriangleleft [P']} \qquad \text{(CC-HereL)}\quad \frac{P \xrightarrow{\lambda^{\uparrow}} P'}{c \blacktriangleleft [P] \xrightarrow{\lambda^{\downarrow}} c \blacktriangleleft [P']}$$

$$\text{(CC-ThisClose1)}\quad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu n)\ c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} (\nu n)\ (P' \mid Q')} \qquad \text{(CC-ThisComm1)}\quad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} P' \mid Q'} \qquad \text{(CC-ThruL)}\quad \frac{P \xrightarrow{a\ \lambda^{\downarrow}} P'}{c \blacktriangleleft [P] \xrightarrow{a\ \lambda^{\downarrow}} c \blacktriangleleft [P']}$$

$$\text{(CC-TauL)}\quad \frac{P \xrightarrow{\tau} P'}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [P']} \qquad \text{(CC-ThisLocL)}\quad \frac{P \xrightarrow{c\ \mathtt{this}} P'}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [P']}$$

Figure 2.6: An LTS for CC. Rules with labels ending with "1" have a symmetric counterpart (with label ending with "2") which is elided.

and rule (CC-HereL) changes the direction of an action occurring inside a context. Rules (CC-ThisClose1) and (CC-ThisComm1) are located versions of (CC-Close) and (CC-Comm), respectively. Rule (CC-ThisLocL) hides an action occurring inside a conversation context. Rules (CC-ThruL) and (CC-TauL) formalise how actions change when they "cross" a conversation context.

2.2 Verification

2.2.1 Linear Temporal Logic

Temporal logics were introduced into computer science by Pnueli [Pnueli 1977] and thereafter proven to be a good basis for specification as well as for automated rea-

soning about concurrent, reactive systems.

We recall the syntax and semantics of FLTL. We refer the reader to [Manna & Pnueli 1992] for further details.

Recall that a signature Σ is a set of constant, function and predicate symbols. A first-order language $\mathcal{L}$ is built from the symbols in Σ, a denumerable set of variables $x, y, \ldots$, and the logic symbols $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall$, tt and ff

Definition 2.2.1 (FLTL Syntax). *Given a first-order language $\mathcal{L}$, FLTL formulae is given by the syntax:*

$$F, G, \ldots ::= \mathsf{c} \mid F \wedge G \mid F \vee G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F.$$

Where c *is a predicate symbol in $\mathcal{L}$.*

As done in Model Theory, the non-logical symbols of $\mathcal{L}$ (predicate, function and constant symbols) are given meaning in an underlying $\mathcal{L}$-structure, or $\mathcal{L}$-model, $M(\mathcal{L}) = (\mathcal{I}; D)$. This means, they are interpreted via $\mathcal{I}$ as relations over a domain $\mathcal{D}$ of the corresponding arity.

States and Interpretations: A state s is a mapping assigning to each variable $x \in \mathcal{L}$ a value $s[x]$ in $\mathcal{D}$. This interpretation is extended to $\mathcal{L}$-expressions in the usual way, for example, $s[f(x)] = \mathcal{I}(f)(s[x])$. We write $s \models_{M(L)} \mathsf{c}$ if and only if ⌋ is true with respect to s in $\mathcal{M}(\mathcal{L})$.

The state s is said to be an x-variant of s' iff $s'[y] = s[y]$ for each $y \neq x$. This is, s and s' are the same except possible for the value of the variable x.

We shall use $\sigma, \sigma', \ldots$ to range over infinite sequences of states. We say that σ is an x-variant of σ' iff for each $i \geq 0$, $\sigma(i)$ is an x-variant of $\sigma'(i)$.

Definition 2.2.2 (FLTL Semantics). *We say that σ satisfies F in a $\mathcal{L}$-structure $\mathcal{M}(\mathcal{L})$, written $\sigma \models_{\mathcal{M}(\mathcal{L})} F$, iff $\langle \sigma, 0 \rangle \models_{\mathcal{M}(\mathcal{L})} F$ following the assertions in Figure 2.7.*

We say that F is valid in $\mathcal{M}(\mathcal{L})$ if and only if for all σ, $\sigma \models_{\mathcal{M}(\mathcal{L})} F$. F is said to be valid if F is valid for every model $\mathcal{M}(\mathcal{L})$.

Ensuring specifications have *no communication errors*.

2.2.2 Session Types for the Global Calculus

The Global Calculus comes accompanied with a type discipline that ensures the proper control flow among interactions. It is built as a generalisation of session types [Honda *et al.* 1998] for global interactions, first presented in [Carbone *et al.* 2007]. Here we informally describe their use through examples, and direct to their original presentation for a more formal view.

Session types in GC are used to structure sequence of message exchanges in a session. Their syntax is as follows:

$$\begin{aligned} \theta &= \mathsf{bool} \mid \mathsf{int} \mid \ldots \\ \alpha &= k \rhd \Sigma_i \mathsf{l}_i(\theta_i).\ \alpha_i \mid k \lhd \Sigma_i \mathsf{l}_i(\theta_i).\ \alpha_i \mid \alpha_1 \mid \alpha_2 \mid \mathsf{end} \mid \mu \mathbf{t}.\ \alpha \mid \mathbf{t} \end{aligned} \tag{2.1}$$

$$
\begin{array}{lcl}
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \mathsf{tt} & & \\
\langle \sigma, i\rangle \not\models_{\mathcal{M}(\mathcal{L})} \mathsf{ff} & & \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \mathsf{c} & \stackrel{\text{def}}{=} & \sigma(i) \models_{\mathcal{M}(\mathcal{L})} \mathsf{c} \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \neg F & \stackrel{\text{def}}{=} & \langle \sigma, i\rangle \not\models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} F \wedge G & \stackrel{\text{def}}{=} & \langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} F \text{ and } \langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} G \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \ominus F & \stackrel{\text{def}}{=} & i > 0 \text{ and } \langle \sigma, i-1\rangle \models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \circ F & \stackrel{\text{def}}{=} & \langle \sigma, i+1\rangle \models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \exists t.\ F & \stackrel{\text{def}}{=} & \text{for some x-variant } \sigma' \text{ of } \sigma, \langle \sigma', i\rangle \models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \Box F & \stackrel{\text{def}}{=} & \forall j \geq i, \langle \sigma, j\rangle \models_{\mathcal{M}(\mathcal{L})} F
\end{array}
$$

Figure 2.7: FLTL semantics

Here, θ range over standard data types bool, string, int, ... and α describe session types. We describe the forms of α.

- $k \triangleright \Sigma_i \mathsf{l}_i(\theta_i).\ \alpha_i$ and $k \triangleleft \Sigma_i \mathsf{l}_i(\theta_i).\ \alpha_i$ are branching-input and selection-output types, they describe the provision of processes with labelled inputs (or a labelled output, respectively) followed by the continuation α_i.
- The type $\alpha_1 \mid \alpha_2$ is a parallel composition of session types α_1 and α_2.
- The type end indicates session termination and is often omitted.
- $\mu \mathbf{t}.\ \alpha$ indicates a recursive type with $\mathbf{t}$ as a type variable. $\mu \mathbf{t}.\ \alpha$ binds the free occurrences of $\mathbf{t}$ in α. We take an *equi-recursive* view on types, not distinguishing between $\mu \mathbf{t}.\ \alpha$ and its unfolding $\alpha[\mu \mathbf{t}.\ \alpha/\mathbf{t}]$.

Typing judgments in GC have the form $\Gamma \vdash \mathcal{C} \triangleright \Delta$, where Γ is a type environment describing *services*, and Δ the type environment describing *sessions*. Typically, Γ contains a set of type assignments of the form $a@A : \alpha$, which say that a service a located at participant A may be invoked and run a session according to type α. Δ contains type assignments of the form $k[A, B] : \alpha$ which say that a session channel k identifies a session between participants A and B and has session type α when seen from the viewpoint of A. There is no particular reason why one has to choose a strict direction when considering interactions, and one may as well consider $k[A, B] : \alpha$ from the viewpoint of B. Consider an online booking scenario (as the one in Equation 6.2.5). One possible type assignment for Δ is:

$$k_1, k_2[Cust, AC] : k_1 \triangleright \text{booking(string)}.\ k_2 \triangleleft \text{offer(int)}.\ k_1 \triangleright \text{accept(string)}.\ \text{end}$$

Describing that k_1 and k_2 are names corresponding to the same session between participants $Cust$ and AC, and corresponds to the session type $\alpha = k_1 \triangleright$

booking(string). $k_2 \lhd$ offer(int). $k_1 \rhd$ accept(string). end when seeing it from the point of view of $Cust$.

We provide some examples on the typing rules for the GC. The full set typing rules are similar to the original ones attached in Appendix 6.B[1]. First, we comment the rule (G-TInit), which types the establishment of a new session between two participants.

$$\frac{\Gamma, a@B : (\vec{k})\alpha \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[B, A] : \alpha \quad A \neq B}{\Gamma, a@B : (\vec{k})\alpha \vdash A{\to}B{:}a(\vec{k}).\ \mathcal{C} \triangleright \Delta} \; \mathrm{G-TInit}$$

Here, the typing rule dictates some requirements on the structure of the choreography: first, the initialisation of a session between participants in $A{\to}B{:}a(\vec{k}).\ \mathcal{C}$ requires that sessions names in $\vec{k}$ correspond to a session type in the premise. Moreover, it checks that the service channel $a@B : (\vec{k})\alpha$ is declared in the service typing Γ. The rule (G-TCom) describes communication between participants:

$$\frac{\Gamma \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad k \in \vec{k} \quad A \neq B \quad j \in J}{\Gamma \vdash A{\to}B : k\langle \mathsf{l}_i, e, x\rangle.\ \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : k \rhd \Sigma_{i\in J}\mathsf{l}_i(\theta_i).\ \alpha_i} \; \mathrm{G-TCom}$$

Here, the interaction $\mathcal{C} = A{\to}B : k\langle \mathsf{l}_i, e, x\rangle.\ \mathcal{C}'$ will be typable with a session type $\Delta \cdot \vec{k}[A, B] : k \rhd \Sigma_{i\in J}\mathsf{l}_i(\theta_i).\ \alpha_i$ provided that: 1) The evaluation of the expression e at A and its recipient variable x at B correspond to the same value type, 2) the communication is performed between different participants A and B, 3) the continuation $\mathcal{C}$ contains a session type between A and B such that its session names in $\vec{k}$ contain k, and 4) the branch selected in i is a valid one. In the conclusion, we use an output type $k \rhd \Sigma_{i\in J}\mathsf{l}_i(\theta_i).\ \alpha_i$ describing the emision of value from the point of view of A. It is clear, that we could use a complementary rule to type the input of values from the point of view of B.

Assumption 2.2.3 (Well-typedness). Henceforth we only consider well-typed terms for the Global calculus, unless otherwise specified.

2.2.3 Session Types for the End-Point Calculus

Session types for the EPC builds from the syntax of session types in equation 2.1. Basically, the type discipline of the EPC stems from the Global Calculus, but assigns session types to every single participant instead of the whole choreography. In this way, the session typing in the EPC describes the end-point behaviour. An *end-point typing judgment* contains judgements for processes in the form $\Gamma \vdash_A P \triangleright \Delta$ (where P is typed as a behaviour for A) and judgements for networks $\Gamma \vdash N \triangleright \Delta$. In both,

[1] Pay attention that the session types used in Appendix 6.B bear some differences from the ones exemplified here. Note there that branching inputs and selections are replaced by corresponding separate input-output types and branching-selection types.

mappings Γ and Δ are *service* and *session typings* respectively. Here, Γ and Δ are defined as:

$$\Gamma ::= \emptyset \mid \Gamma, a@A:(\tilde{k})\alpha \mid \Gamma, \overline{a}@A:(\tilde{k})\alpha \mid \Gamma, x@A:\theta \mid \Gamma, X:\Delta$$
$$\Delta ::= \emptyset \mid \Delta, \tilde{k}@A:\alpha \mid \Delta, \tilde{k}:\bot$$

Above, $a@A:(\tilde{k})\alpha$ indicates the service located at A which is invoked with fresh session channels $\tilde{k}$ and offers service of the shape α, while $\overline{a}@A:(\tilde{k})\alpha$ indicates the type abstraction for the dual invocation, i.e. a client of an A's service which invokes with fresh channels $\tilde{k}$ and engages in interactions abstracted as α. Note $@A$ indicates the location of a service in both forms. As before, $\tilde{k}$ should be a vector of pairwise distinct session channels which should cover all session channels in α, and α does not contain free type variables. $(\tilde{k})$ binds occurrences of session channels in $(\tilde{k})$ in α, which induces the standard alpha-equality. A central concept in this type discipline is the notion of duality for session types, which is defined as:

$$\overline{(\tilde{k})\alpha@A} = ?(\tilde{k})\overline{\alpha}@A \qquad \overline{?(\tilde{k})\alpha@A} = (\tilde{k})\overline{\alpha}@A$$

where the notion of duality α of α remains the same.

Here we only comment some examples on the typing rules, and the full type system can be found in Appendix 6.D[2]. Similarly as with the type system for the Global Calculus, we will focus the examples in session initiation and communication. The two rules (E-TInit.In),(E-TInit.Out) describe session initiation primitives:

$$\frac{\Gamma \vdash_A P \triangleright \tilde{k}@A:\alpha \quad a \notin dom(\Gamma) \quad client(\Gamma)}{\Gamma, !\,a(\tilde{k})\alpha@A \vdash_A !a(\tilde{k}).\ P \triangleright \emptyset}\ \text{E-TInit.In} \qquad \frac{\Gamma, a:(\tilde{k})\alpha@B \vdash_A P \triangleright \Delta \cdot \tilde{k}@A:\alpha}{\Gamma, a:(\tilde{k})\alpha@B \vdash_A \overline{a}\langle\tilde{k}\rangle P \triangleright \Delta}\ \text{E-TInit.Out}$$

In (E-TInit.In), the premise only allows for typings of session channels involved in the session initialisation of service a, that is, only the channels in $\tilde{k}$. This linearity condition blocks free session channels from occurring during a replicated input. The condition $a \notin dom(\Gamma)$ prevents from self-calls and ensures that the type assignment occurs at the side of the client. Requirements for the complementary typing rule (E-TInit.In) are analogous, although the linearity condition is removed. Communication rules are standard for session types, for instance, the rule (E.TSelOut) is used to type message outputs:

$$\frac{j \in I \quad \Gamma \vdash e:\theta_j \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{k}@A:\alpha_j}{\Gamma \vdash_A k \lhd l_j\langle e\rangle.\ P \triangleright \Delta \cdot \tilde{k}@A:k \lhd \Sigma_{i\in I} l_i(\theta_i).\ \alpha_i}\ \text{E.TSelOut}$$

Here, process $k \lhd l_j\langle e\rangle.\ P$ types after evaluation that the typing of e corresponds to a correct value type and that the continuation P behaves as established by the session type in $\Delta \cdot \tilde{k}@A:\alpha$. Analogous requirements hold for typing the input process $k \rhd \Sigma_i l_i(y_i).\ P_i$.

[2]The same considerations for global types apply also for end-point types.

2.2.4 End Point Projection

The relation between global and local views at the specification of communication protocols is given at the level of types. The central idea is that one can *project* the behaviour (type) of a global specification given in terms of choreography into a parallel composition of the behaviours of end-points. The mapping is far from trivial, and need to preserve causal relations between messages and threads, namely *connectedness*, *well-threadedness* and *coherence*. The next subsection presents a recap from the work at [Carbone *et al.* 2007]. In order to give the formal definition of end point projection, we first annotate global specifications with identifiers for threads.

An annotated interaction, is an annotation of a choreography with t's denoting each thread in play. Annotated interactions are written $\mathcal{A}, \mathcal{A}', \ldots$, and they are given by the following grammar:

$$\begin{array}{rlll} \mathcal{A} ::= & A^{t_1} \rightarrow B^{t_2} : a(k).\ \mathcal{A} & \mid & \mathcal{A}_1 |^t \mathcal{A}_2 \\ \mid & A^{t_1} \rightarrow B^{t_2} : k\langle \mathsf{l}, e, y \rangle.\ \mathcal{A} & \mid & \mu^t X^A.\ \mathcal{A} \\ \mid & \textbf{if } e@A^t \textbf{ then } \mathcal{A}_1 \textbf{ else } \mathcal{A}_2 & \mid & X_t^A \\ \mid & \mathbf{0} & & \end{array}$$

where each t is a natural number. We call $t, t', \cdots$ occurring in an annotated interaction, *threads*. Each $\mathcal{A}$ can be regarded as an abstract syntax built from a constructor in its root (either a prefix or a parallel product), if the tree is originated from a single thread, or a pair of threads if the interaction involves an interaction (session initiation, message communication or selection/branching). The following is the consistent annotation of online booking example referred before[3].

$$\begin{array}{l} \mathsf{Cust}^1 \rightarrow \mathsf{AC}^2 : \mathsf{ob}(k_1, k_2).\ \mathsf{Cust}^1 \rightarrow \mathsf{AC}^2 : k_1\langle \mathsf{booking}, \mathsf{Paris}, x \rangle. \qquad (OB_{\mathcal{A}}) \\ \quad \mathsf{AC}^2 \rightarrow \mathsf{Cust}^1 : k_2\langle \mathsf{offer}, \$\ 100.00, y \rangle.\ \mathsf{Cust}^1 \rightarrow \mathsf{AC}^2 : k_1\langle \mathsf{accept}, \mathsf{cardNr.12345}, z \rangle.\ \mathbf{0} \end{array}$$

Which, although simple, could be more complicated in the case there are more than one session initiation involved in the choreography. Take for instance the case where $\mathsf{Cust} \rightarrow \mathsf{AC} : \mathsf{ob}(k_1, k_2)$ is decomposed by the sequence of processes $\mathsf{Cust} \rightarrow \mathsf{AC} : \mathsf{ob}(k_1).\ \mathsf{AC} \rightarrow \mathsf{Cust} : \mathsf{ob}(k_2)$ We can have different annotations for Cust and AC. The sequence: $\mathsf{Cust}^1 \rightarrow \mathsf{AC}^2 : \mathsf{ob}(k_1).\ \mathsf{AC}^2 \rightarrow \mathsf{Cust}^3 : \mathsf{ob}(k_2)$ generates a valid annotation as it places each session initiation between the customer and the AC in different threads, any other annotation would be invalid.

A choreography $\mathcal{C}$ is *connected*, if the interactions within $\mathcal{C}$ describe strongly connected sequences of interactions where active/passive participants (the ones originating/receivers of an interaction). Informally, for each participant A in the set of

[3]The data types for the messages in each of the interactions are obvious, therefore they will not be described.

participants of a choreography $\mathcal{C}$, a communication activity originated by A should have been immediately by a communication activity where A had acted as a receiver, or been preceded by a self-contained action (evaluation of expressions, for instance).

Consistent annotations In order to provide meaningful projections between choreographies and its end-points, we need to define a notion of "consistent annotation", that is, an annotation $\mathcal{A}$ such that it respects causality conditions, and can be realised by a projection. Such conditions are: 1) Causal Consistency: if a participant annotated with t is passive in an interaction (a receiver), then the subsequent interaction will be marked with t as well, or it will be a self-contained action, 2) Session Consistency: Two actions in $\mathcal{A}$ identified by the same session name are annotated with the same thread, and 3) Distinctness Condition: The input of session initiation is always given a fresh thread.

The *Well-threadedness* condition ensures global specifications are free from unrealisable dependencies among actions. We say $\mathcal{A}$ is well-threaded if it is connected and it has a consistent annotation.

Mergeability Annotations in a choreography allow for the extraction of threads directly from the global behaviour. As threads are sequences of actions to be executed at each end-point, we need to ensure that threads generated from choreographical annotations are meaningful, in the sense that they project only to the required end-points, and threads describing the behaviour of the same end point are encapsulated (*merged*) on a single service description. Mergeability, denoted by $\bowtie$, is the smallest equivalence over typed terms up to $\equiv$, closed under all typed contexts and

$$\frac{\forall l \in (I \cap J).(P_l \bowtie Q_l \wedge x_l = y_l) \qquad \forall i \in I \backslash J. \forall j \in J \backslash I. \mathsf{l}_i \neq \mathsf{l}_j}{k \rhd \Sigma_i \mathsf{l}_i(x_i).\ P_i \bowtie k \rhd \Sigma_j \mathsf{l}_j(y_j).\ Q_j} \ \mathrm{M-In} \qquad \frac{fsc(P) = 0}{P \bowtie \mathbf{0}} \ \mathrm{M-Zero}$$

When $P \bowtie Q$, we say that P and Q are mergeable.

Above, a context is any end-point calculus process with some holes. (M-In) is for branching and says that we can allow differences in branches which do not overlap, but we do demand each pair of behaviours with the same operation to be identical.

The operation $P \sqcup Q$ allows for merging typed processes as long as they are mergeable according to the rules above. $P \sqcup Q$ is a partial commutative binary operator on typed processes which is well-defined iff $P \bowtie Q$. We see an example of the merging rules, and the full set can be consulted in Appendix 6.E. The merging of two branching processes $k \rhd \Sigma_{i \in I} \mathsf{l}_i(x_i).\ P_i$ and $k \rhd \Sigma_{i \in J} \mathsf{l}_i(x_i).\ P_i$ is given as:

$$k \rhd \Sigma_{i \in I} \mathsf{l}_i(x_i).\ P_i \sqcup k \rhd \Sigma_{i \in J} \mathsf{l}_i(x_i).\ P_i \stackrel{\text{def}}{=} k \rhd \left(\begin{array}{l} \Sigma_{i \in I \cap J} \mathsf{l}_i(y_i).\ (P_i \sqcup Q_i) \\ + \Sigma_{i \in I \setminus J} \mathsf{l}_i(y_i).\ P_i \\ + \Sigma_{i \in J \setminus I} \mathsf{l}_i(y_i).\ Q_i \end{array} \right)$$

That is, the resulting merge groups in a single session branching all the options coming from multiple branches that have the same session key.

Given a consistent annotation, we can project each of its threads onto an end-point process. The thread projection $\mathsf{TP}(\mathcal{A}, t)$ is a partial operation that uses the merge operator, some of the rules are given below (the full set are included in Appendix 6.F):

$$\mathsf{TP}(A^{t_1} \to B^{t_2} : b(\tilde{k}).\ \mathcal{A}, t) \stackrel{\text{def}}{=} \begin{cases} \overline{b}\langle \tilde{k} \rangle.\ \mathsf{TP}(\mathcal{A}, t_1) & \text{if } t = t_1 \\ !\, b(\tilde{k}).\ \mathsf{TP}(\mathcal{A}, t_2) & \text{if } t = t_2 \\ \mathsf{TP}(\mathcal{A}, t) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(A^{t_1} \to B^{t_2} : k\langle \mathsf{l}, e, x \rangle.\ \mathcal{A}, t) \stackrel{\text{def}}{=} \begin{cases} k \lhd \mathsf{l}\langle e \rangle.\ \mathsf{TP}(A, t) & \text{if } t = t_1 \\ k \rhd \mathsf{l}(x).\ \mathsf{TP}(A, t) & \text{if } t = t_2 \\ \mathsf{TP}(\mathcal{A}, t) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(\textbf{if } e@A^{t'} \textbf{ then } \mathcal{A}_1 \textbf{ else } \mathcal{A}_2\ , t) \stackrel{\text{def}}{=} \begin{cases} \textbf{if } e \textbf{ then } \mathsf{TP}(\mathcal{A}_1, t') \textbf{ else } \mathsf{TP}(\mathcal{A}_2, t') & \text{if } t = t' \\ \mathsf{TP}(\mathcal{A}_1, t) \sqcup \mathsf{TP}(\mathcal{A}_2, t) & \text{otherwise} \end{cases}$$

Definition 2.2.4 (Coherent Interactions). *Given a well-threaded, consistently annotated interaction $\mathcal{A}$, we say that $\mathcal{A}$ is coherent if the following two conditions hold:*

1. *For each thread t in $\mathcal{A}$, $TP(\mathcal{A}, t)$ is well-defined.*
2. *For each pair of threads t_1, t_2 in $\mathcal{A}$ with $t_1 \equiv_A t_2$, we have $TP(A, t_1) \bowtie TP(\mathcal{A}, t_2)$.*

Below, $part(\mathcal{C})$ denotes the set of participants names occurring in $\mathcal{C}$. Recall also being coherent entails being well-typed, connected and well-threaded.

Definition 2.2.5 (End-Point Projection). *Let $\mathcal{C}$ be a coherent interaction, and $\mathcal{A}$ be a consistent annotation of $\mathcal{C}$. Then the* end point projection of $\mathcal{A}$ under a state σ, *denoted* $\mathsf{EPP}(\mathcal{A}, \sigma)$, *is given as the following network.*

$$\mathsf{EPP}(\mathcal{A}, \sigma) \stackrel{def}{=} \Pi_{A \in part(\mathcal{C})}\ A[\, \Pi_{[t]} \bigsqcup_{t' \in [t]} \mathsf{TP}(\mathcal{A}, t')\,]_{\sigma @ A}$$

The mapping given above is defined after choosing a specific annotation of an interaction. The following result shows the map in fact does not depend on a specific (consistent) annotation chosen, as far as a global description has no incomplete threads, i.e. it has no free session channels (which is what programmers/designers usually produce).

Theorem 2.2.6 (Soundness and Completeness of End-point Projections[Carbone *et al.* 2007]). *Assume $\mathcal{A}$ is well-typed, strongly connected, well-threaded and coherent. Assume further $\Gamma \vdash \mathcal{A} \triangleright \Delta$ and $\Gamma \vdash \sigma$. Then the following properties hold:*

- *(soundness) if $\mathsf{EPP}(\mathcal{A}, \sigma) \longrightarrow N$ then there exists $\mathcal{A}'$ such that $(\sigma, \mathcal{A}) \longrightarrow (\sigma', \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.*

- *(completeness) If* $(\sigma, \mathcal{A}) \longrightarrow (\sigma', \mathcal{A}')$ *then there exist* N *such that* $\mathrm{EPP}(\mathcal{A}, \sigma) \longrightarrow N$ *and* $\mathrm{EPP}(\mathcal{A}', \sigma') \prec N$.

- *(soundness with action labels) if* $\mathrm{EPP}(\mathcal{A}, \sigma) \xrightarrow{\mathrm{m}} N$ *then there exists* $\mathcal{A}'$ *such that* $(\sigma, \mathcal{A}) \xrightarrow{\ell} (\sigma', \mathcal{A}')$ *such that* $\mathrm{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.

- *(completeness with action labels) If* $(\sigma, \mathcal{A}) \xrightarrow{\ell} (\sigma', \mathcal{A}')$ *then there exist* N *such that* $\mathrm{EPP}(\mathcal{A}, \sigma) \xrightarrow{\mathrm{m}} N$ *and* $\mathrm{EPP}(\mathcal{A}', \sigma') \prec N$.

Where $\equiv_{rec}$ denotes equality induced by the unfolding of process recursion. The assymetric relation $P \prec Q$ indicates that P is the result of cutting off "unnecessary branches" of Q, in the light of P's own typing, is formally defined as follows:

Definition 2.2.7 (Pruning). *Let* $\Gamma \vdash_A P \rhd \Delta$ *for* Γ *and* Δ *minimal and* $\Gamma, \Gamma' \vdash_A Q \rhd \Delta$. *If further we have* $Q \equiv Q_0 \mid !R$ *where* $\Gamma \vdash Q_0 \rhd \Delta$, $\Gamma' \vdash_A R$ *and* $P \sqcup Q_0$, *then we can write:* $\Gamma \vdash_A P \prec Q\,\Delta$ *or* $P \prec Q$ *for short, and say* P *prunes* Q *under* $\Gamma; \Delta$. $\prec$ *is extended to networks accordingly.*

2.3 Behavioural Equivalences between Processes

One central concern of concurrency theory is to determine whether two processes exhibit the same behaviour; to this end, many notions of behavioural equivalence have been investigated [van Glabbeek 1990]. In this section, we will recall some of the behavioural equivalences used in this thesis.

2.3.1 Simulations & Bisimulations

The following definitions are presented with respect to the definition of transition systems given in Definition 2.1.1.

Definition 2.3.1 (Simulation [Milner 1999]). *Two transition systems* $T_i = \langle S_i, l_i, \longrightarrow_i, \mathrm{Act} \rangle$ *for* $i \in \{0, 1\}$ *are* similar *if there exists a relation* $\mathcal{R} \subseteq S_0 \times S_1$ *such that* $\forall s \in l_i . \exists s' \in l_{1-i} . s \mathcal{R} s'$ *and for all* s_0', $s_0 \xrightarrow{a}_0 s_0'$ *implies* $s_1 \xrightarrow{a}_1 s_1'$ *and* $s_0' \mathcal{R} s_1'$.

Definition 2.3.2 (Bisimulation [Milner 1999]). *Two transition systems* $T_i = \langle S_i, l_i, \longrightarrow_i, \mathrm{Act} \rangle$ *for* $i \in \{0, 1\}$ *are* similar *if there exists a relation* $\mathcal{S} \subseteq S_0 \times S_1$ *such that both* $\mathcal{S}$ *and its converse* $\mathcal{S}^{-1}$ *are simulations. We say that* T_0 *and* T_1 *are bisimilar, written* $T_0 \sim T_1$ *if there exists a bisimulation* $\mathcal{S}$ *such that* $T_0 \mathcal{S} T_1$.

A coalgebraic approach to supervisory control introduced the notion of *partial bisimulation* as a behavioural relation suitable for controllability [Rutten 2000]. In principle, it suggest that controllable event should be simulated, whereas uncontrollable events should be bisimulated, hence the term partial bisimulation.

Definition 2.3.3 (Partial Bisimulation [Rutten 2000]). *Two transition systems* $T_i = \langle S_i, I_i, \longrightarrow_i, \mathrm{Act}\rangle$ *for* $i \in \{0,1\}$ *are* partially bisimilar *(with respect to the bisimulation set B), written* $T_0 \preceq_B T_1$*, if there exists a relation* $\mathcal{T} \subseteq S_0 \times S_1$ *such that,* $\forall s \in I_i . \exists s' \in I_{1-i} . s \mathcal{T} s'$ *and whenever* $s_0 \mathcal{T} s_1$*, we have that*

1. *For all* s_0', $s_0 \xrightarrow{a}_0 s_0'$ *implies* $s_1 \xrightarrow{a}_1 s_1'$ *and* $s_0' \mathcal{T} s_1'$;
2. *For all* s_1' *and* $b \in B$, $s_1 \xrightarrow{b}_1 s_1'$ *implies* $s_0 \xrightarrow{b}_0 s_0'$ *and* $s_0' \mathcal{T} s_1'$.

A recent proposal for a simulation-based behavioural relation over LTS has been put forward in [Fábregas *et al.* 2010]. The *covariant-contravariant simulation* is based on considering a partition of their set of actions into three sets: a collection of covariant actions (being in control of the specification), a collection of contravariant actions (being under the control of the implementation–or the environment–) and a collection of bivariant actions (these ones treating the classic notion of bisimulation).

Definition 2.3.4 (covariant-contravariant simulation [Fábregas *et al.* 2010]). *Assume a partition* $\{B^r, B^l, B^{bi}\}$ *of the set of actions, i.e.* $\mathrm{Act} = B^r \uplus B^l \uplus B^{bi}$*. Two transition systems* $T_i = (S_i, I_i, \longrightarrow_i, \mathrm{Act})$ *for* $i \in \{0,1\}$ *with such partioned action set are* (B^r, B^l)–similar *(or just a* covariant-contravariant similar*) if there exists a relation* $\mathcal{U} \subseteq S_0 \times S_1$ *such that,* $\forall s \in I_i . \exists s' \in I_{1-i} . s \mathcal{U} s'$ *and whenever* $s_0 \mathcal{U} s_1$*, we have that*

- *for all* $a \in B^r \cup B^{bi}$ *and all* $s_0 \xrightarrow{a} s_0'$ *there exists* $s_1 \xrightarrow{a} s_1'$ *with* $s_0' \mathcal{U} s_1'$.
- *for all* $a \in B^l \cup B^{bi}$ *and all* $s_1 \xrightarrow{a} s_1'$ *there exists* $s_0 \xrightarrow{a} s_0'$ *with* $s_0' \mathcal{U} s_1'$.

2.3.2 Testing Theories

A different approach, proposed in [De Nicola & Hennessy 1984], is based on tests. Intuitively two processes are testing equivalent, $p \approx_{test} q$, relative to a set of tests T if p and q pass exactly the same set of tests from T. Much here depends on the nature of tests, how they are applied and how they succeed.

Informally processes are conceived as completely independent entities who may or may not react to testing requests; more importantly the application of a test to a process simply consists of a run to completion of the process in a test harness. Because processes are in general nondeterministic, formally this leads to two testing based equivalences, $p \approx_{may} q$ and $p \approx_{must} q$; the latter is determined by the set of tests a process guarantees to pass, written p *must satisfy* t, while the former by those it is possible to pass, p *may satisfy* t. Notably, must equivalences have been used as foundational theories to analyse the compliance between web service specifications, as presented in [Laneve & Padovani 2007], and endow a logical characterisation based on HML logic [Cerone & Hennessy 2010].

Formally speaking, a test is a state from a LTS $T_o = \langle S, \mathrm{Act}_\tau \cup \{\omega\}, \xrightarrow{a}\rangle$ where ω is an action not contained in Act and τ does not occur in Act.

Given a LTS of processes $T_p = \langle S, \mathrm{Act}_\tau, \xrightarrow{a} \rangle$, an experiment consists of a pair $p \mid t$ from the product LTS $(T_p \mid T_o)$. We refer to a maximal path $p \mid t \xrightarrow{\tau} p_1 \mid t_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_k \mid t_k \xrightarrow{\tau} \ldots$ as a computation of $p \mid t$. It may be finite or infinite; it is successful if there exists some $n \geq 0$ such that $t_n \xrightarrow{\omega}$. As only τ-actions can be performed in an experiment, we will omit the symbol τ in computations and in computation prefixes. Successful computations lead to the definition of two well known testing relations, [De Nicola & Hennessy 1984]:

Definition 2.3.5 (May Satisfy, Must Satisfy). *Assuming a LTS of processes and a LTS of tests, let s and t be a state and a test from such LTSs, respectively. We say*

1. s may satisfy t *if there exists a successful computation for the experiment $s \mid t$.*
2. s must satisfy t *if each computation of the experiment $s \mid t$ is successful.*

Chapter 3

A Unified Framework for Declarative Structured Communications

Abstract: We present a unified framework for the declarative analysis of structured communications. By relying on a (timed) concurrent constraint programming language, we show that in addition to the usual operational techniques from process calculi, the analysis of structured communications can elegantly exploit logic-based reasoning techniques. We introduce a concurrent constraint a declarative interpretation of the language for structured communications proposed by Honda, Vasconcelos, and Kubo. Distinguishing features of our approach are: the possibility of including partial information (constraints) in the session model; the use of explicit time for reasoning about session duration and expiration; a tight correspondence with logic, which formally relates session execution and linear-time temporal logic formulas. We provide compelling examples of our approach, as well as comment on directions of current and future work.

Contents

3.1 Introduction

3.1.1 Motivation.

As recently pointed out by the ICT theme of EU Seventh Framework Programme (FP7), the need of trustworthy and pervasive services infrastructure is considered one of the three mayor challenges in ICT for the next ten years. The "future internet" proposes challenges in terms of scalability, mobility, flexibility, security, trust and robustness to a current Internet architecture of more than 30 years old. A vast landscape of application and ever-changing requirements and environments must be supported, and new ways of interaction must be devised, coping with safety and reliability in their coordination methods. Service Oriented Computing (SOC for short) has emerged as one of the driving forces

From the viewpoint of *reasoning techniques*, two main trends in modeling in Service Oriented Computing (SOC) can be singled out. On the one hand, an *operational approach* focuses on how process interactions can lead to correct configurations. Typical representatives of this approach are based on process calculi and Petri nets (see, e.g., [van der Aalst 1998, Boreale *et al.* 2006, Lanese *et al.* 2007, Lapadula *et al.* 2007a]), and count with behavioral equivalences and type disciplines as main analytic tools. On the other hand, in a *declarative approach* the focus is on the set of conditions components should fulfill in order to be considered correct, rather than on the complete specification of the control flows within process activities (see, e.g., [van der Aalst & Pesic 2006, Pesic & van der Aalst 2006]). Even if these two trends address similar concerns, we find that they have evolved rather independently from each other.

The quest for a unified approach in which operational and declarative techniques can harmoniously converge is therefore a legitimate research direction. In this paper we shall argue that Concurrent Constraint Programming (CCP) [Saraswat 1993] can serve as a foundation for such an approach. Indeed, the unified framework for operational and logic techniques that CCP provides can be fruitfully exploited for analysis in SOC, possibly in conjunction with other techniques such as type systems. Below we briefly introduce the CCP model and then elaborate on how it can shed light on a particular issue: the analysis of structured communications.

CCP [Saraswat 1993] is a well-established model for concurrency where processes interact with each other by *telling* and *asking* for pieces of information (*constraints*) in a shared medium, the *store*. While the former operation simply adds a given constraint to the store (thus making it available for other processes), the latter allows for rich, parameterizable forms of process synchronization. Interaction is thus inherently *asynchronous*, and can be related to a broadcast-like communication discipline, as opposed to the point-to-point discipline enforced by formalisms such as the π-calculus [Sangiorgi & Walker 2001]. In CCP, the information in the store grows monotonically, as constraints cannot be removed. This condition is relaxed in *timed* extensions of CCP (e.g., [Saraswat *et al.* 1994, Nielsen *et al.* 2002]), where processes evolve along a series of *discrete time units*. Although each unit contains its own

store, information is not automatically transferred from one unit to another. In this paper we shall adopt a CCP process language that is timed in this sense.

In addition to the traditional operational view of process calculi, CCP enjoys a *declarative* nature that distinguishes it from other models of concurrency: CCP programs can be seen, at the same time, as computing agents and as logic formulas [Saraswat 1993, Nielsen *et al.* 2002, Olarte & Valencia 2008b], i.e., they can be read and understood as logical specifications. Hence, CCP-based languages are suitable for *both* the specification and verification of programs. In the CCP language used in this paper, processes can be interpreted as linear-time temporal logic formulas; we shall exploit this correspondence to verify properties of our models.

3.1.2 This Work.

We describe initial results on the definition of a formal framework for the declarative analysis of structured communications. We shall exploit utcc [Olarte & Valencia 2008a], a timed CCP process calculus, to give a declarative interpretation to the language defined by Honda, Vasconcelos, and Kubo in [Honda *et al.* 1998] (henceforth referred to as HVK). This way, structured communications can be analyzed in a declarative framework where time is defined explicitly. We begin by proposing an encoding of the HVK language into utcc and studying its correctness. We then move to the timed setting, and propose HVK$^{\mathsf{T}}$, a timed extension of HVK. The extended language explicitly includes information on session duration, allows for declarative preconditions within session establishment constructs, and features a construct for session abortion. We then discuss how the encoding of HVK into utcc straightforwardly extends to HVK$^{\mathsf{T}}$.

3.1.3 A Compelling Example.

We now give intuitions on how a declarative approach could be useful in the analysis of structured communications. Consider the ATM example from [Honda *et al.* 1998, Sect. 4.1], given below:

$$
\begin{array}{l}
ATM(a,b) = \textbf{accept } a(k) \textbf{ in } k![id];\\
\qquad k \rhd \left\{ \begin{array}{l}
deposit : \textbf{request } b(h) \textbf{ in}\\
\quad k?(amt)\,.\; h \lhd deposit;\\
\quad h![id, amt]; ATM(a,b)\\
\|\, withdraw : \textbf{request } b(h) \textbf{ in}\\
\quad k?(amt)\,.\; h \lhd withdraw; h![id, amt];\\
\quad h \rhd \left\{ \begin{array}{l} success : k \lhd dispense; k![amt]; ATM(a,b)\\ \|\, failure : k \lhd overdraft; ATM(a,b) \end{array} \right\}\\
\|\, balance : \textbf{request } b(h) \textbf{ in } h \lhd balance; h?(amt)\,.\\
\quad k![amt]; ATM(a,b)
\end{array} \right\}
\end{array}
$$

Here, an ATM has established two sessions: the first one with a user, sharing session k over service a, and the second one with the bank, sharing session h over service b. The ATM offers `deposit`, `balance`, and `withdraw` operations. When executing a withdraw, if there is not enough money in the account, then an *overdraft* message appears to the user. It is interesting to analyze what occurs when this scenario is extended to consider a card reader that acts as a malicious interface between the user and the ATM. The user communicates his personal data with the reader using the service r, which will be kept by the reader after the first withdraw operation to continue withdrawing money without the authorization of the user. A greedy card reader could even withdraw repeatedly until causing an overdraft (labelled "over"), as expressed below:

$$\begin{aligned} \mathit{Reader} = & \textbf{accept } r(k') \textbf{ in } k'?(id)\,.\ \textbf{request } a(k) \textbf{ in } k![id]; \\ & k' \rhd \left\{ \begin{array}{l} withdraw : k'?(amt)\,.\ k \lhd withdraw; k![amt]; \\ \quad k \rhd \{dispense : k' \lhd dispense; k![amt]; R(k, amt) \,||\, over : Q\} \end{array} \right\} \end{aligned}$$

$$R(j, x) = \textbf{def } R' \textbf{ in } k \lhd withdraw; j![x]; j \rhd \Big\{ dispense : j?(amt)\,.\ R' \Big|\Big| over : Q \Big\}$$

$$\begin{aligned} \mathit{User} = & \textbf{request } r(k') \textbf{ in } k'![myId]; \\ & k' \lhd withdraw; k'![58]; k' \rhd \Big\{ dispense : k'?(amt)\,.\ P \Big|\Big| over : Q \Big\} \end{aligned}$$

By creating sessions between them, the card reader Reader is able to receive the user's information, and to use it later by attempting a session establishment with the bank. Following authentication steps (not modeled above), the card reader allows the user to obtain the requested amount. Additional withdrawing transactions between the reader and the bank are defined by the recursive process R. In the specification above, the process Q can be assumed to send a message (through a session with the bank) representing the fact that the account has run out of money: $Q = k_{bank}![\underline{0}]; \textbf{inact}$.

Even in this simple scenario, the combination of operational and declarative reasoning techniques may come in handy to reason about the possible states of the system. Indeed, while an operational approach can be used to describe an operational description of the compromised ATM above, the declarative approach can complement such a description by offering declarative insights regarding its evolution. For instance, assuming Q as above, one could show that a utcc specification of the ATM example satisfies the linear temporal logic formula $\Diamond\, \texttt{out}(k_{bank}, 0)$, which intuitively means that in presence of a malicious card reader the user's bank account will eventually reach an overdraft status.

3.1.4 Related Work.

One approach to combine the declarative flavor of constraints and process calculi techniques is represented by a number of works that have extended name-passing calculi with some form of partial information (see, e.g., [Victor & Parrow 1998, Diaz *et al.* 1998]). The crucial difference between such a strand of work and CCP-based calculi is that the latter offer a tight correspondence with logic, which greatly broadens the spectrum of reasoning techniques at one's disposal. Recent works similar to ours include CC-Pi [Buscemi & Montanari 2007] and the calculus for structured communications in [Coppo & Dezani-Ciancaglini 2009]. Such languages feature elements that resemble much ideas underlying CCP (especially [Buscemi & Montanari 2007]). The main difference between our approach and such works is that we adhere to the use of declarative reasoning techniques based on temporal logic as an effective way of complementing operational reasoning techniques. In [Buscemi & Montanari 2007], the reasoning techniques associated to CC-Pi are essentially operational, and used to reason about service-level agreement protocols. In [Coppo & Dezani-Ciancaglini 2009], the key for analysis is represented by a type system which provides consistency for session execution, much as in the original approach in [Honda *et al.* 1998].

Plan of the Document. Section 3.2 recalls the syntax and main intuitions of the session language in [Honda *et al.* 1998] and introduces utcc, the timed CCP language we shall use to give a declarative interpretation of sessions. Such an interpretation is presented in Section 3.3, where additional timed sessions constructs are motivated and introduced. An extended example is presented in Section 3.4.1. Section 3.5 offers some concluding remarks and outlines directions for ongoing and future work.

3.2 Preliminaries

3.2.1 A Language for Structured Communication

We begin by introducing HVK, a language for structured communication proposed in [Honda *et al.* 1998]. We assume the following conventions: *names* are ranged over by $a, b, \ldots$; *channels* are ranged over by k, k'; *variables* are ranged over by $x, y, \ldots$; *constants* (names, integers, booleans) are ranged over by $c, c', \ldots$; *expressions* (including constants) are ranged over by $e, e', \ldots$; *labels* are ranged over by $l, l', \ldots$; *process variables* are ranged over by $X, Y, \ldots$. Finally, $u, u', \ldots$ denote names and channels. We shall use $\vec{x}$ to denote a sequence (tuple) of variables $x_1 \ldots x_n$ of length $n = |\vec{x}|$. Notation $\vec{x}$ will be similarly applied to other syntactic entities. The sets of free names/channels/variables/process variables of P, is defined in the standard way, and are respectively denoted by $\mathsf{fn}(\cdot)$, $fc(\cdot)$, $\mathsf{fv}(\cdot)$, and $fpv(\cdot)$. Processes without free variables or free channels are called *programs*.

Definition 3.2.1 (The HVK language [Honda *et al.* 1998]). *Processes in* HVK *are built from:*

$$
\begin{array}{rcll}
P,Q & ::= & \textbf{request}\ a(k)\ \textbf{in}\ P & \textit{Session Request} \\
 & | & \textbf{accept}\ a(k)\ \textbf{in}\ P & \textit{Session Acceptance} \\
 & | & k![\vec{e}];\ P & \textit{Data Sending} \\
 & | & k?(\vec{x})\,.\ P & \textit{Data Reception} \\
 & | & k \lhd l;P & \textit{Label Selection} \\
 & | & k \rhd \{l_1 : P_1 \,[]\, \cdots \,[]\, l_n : P_n\} & \textit{Label Branching} \\
 & | & \textbf{throw}\ k[k'];\ P & \textit{Channel Sending} \\
 & | & \textbf{catch}\ k(k')\ \textbf{in}\ P & \textit{Channel Reception} \\
 & | & \textbf{if}\ e\ \textbf{then}\ P\ \textbf{else}\ Q & \textit{Conditional Statement} \\
 & | & P \mid Q & \textit{Parallel Composition} \\
 & | & \textbf{inact} & \textit{Inaction} \\
 & | & (\nu u)P & \textit{Hiding} \\
 & | & \textbf{def}\ D\ \textbf{in}\ P & \textit{Recursion} \\
 & | & X[\vec{e}\,\vec{k}] & \textit{Process Variables} \\
D & ::= & X_1(x_1k_1) = P_1\ \textbf{and}\ \cdots\ \textbf{and}\ X_n(x_nk_n) = P_n & \\
 & & & \textit{Declaration for Recursion}
\end{array}
$$

3.2.1.1 Operational Semantics of HVK.

The operational semantics of HVK is given by the reduction relation $\longrightarrow_h$ which is the smallest relation on processes generated by the rules in Figure 3.1.

Definition 3.2.2 (Structural Congruence). *The structural congruence relation $\equiv_h$ is the smallest relation satisfying:*

(i) $P \equiv_h Q$ *if they differ only by a renaming of bound variables (alpha-conversion).*

(ii) $P \mid \textbf{inact} \equiv_h P$, $P \mid Q \equiv_h Q \mid P$, $(P \mid Q) \mid R \equiv_h P \mid (Q \mid R)$.

(iii) $(\nu u)\textbf{inact} \equiv_h \textbf{inact}$, $(\nu u u')P \equiv_h (\nu u' u)P$, $(\nu u)(P \mid Q) \equiv_h (\nu u)P \mid Q$ *if* $x \notin \mathrm{fv}(Q)$, $(\nu u)(\textbf{def}\ D\ \textbf{in}\ P) \equiv_h (\textbf{def}\ D\ \textbf{in}\ ((\nu u)P))$ *if* $u \notin \mathrm{fv}(D)$.

(iv) $(\textbf{def}\ D\ \textbf{in}\ P) \mid Q \equiv_h \textbf{def}\ D\ \textbf{in}\ (P \mid Q)$ *if* $fpv(D) \cap fpv(Q) = \emptyset$.

(v) $\textbf{def}\ D\ \textbf{in}\ (\textbf{def}\ D'\ \textbf{in}\ P) \equiv_h \textbf{def}\ D$ *and* $D'\ \textbf{in}\ P$ *if* $fpv(D) \cap fpv(D') = \emptyset$.

Let us give some intuition about the language constructs and the rules in Figure 3.1. The central idea in HVK is the notion of a *session*, i.e., a series of reciprocal interactions between two parties, possibly with branching, delegation and recursion, which serves as an abstraction unit for describing structured communication. Each session has associated a specific port, or *channel*. Channels are generated at session initialization; communications inside the session take place on the same channel.

More precisely, sessions are initialized by a process of the form $\textbf{request}\ a(k)\ \textbf{in}\ Q \mid \textbf{accept}\ a(k)\ \textbf{in}\ P$. In this case, there is a request, on name a, for the initiation of a session and the generation of a fresh channel. This request is matched by an accepting process on a, which generates a new channel k, thus allowing P and Q to communicate each other. This is the intuition behind rule Link. Three kinds of

$$\textbf{request}\ a(k)\ \textbf{in}\ Q \mid \textbf{accept}\ a(k)\ \textbf{in}\ P \longrightarrow_h (\nu k)(P \mid Q) \quad \text{(Link)}$$

$$(k![\vec{e}]; P) \mid (k?(\vec{x})\,.\,Q) \longrightarrow_h P \mid Q[\vec{c}/\vec{x}] \quad \text{if } e \downarrow \vec{c} \quad \text{(Com)}$$

$$k \lhd l_i; P \mid k \rhd \left\{ l_1 : P_1 \,\middle|\middle|\, \cdots \,\middle|\middle|\, l_n : P_n \right\} \longrightarrow_h P \mid Pi\ (1 \leq i \leq n) \quad \text{(Label)}$$

$$\textbf{throw}\ k[k']; P \mid \textbf{catch}\ k(k')\ \textbf{in}\ Q \longrightarrow_h P \mid Q \quad \text{(Pass)}$$

$$\textbf{if}\ e\ \textbf{then}\ P\ \textbf{else}\ Q \longrightarrow_h P\ (e \downarrow \mathrm{tt}) \quad \text{(If1)}$$

$$\textbf{if}\ e\ \textbf{then}\ P\ \textbf{else}\ Q \longrightarrow_h Q\ (e \downarrow \mathrm{ff}) \quad \text{(If2)}$$

$$\textbf{def}\ D\ \textbf{in}\ (X[\vec{e}\,\vec{k}] \mid Q) \longrightarrow_h \textbf{def}\ D\ \textbf{in}\ (P[\vec{c}/\vec{x}] \mid Q)\ (e \downarrow \vec{c}, X(\vec{x}\vec{k}) = P \in D) \quad \text{(Def)}$$

$$P \longrightarrow_h P' \text{ implies } (\nu u)P \longrightarrow_h (\nu u)P' \quad \text{(Scop)}$$

$$P \longrightarrow_h P' \text{ implies } P \mid Q \longrightarrow_h P' \mid Q \quad \text{(Par)}$$

$$\text{If } P \equiv_h P' \text{ and } P' \longrightarrow_h Q' \text{ and } Q' \equiv_h Q \text{ then } P \longrightarrow_h Q \quad \text{(Str)}$$

Figure 3.1: Reduction Semantics of HVK ($\longrightarrow_h$)[Honda *et al.* 1998].

atomic interactions are available in the language: sending (including name passing), branching, and channel passing (also referred to as delegation). Those actions are described by rules Com, Label, and Pass, respectively. In the case of Com, the expression $\vec{e}$ is sent on the port (session channel) k. Process $k?(\vec{x})\,.\,Q$ then receives such a data and executes $Q[\vec{c}/\vec{x}]$, where $\vec{c}$ is the result of evaluating the expression $\vec{e}$. The case of Pass is similar but considering that in the constructs **throw** $k[k']; P$ and **catch** $k(k')$ **in** Q, only session names can be transmitted. In the case of Label, the process $k \lhd l_i; P$ selects one label and then the corresponding process P_i is executed. The other rules are self-explanatory.

For the sake of simplicity, and without loss of generality (due to rule 5 of $\equiv_h$), in the sequel we shall assume programs of the form **def** D **in** P where there are not procedure definitions in P.

3.2.2 Timed Concurrent Constraint Programming

Timed concurrent constraint programming (tcc) [Saraswat *et al.* 1994] extends CCP for modeling reactive systems. In tcc, time is conceptually divided into *time units* (or *time intervals*). In a particular time unit, a tcc process P gets an input (i.e. a constraint) c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines also a residual process Q which is then executed in the next time unit. It is worth noticing that the final store is not automatically transferred to the next time unit.

The utcc calculus [Olarte & Valencia 2008a] extends tcc for reactive systems featuring mobility. Here *mobility* is understood as the dynamic reconfiguration of system linkage through communication, much like in the π-calculus [Sangiorgi & Walker 2001]. utcc generalizes tcc by considering a *parametric* ask operator of the form (**abs** $\vec{x}; c)\, P$, with the following intuitive meaning: process $P[\vec{t}/\vec{x}]$ is executed for every term $\vec{t}$ such that the current store entails an admissible substitution $c[\vec{t}/\vec{x}]$. This process can be seen as an *abstraction* of the process P on the variables $\vec{x}$ under the constraint (or with the *guard*) c.

utcc provides a number of reasoning techniques: First, utcc processes can be represented as partial closure operators (i.e. idempotent and extensive functions). Also, for a significant fragment of the calculus, the input-output behavior of a process P can be retrieved from the set of fixed points of its associated closure operator [Olarte & Valencia 2008b]. Second, utcc processes can be characterized as First-order Linear-time Temporal Logic (FLTL) formulas [Manna & Pnueli 1992]. This declarative view of the processes allows for the use of the well-established verification techniques from FLTL to reason about utcc processes.

Syntax. Processes in utcc are parametric in a *constraint system* [Saraswat 1993] which specifies the basic constraints that agents can tell or ask during execution. It also defines an *entailment* relation "$\Vdash$" specifying interdependencies among constraints. Intuitively, $c \Vdash d$ means that the information in d can be deduced from that in c (as in, e.g., $x > 42 \Vdash x > 0$).

The notion of constraint system can be set up by using first-order logic (see e.g., [Nielsen *et al.* 2002]). We assume a first-order signature Σ and a (possibly empty) first-order theory Δ, i.e., a set of sentences over Σ having at least one model. Constraints are then first-order formulas over Σ. Consequently, the entailment relation is defined as follows: $c \Vdash d$ if the implication $c \Rightarrow d$ is valid in Δ.

The syntax of the language is as follows:

$$
\begin{array}{rll}
P, Q := & \textbf{skip} & \\
 & \mid \textbf{tell}(c) & \\
 & \mid (\textbf{abs}\ \vec{x}; c)\, P & \\
 & \mid P \,||\, Q & \\
 & \mid (\textbf{local}\ \vec{x}; c)\, P & \\
 & \mid \textbf{next}\ P & \\
 & \mid \textbf{unless}\ c\ \textbf{next}\ P & \\
 & \mid\ !\, P &
\end{array}
$$

with the variables in $\vec{x}$ being pairwise distinct.

A process **skip** does nothing; a process **tell**(c) adds c to the store in the current time interval. A process $Q = (\textbf{abs}\ \vec{x}; c)\, P$ binds the variables $\vec{x}$ in P and c. It executes $P[\vec{t}/\vec{x}]$ for every term $\vec{t}$ s.t. the current store entails an admissible substitution over $c[\vec{t}/\vec{x}]$. The substitution $[\vec{t}/\vec{x}]$ is admissible if $|\vec{x}| = |\vec{t}|$ and no x_i in $\vec{x}$ occurs in $\vec{t}$.

Furthermore, Q evolves into **skip** at the end of the time unit, i.e., abstractions are not persistent when passing from one time unit to the next one. $P \,||\, Q$ denotes P and Q running in parallel during the current time unit. A process $(\mathbf{local}\,\vec{x}; c)\,P$ *binds* the variables $\vec{x}$ in P by declaring them private to P under a constraint c. If $c = \mathrm{tt}$, we write $(\mathbf{local}\,\vec{x})\,P$ instead of $(\mathbf{local}\,\vec{x}; \mathrm{tt})\,P$. The *unit delay* $\mathbf{next}\,P$ executes P in the next time unit. The *time-out* $\mathbf{unless}\ c\ \mathbf{next}\,P$ is also a unit delay, but P is executed in the next time unit iff c is not entailed by the final store at the current time unit. Finally, the *replication* $!\,P$ means $P \,||\, \mathbf{next}\,P \,||\, \mathbf{next}^2 P \,||\, \ldots$, i.e., an unbounded number of copies of P but one at a time. We shall use $!_{[n]}P$ to denote *bounded replication*, i.e., $P \,||\, \mathbf{next}\,P \,||\, \ldots \,||\, \mathbf{next}^{n-1} P$.

From a programming language perspective, variables $\vec{x}$ in $(\mathbf{abs}\,\vec{x}; c)\,P$ can be seen as the formal parameters of P. This way, *recursive definitions* of the form $X(\vec{x}) \stackrel{\mathtt{def}}{=} P$ can be encoded in utcc as

$$\mathcal{R}[X(\vec{x}) \stackrel{\mathtt{def}}{=} P] = !\,(\mathbf{abs}\ \vec{x}; \mathit{call}_X(\vec{x}))\,\widehat{P} \tag{3.1}$$

where call_X is an uninterpreted predicate (a constraint) of arity $|\vec{x}|$. Process $\widehat{P}$ is obtained from P by replacing recursive calls of the form $X(\vec{t})$ with $\mathbf{tell}(\mathit{call}_X(\vec{t}))$. Similarly, calls of the form $X(\vec{t})$ in other processes are replaced with $\mathbf{tell}(\mathit{call}_X(\vec{t}))$.

3.2.2.1 Operational Semantics.

The operational semantics considers *transitions* between process-store *configurations* $\langle P, c\rangle$ with stores represented as constraints and processes quotiented by the structural congruence $\equiv_u$ defined below. We shall use $\gamma, \gamma', \ldots$ to range over configurations.

The semantics is given in terms of an *internal* and an *observable* transition relation; both are given in Figure 3.2. In $\mathrm{R_A}$, $\vec{x} \neq \vec{t}$ ($\vec{x}$ syntactically different from $\vec{t}$) denotes $\bigvee_{1\le i\le|\vec{x}|} x_i \neq t_i$. If $|\vec{x}| = 0$, $\vec{x} \neq \vec{t}$ is defined as ff. The *internal transition* $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ informally means "P with store d reduces, in one internal step, to P' with store d'". We sometimes abuse of notation by writing $P \longrightarrow P'$ when d, d' are unimportant. The *observable transition* $P \xrightarrow{(c,d)} R$ means "P on input c, reduces *in one time unit* to R and outputs d". The latter is obtained from a finite sequence of internal transitions.

Definition 3.2.3 (Structural Congruence). *The structural congruence $\equiv_u$ is the smallest congruence satisfying:*

(i) $P \equiv_u Q$ *if they differ only by a renaming of bound variables,*

(ii) $P \,||\, \mathbf{skip} \equiv_u P$,

(iii) $P \,||\, Q \equiv_u Q \,||\, P$,

(iv) $P \,||\, (Q \,||\, R) \equiv_u (P \,||\, Q) \,||\, R$,

(v) $P \,||\, (\mathbf{local}\,\vec{x}; c)\,Q \equiv_u (\mathbf{local}\,\vec{x}; c)\,(P \,||\, Q)$ *if* $\vec{x} \notin \mathrm{fv}(P)$,

$$\mathrm{R_T}\ \frac{}{\langle \mathbf{tell}\,(\mathrm{d}),\mathrm{c}\rangle \longrightarrow \langle \mathbf{skip},\mathrm{c}\wedge\mathrm{d}\rangle} \qquad \mathrm{R_P}\ \frac{\langle P,\mathrm{c}\rangle \longrightarrow \langle P',\mathrm{c}'\rangle}{\langle P \,||\, Q,\mathrm{c}\rangle \longrightarrow \langle P' \,||\, Q,\mathrm{c}'\rangle} \qquad \mathrm{R_S}\ \frac{\gamma_1' \longrightarrow \gamma_2'}{\gamma_1 \longrightarrow \gamma_2}\ \text{if}\ \begin{matrix}\gamma_1 \equiv \gamma_1',\\ \gamma_2 \equiv \gamma_2'\end{matrix}$$

$$\mathrm{R_R}\ \frac{}{\langle !P,\mathrm{c}\rangle \longrightarrow \langle P \,||\, \mathbf{next}\,(!P),\mathrm{c}\rangle} \qquad \mathrm{R_U}\ \frac{d \Vdash c}{\langle \mathbf{unless}\,\mathrm{c}\,\mathbf{next}\,P,\mathrm{d}\rangle \longrightarrow \langle \mathbf{skip},\mathrm{d}\rangle}$$

$$\mathrm{R_L}\ \frac{\langle P,(\exists \tilde{\mathrm{x}}\mathrm{d})\wedge\mathrm{c}\rangle \longrightarrow \langle P',(\exists\tilde{\mathrm{x}}\mathrm{d})\wedge\mathrm{c}'\rangle}{\langle (\mathbf{local}\ \vec{x};\mathrm{c})\ P,\mathrm{d}\rangle \longrightarrow \langle (\mathbf{local}\ \vec{x};\mathrm{c}')\ P',(\exists\tilde{\mathrm{x}}\mathrm{c}')\wedge\mathrm{d}\rangle}$$

$$\mathrm{R_A}\ \frac{\mathrm{d}\Vdash \mathrm{c}[\vec{t}/\vec{x}] \quad |\vec{t}| = |\vec{x}|}{\langle (\mathbf{abs}\ \vec{x};\mathrm{c})\,P,\mathrm{d}\rangle \longrightarrow \langle P[\vec{t}/\vec{x}] || (\mathbf{abs}\ \vec{x};\mathrm{c}\wedge(\tilde{\mathrm{x}}\neq\tilde{\mathrm{t}}))\,P,\mathrm{d}\rangle}$$

$$\mathrm{R_O}\ \frac{\langle P,\mathrm{c}\rangle \longrightarrow^* \langle Q,\mathrm{d}\rangle \not\longrightarrow}{P \xrightarrow{(c,d)} F(Q)}$$

$$\text{Where } F(Q) = \begin{cases} \mathbf{skip} & \text{if } Q = \mathbf{skip} \\ F(Q_1) \,||\, F(Q_2) & \text{if } Q = Q_1 \,||\, Q_2 \\ R & \text{if } Q = \mathbf{next}\,(R) \\ \mathbf{skip} & \text{if } Q = (\lambda\ \vec{x};\mathrm{c})\ R \\ (\mathbf{local}\ \vec{x})\ F(R) & \text{if } Q = (\mathbf{local}\ \vec{x};\mathrm{c})\ R \\ R & \text{if } Q = \mathbf{unless}\,\mathrm{c}\,\mathbf{next}\,R \end{cases}$$

Figure 3.2: Transition System for utcc: Internal and Observable transitions

(vi) $(\mathbf{local}\,\vec{x};c)\,(\mathbf{local}\,\vec{y};d)\,P \equiv_u (\mathbf{local}\,\vec{x};\vec{y}\ ;c\wedge d)\,P \quad$ *if* $\vec{x}\cap\vec{y}=\emptyset$ *and* $\vec{y}\notin \mathrm{fv}(c)$.

We extend $\equiv_u$ *by decreeing that* $\langle P,c\rangle \equiv_u \langle Q,c\rangle$ *iff* $P\equiv_u Q$.

Definition 3.2.4 (Output Behavior). *Let* $s = c_1.c_2....c_n$ *be a sequence of constraints. If* $P = P_1 \xrightarrow{(\mathrm{tt},c_1)} P_2 \xrightarrow{(\mathrm{tt},c_2)} \ldots P_n \xrightarrow{(\mathrm{tt},c_n)} P_{n+1} \equiv_u Q$ *we shall write* $P \xrightarrow{s}{}^* Q$. *If* $s = c_1.c_2.c_3\ldots$ *is an infinite sequence, we omit* Q *in* $P \xrightarrow{s}{}^* Q$. *The* output behavior *of* P *is defined as* $o(P) = \{s \mid P \xrightarrow{s}{}^*\}$. *If* $o(P) = o(Q)$ *we shall write* $P \sim^o Q$. *Furthermore, if* $P \xrightarrow{s} Q$ *and* s *is unimportant we simply write* $P \longrightarrow^* Q$.

Logic Correspondence. Remarkably, in addition to this operational view, utcc processes admit a declarative interpretation based on Pnueli's first-order linear-time temporal logic (FLTL) [Manna & Pnueli 1992]. This is formalized by the encoding below, which maps utcc processes into FLTL formulas.

Definition 3.2.5. *Let* TL[·] *a map from* utcc *processes to FLTL formulas given by:*

$$\begin{aligned}
&\mathrm{TL}[\mathbf{skip}] = \mathrm{tt}\\
&\mathrm{TL}[\mathbf{tell}(c)] = c\\
&\mathrm{TL}[P \,||\, Q] = \mathrm{TL}[P] \wedge \mathrm{TL}[Q]\\
&\mathrm{TL}[(\mathbf{abs}\ \vec{y}; c)\ P] = \forall \vec{y}(c \Rightarrow \mathrm{TL}[P])\\
&\mathrm{TL}[(\mathbf{local}\ \vec{x}; c)\ P] = \exists \vec{x}(c \wedge \mathrm{TL}[P])\\
&\mathrm{TL}[\mathbf{next}\ P] = \circ \mathrm{TL}[P]\\
&\mathrm{TL}[\mathbf{unless}\ c\ \mathbf{next}\ P] = c \vee \circ \mathrm{TL}[P]\\
&\mathrm{TL}[!\ P] = \Box \mathrm{TL}[P]
\end{aligned}$$

Modalities $\circ F$ and $\Box F$ represent that F holds *next* and *always*, respectively. We use the *eventual* modality $\Diamond F$ as an abbreviation of $\neg\Box\neg F$.

The following theorem relates the operational view of processes with their logic interpretation.

Theorem 3.2.6 (Logic correspondence [Olarte & Valencia 2008a]). *Let* TL[·] *be as in Definition 3.2.5, P a* utcc *process and $s = c_1.c_2.c_3\ldots$ an infinite sequence of constraints s.t. $P \xrightarrow{s}{}^{*}$. For every constraint d, it holds that:* $\mathrm{TL}[P] \Vdash \Diamond d$ *iff there exists $i \geq 1$ s.t. $c_i \Vdash d$.*

Recall that an observable transition $P \xrightarrow{(c,c')} Q$ is obtained from a finite sequence of internal transitions (rule R_O). We notice that there exist processes that may produce infinitely many internal transitions and as such, they cannot exhibit an observable transition; an example is $(\mathbf{abs}\ x; c(x))\ \mathbf{tell}(c(x+1))$. The utcc processes considered in this paper are *well-terminated*, i.e., they never produce an infinite number of internal transitions during a time unit. Notice also that in the Theorem 3.2.6 the process P is assumed to be able to output a constraint c_i for all time-unit $i \geq 1$. Therefore, P must be a well-terminated process.

Derived Constructs. Let `out` be an uninterpreted predicate. One could attempt at representing the actions of sending and receiving as in a name-passing calculus (say, $k![\vec{e}]$ and $k?(\vec{x})\,.\ P$, resp.) with the utcc processes $\mathbf{tell}(\mathtt{out}(k, \vec{e}))$ and $(\mathbf{abs}\ \vec{x}; \mathtt{out}(k, \vec{x}))\ P$, respectively. Nevertheless, since these processes are not automatically transferred from one time unit to the next one, they will disappear right after the current time unit, even if they do not interact. To cope with this kind of behavior, we shall define versions of $(\mathbf{abs}\ \vec{x}; c)\ P$ and $\mathbf{tell}(c)$ processes that are *persistent in time*. More precisely, we shall use the process $(\underline{\mathbf{wait}}\ \vec{x}; c)\ \mathbf{do}\ P$, which transfers itself from one time unit to the next one until, for some $\vec{t}$, $c[\vec{t}/\vec{x}]$ is entailed by the current store. Intuitively, the process behaves like an input that is active until interacting with an output. When this occurs, the process outputs the constraint $\overline{c}[\vec{t}/\vec{x}]$, as a way of acknowledging the successful read of c. When $|\vec{x}| = 0$, we shall write $\mathbf{whenever}\ c\ \mathbf{do}\ P$ instead of $(\underline{\mathbf{wait}}\ \vec{x}; c)\ \mathbf{do}\ P$. Similarly, we define $\underline{\mathbf{tell}}(c)$ for the

persistent output of c until some process "reads" c. These processes can be expressed in the basic utcc syntax as follows (in all cases, we assume $stop, go \notin \mathrm{fv}(c)$):

$$\underline{\mathbf{tell}}(c) \stackrel{\mathtt{def}}{=} (\mathbf{local}\, go, stop)\,(\mathbf{tell}(\mathtt{out}'(go)) \,||!\, \mathbf{when}\ \mathtt{out}'(go)\ \mathbf{do}\ \mathbf{tell}(c) \,||\, !\,\mathbf{unless}\ \mathtt{out}'(stop)\ \mathbf{next}\,\mathbf{tell}(\mathtt{out}'(go)) \,||\, !\,\mathbf{when}\ \overline{c}\ \mathbf{do}\ !\,\mathbf{tell}(\mathtt{out}'(stop)))$$

$$(\mathbf{wait}\ \vec{x}; c)\ \mathbf{do}\ P \stackrel{\mathtt{def}}{=} (\mathbf{local}\, stop, go)\,(\mathbf{tell}(\mathtt{out}'(go)) \,||!\, \mathbf{unless}\ \mathtt{out}'(stop)\ \mathbf{next}\,\mathbf{tell}(\mathtt{out}'(go)) \,||!\, (\mathbf{abs}\ \vec{x}; c \wedge \mathtt{out}'(go))\,(P \,||!\, \mathbf{tell}(\mathtt{out}'(stop))))$$

$$(\underline{\mathbf{wait}}\ \vec{x}; c)\ \mathbf{do}\ P \stackrel{\mathtt{def}}{=} (\mathbf{wait}\ \vec{x}; c)\ \mathbf{do}\ (P \,||\, \mathbf{tell}(\overline{c}))$$

Notice that once a pair of processes **tell** and **wait** interact, their continuation in the next time unit is a process able to output only a constraint of the form $\exists_x \mathtt{out}'(x)$ (e.g., $\exists_{stop}(\mathtt{out}'(stop))$). We define the following equivalence relation that allows us to abstract from these processes.

Definition 3.2.7 (Observables). *Let $\sim^o$ be the output equivalent relation in Definition 3.2.4. We say that P and Q are observable equivalent, notation $P \sim^{obs} Q$, if $P \,||!\, \mathbf{tell}(\exists_x \mathtt{out}'(x)) \sim^o Q \,||!\, \mathbf{tell}(\exists_x \mathtt{out}'(x))$.*

Using the previous equivalence relation, we can show the following.

Proposition 3.2.8. *Assume that $c(\vec{x})$ is a predicate symbol of arity $|\vec{x}|$.*

1. *If $d \not\Vdash c[\vec{t}/\vec{x}]$ for any $\vec{t}$ then $(\underline{\mathbf{wait}}\ \vec{x}; c)\ \mathbf{do}\ P \stackrel{(d,d)}{\Longrightarrow} (\underline{\mathbf{wait}}\ \vec{x}; c)\ \mathbf{do}\ P$.*
2. *If $P \equiv_u \underline{\mathbf{tell}}(c(\vec{t})) \,||\, (\underline{\mathbf{wait}}\ \vec{x}; c(\vec{x}))\ \mathbf{do}\ \mathbf{next}\, Q$ then $P \Longrightarrow \sim^{obs} Q[\vec{t}/\vec{x}]$.*

3.3 A Declarative Interpretation for Structured Communications

The encoding $[\![\cdot]\!]$ from HVK into utcc is defined in Figure 3.3. Two noteworthy aspects when considering such a translation are *determinacy* and *timed behavior*. Concerning determinacy, it is of uttermost importance to recall that while utcc is a deterministic language, HVK processes may exhibit non-deterministic behavior. Moreover, while HVK is a synchronous language, whereas utcc is asynchronous. Consider, for instance, the HVK process:

$$P = k![\vec{e}]; Q_1 \mid k![\vec{e'}]; Q_2 \mid k?(\vec{x})\,.\, Q_3$$

Process P can have two possible transitions, and evolve into $k![\vec{e'}]; Q_2 \mid\ Q_3[\vec{e}/\vec{x}]$ or into $k![\vec{e}]; Q_1 \mid\ Q_3[\vec{e'}/\vec{x}]$. In both cases, there is an output that cannot interact with the

input $k?(\vec{x})\,.\,Q_3$. In utcc, inputs are represented by abstractions which are persistent during a time unit. As a result, in the encoding of P we shall observe that *both* outputs react with the same input, i.e. that $[\![P]\!] \Longrightarrow [\![Q_3[\vec{e}/\vec{x}]]\!] \,||\, [\![Q_3[\vec{e'}/\vec{x}]]\!]$.

As for timed behavior, it is crucial to observe that while HVK is an untimed calculus, utcc provides constructs for explicit time. In the encoding we shall advocate a timed interpretation of HVK in which all available synchronizations between processes occur at a given time unit, and the continuations of synchronized processes will be executed in the next time unit. This will prove convenient when showing the operational correspondence between both calculi, as we can relate the observable behavior in utcc and the reduction semantics in HVK.

Let us briefly provide some intuitions on $[\![\cdot]\!]$. Consider HVK processes $P =$ **request** $a(k)$ **in** P' and $Q =$ **accept** $a(x)$ **in** Q'. The encoding of P declares a new variable session k and sends it through the channel a by posting the constraint $\texttt{req}(a,k)$. Upon reception of the session key (local variable) generated by $[\![P]\!]$, process $[\![Q]\!]$ adds the constraint $\texttt{acc}(a,k)$ to notify the acceptance of k. They can then synchronize on this constraint, and execute their continuations in the next time unit. The encoding of label selection and branching is similar, and uses constraint $\texttt{sel}(k,l)$ for synchronization. We use the parallel composition $\prod_{1\leq i\leq n}$ **when** $l = l_i$ **do next**$[\![P_i]\!]$ to execute the selected choice. Notice that we do not require a non-deterministic choice since the constraints $l = l_i$ are mutually exclusive. As in [Honda *et al.* 1998], in the encoding of **if** e **then** P **else** Q we assume an evaluation function on expressions. Once e is evaluated, $\downarrow e$ is a *constant* boolean value. The encoding of **def** D **in** P exploits the scheme described in Equation 3.1.

3.3.1 Operational Correspondence.

Here we study an operational correspondence property for our encoding. The differences with respect to (a)synchrony and determinacy discussed above will have a direct influence on the correspondence. Intuitively, the encoding falls short for HVK programs featuring the kind of non-determinism that results from "uneven pairings" between session requesters/providers, label selection/branching, and inputs/outputs as in the example above.

We thus find it convenient to appeal to the type system of HVK to obtain some basic determinacy of the source terms. Roughly speaking, the type discipline in [Honda *et al.* 1998] ensures a correct pairing between actions and co-actions once a session is established. Although the type system guarantees a correct match between (the types of) session requesters and providers, it does not rule out the kind of non-determinism induced by different orders in the pairing of requesters and providers. We shall then require session providers to be always willing to engage into a session. This is, given a channel a, we require that there is at most one **accept** process (possibly replicated) on a that is able to synchronize with every process requesting a session on a. Notice that this requirement is in line with a meaningful class of programs, namely those described by the type discipline developed in [Berger

$$
\begin{array}{ll}
[\textbf{request } a(k) \textbf{ in } P] & = (\textbf{local } k)\,(\underline{\texttt{tell}}(\texttt{req}(a,k)) \,||\, \textbf{whenever } \texttt{acc}(a,k) \textbf{ do next}[P]) \\
[\textbf{accept } a(k) \textbf{ in } P] & = (\underline{\textbf{wait}}\ k; \texttt{req}(a,k))\ \textbf{do}\ (\textbf{tell}(\texttt{acc}(a,k)) \,||\, \textbf{next}[P]) \\
\\
[k![\vec{e}]; P] & = \underline{\texttt{tell}}(\texttt{out}(k,\vec{e})) \,||\, \textbf{whenever } \overline{\texttt{out}(k,\vec{e})} \textbf{ do next}[P] \\
[k?(\vec{x}) \textbf{ in } P] & = (\underline{\textbf{wait}}\ \vec{x}; \texttt{out}(k,\vec{x}))\ \textbf{do next}[P] \\
\\
[k \lhd l; P] & = \underline{\texttt{tell}}(\texttt{sel}(k,l)) \,||\, \textbf{whenever } \overline{\texttt{sel}(k,l)} \textbf{ do next}[P] \\
[k \rhd \{l_1 : P_1 \,||\, \ldots \,||\, l_n : P_n\}] & = (\underline{\textbf{wait}}\ l; \texttt{sel}(k,l))\ \textbf{do} \prod_{1\le i\le n} \textbf{when } l = l_i \textbf{ do next}[P_i] \\
\\
[\textbf{throw } k[k']; P] & = \underline{\texttt{tell}}(\texttt{outk}(k,k')) \,||\, \textbf{whenever } \overline{\texttt{outk}(k,k')} \textbf{ do next}[P] \\
[\textbf{catch } k(k') \textbf{ in } P] & = \underline{\textbf{whenever}}\ \texttt{outk}(k,k')\ \textbf{do next}[P] \\
\\
[\textbf{if } e \textbf{ then } P \textbf{ else } Q\,] & = \textbf{when } e \downarrow \mathrm{tt}\ \textbf{do next}[P] \,||\, \textbf{when } e \downarrow \mathrm{ff}\ \textbf{do next}[Q] \\
[P|Q] & = [P] \,||\, [Q] \\
[\textbf{inact}] & = \textbf{skip} \\
[(\nu u)P] & = (\textbf{local } u)\,[P] \\
[\textbf{def } D \textbf{ in } P] & = \prod_{X_i(x_i k_i)\in D} \mathcal{R}[X_i(x_i k_i)]\widehat{P}
\end{array}
$$

Figure 3.3: Encoding from HVK into utcc. $\mathcal{R}[\cdot]$ and $\widehat{P}$ are defined in Equation 3.1.

et al. 2008, Berger *et al.* 2001].

Before presenting the operational correspondence, we introduce some auxiliary notions.

Definition 3.3.1 (Processes in normal form). *We say that a* HVK *process P is in* normal form *if takes the form* **inact** *or* **def** D **in** $\nu\vec{u}(Q_1 \mid \cdots \mid Q_n)$ *where neither the operators "ν" and "$|$" nor process variables occur in the top level of $Q_1, \ldots, Q_n$.*

The following proposition states that given a process P we can find a process P' in normal form, such that: either P' is structurally congruent to P, or it results from replacing the process variables at the top level of P with their corresponding definition (using rule Def).

Proposition 3.3.2. *For all* HVK *process P there exists P' in normal form s.t. $P \longrightarrow_h^* \equiv_h P'$ only using the rules* Def *and* Str *in Figure 3.1.*

Proof. Let P be a process of the form **def** D **in** Q where there are no procedure definitions in Q. By repeated applications of the rule Def, we can show that $P \longrightarrow_h^* P'$ where P' does not have occurrences of processes variables in the top level. Then, we use the rules of the structural congruence to move the local variables to the outermost position and find $P'' \equiv_h P'$ in the desired normal form. □

Notice that the rules of the operational semantics of HVK are given for pairs of processes that can interact with each other. We shall refer to each of those pairs as a *redex*.

Definition 3.3.3 (Redex). *A* redex *is a pair of complementary processes composed in parallel as in:*

$$(1)\textbf{request } a(k) \textbf{ in } P \mid \textbf{accept } a(k) \textbf{ in } Q$$
$$(2)k![\vec{e}]; P \mid k?(\vec{x}) \textbf{ in } Q$$
$$(3)\textbf{throw } k[k']; P \mid \textbf{catch } k(k') \textbf{ in } Q$$
$$(4)k \lhd l; P \mid k \rhd \left\{ l_1 : P_1 \Big|\Big| \cdots \Big|\Big| l_n : P_n \right\}$$

Notice that a redex in HVK synchronizes and reduces in a single transition as in $(k![\vec{e}]; P) \mid (k?(\vec{x}) \textbf{ in } Q) \longrightarrow_h P \mid Q[\vec{e}/\vec{x}]$. Nevertheless, in utcc, the encoding of the processes above requires several internal transitions for adding the constraint $\mathtt{out}(k, \vec{e})$ to the current store, and for "reading" that constraint by means of $(\underline{\textbf{wait}}\ \vec{x}; \mathtt{out}(k, \vec{x}))$ **do next**$[Q]$ to later execute **next**$[Q[\vec{e}/\vec{x}]]$. We shall then establish the operational correspondence between an observable transition of utcc (obtained from a finite number of internal transitions) and the following subset of reduction relations over HVK processes:

Definition 3.3.4 (Outermost Reductions). *Let* $P \equiv_h$ **def** D **in** $\nu\vec{x}(Q_1 \mid \cdots \mid Q_n)$ *be an* HVK *program in normal form. We define the* outermost reduction relation $P \Longrightarrow_h P'$ *as the maximal sequence of reductions* $P \longrightarrow_h^* P' \equiv_h$ **def** D **in** $\nu\vec{x'}(Q'_1 \mid \cdots \mid Q'_n)$ *such that for every* $i \in \{1, ..n\}$*, either*

1. $Q_i =$ **if** e **then** R_1 **else** $R_2 \longrightarrow_h R_{1/2} = Q'_i$;
2. *for some* $j \in \{1, ..n\}$, $Q_i|Q_j$ *is a redex such that* $Q_i|Q_j \longrightarrow_h \nu\vec{y}(Q'_i|Q'_j)$, *with* $\vec{y} \subseteq \vec{x'}$;
3. *there is no* $k \in \{1, ..n\}$ *such that* $Q_i \mid Q_k$ *is a redex and* $Q_i \equiv_h Q'_i$.

One may argue that the above-presented definition may rule out some possible reductions in HVK. Returning to the concerns about determinacy, an outermost reduction filters out cases where there are more than one possible reduction for a set of parallel processes (i.e.: the parallel composition of two outputs and one input with the same session key). The use of outermost reductions gives us a subset of possible reductions in HVK that keeps synchronous processes and discard processes that are not going to interact in any way (recall that in the typing discipline of HVK

the composition of an input and an output with the same session key will consume the channel used; hence, every other process sending information over the same session will not have any complementary process to synchronize with).

In the sequel we shall thus consider only HVK processes P where for $n \geq 1$, if $P \equiv_h P_1 \Longrightarrow_h P_2 \Longrightarrow_h \cdots \Longrightarrow_h P_n$ and $P \equiv_h P'_1 \Longrightarrow_h P'_2 \Longrightarrow_h \cdots \Longrightarrow_h P'_n$ then $P_i \equiv_h P'_i$ for all $i \in \{1, .., n\}$, i.e., P is a *deterministic* process.

Theorem 3.3.5 (Operational Correspondence). *Let P, Q be deterministic* HVK *processes in normal form and R,S be* utcc *processes. It holds:*
1) Soundness*: If $P \Longrightarrow_h Q$ then, for some R, $[P] \Longrightarrow R \sim^{obs} [Q]$;*
2) Completeness*: If $[P] \Longrightarrow S$ then, for some Q, $P \Longrightarrow_h Q$ and $[Q] \sim^{obs} S$.*

Proof. Assume that $P \equiv_h \textbf{def}\ D\ \textbf{in}\ \nu\vec{x}(Q_1 \mid \cdots \mid Q_n)$ and $Q \equiv_h \textbf{def}\ D\ \textbf{in}\ \nu\vec{x'}(Q'_1 \mid \cdots \mid Q'_n)$.

1. *Soundness*. Since $P \Longrightarrow_h Q$ there must exist a sequence of derivations of the form $P \equiv_h P_1 \longrightarrow_h P_2 \longrightarrow_h \ldots \longrightarrow_h P_n \equiv_h Q$. The proof proceeds by induction on the length of this derivation, with a case analysis on the last applied rule. We then have the following cases:

   (a) **Using the rule IF1**. It must be the case that there exists $Q_i \equiv_h \textbf{if}\ e\ \textbf{then}\ R_1\ \textbf{else}\ R_2$ and $Q_i \longrightarrow_h R_1 \equiv_h Q'_i$ and $e \downarrow \mathrm{tt}$. One can easily show that $\textbf{when}\ e \downarrow \mathrm{tt}\ \textbf{do next}\ [Q'_i] \Longrightarrow [Q'_i]$.

   (b) **Using the rule IF2** Similarly as for IF1.

   (c) **Using the rule LINK**. It must be the case that there exist i, j such that $Q_i \equiv_h \textbf{request}\ a(k)\ \textbf{in}\ Q'_i$ and $Q_j \equiv_h \textbf{accept}\ a(x)\ \textbf{in}\ Q'_j$ and then $Q_i \mid Q_j \longrightarrow_h (\nu k)(Q'_i \mid Q'_j)$. We then have a derivation

$$
\begin{aligned}
[Q_i] \,||\, [Q_k] &\longrightarrow^* (\textbf{local}\ k; c)\,(R'_i \,||\, \textbf{whenever}\ \mathtt{acc}(a,k)\ \textbf{do next}\,[Q'_i] \,||\\
&\qquad (\underline{\textbf{wait}}\ k'; \mathtt{req}(a,k'))\ \textbf{do}\ (\textbf{tell}(\mathtt{acc}(a,k')) \,||\, \textbf{next}\,([Q'_j]))\\
&\longrightarrow^* (\textbf{local}\ k; c')\,(R'_i \,||\, \textbf{whenever}\ \mathtt{acc}(a,k)\ \textbf{do next}\,[Q'_i] \,||\\
&\qquad R'_j \,||\, \textbf{tell}(\mathtt{acc}(a,k)) \,||\, \textbf{next}\,([Q'_j[k/k']])\\
&\longrightarrow^* (\textbf{local}\ k; c'')\,(R'_i \,||\, R'_j \,||\, \textbf{next}\,[Q'_i] \,||\, \textbf{next}\,([Q'_j[k/k']]) \not\longrightarrow
\end{aligned}
$$

   where $c = \mathtt{req}(a,k)$, $c' = c \wedge \overline{\mathtt{req}(a,k)}$, $c'' = c' \wedge \mathtt{acc}(a,k) \wedge \overline{\mathtt{acc}(a,k)}$ and R'_i, R'_j are the processes resulting after the interaction of the processes in

the parallel composition $\underline{\mathbf{tell}}(\mathtt{req}(a,k)) \,||(\underline{\mathbf{wait}}\ k'; \mathtt{req}(a,k'))\ \mathbf{do}\ \cdots$, i.e.:

$$R'_i \equiv_u (\mathbf{local}\ go, stop; \mathtt{out}'(go) \wedge \mathtt{out}'(stop) \wedge c(\vec{t}))$$
$$\mathbf{next}\,!\,\mathbf{unless}\ \ \mathtt{out}'(stop)\ \mathbf{next}\,\mathbf{tell}(\mathtt{out}'(go)) \Big|\Big| \mathbf{next}\,!\,\mathbf{tell}(\mathtt{out}'(stop))$$

$$R'_j \equiv_u (\mathbf{local}\ stop', go'; \mathtt{out}'(go') \wedge \overline{c}(\vec{t}) \wedge \mathtt{out}'(stop'))\,\mathbf{next}\,!\,\mathbf{tell}(\mathtt{out}'(stop'))$$
$$\Big|\Big| \mathbf{next}\,!\,\mathbf{unless}\ \ \mathtt{out}'(stop')\ \mathbf{next}\,\mathbf{tell}(\mathtt{out}'(go'))$$
$$\Big|\Big| (\lambda\ \vec{x}; \mathrm{c} \wedge \mathtt{out}'(\mathrm{go}') \wedge \tilde{\mathrm{x}} \neq \tilde{\mathrm{t}})\ (Q \Big|\Big| \mathbf{tell}(\overline{c}(\vec{t})) \Big|\Big| !\,\mathbf{tell}(\mathtt{out}'(stop'))$$
$$\Big|\Big| \mathbf{next}\,!\,(\lambda\ \vec{x}; \mathrm{c} \wedge \mathtt{out}'(\mathrm{go}'))\ (Q \Big|\Big| \mathbf{tell}(\overline{c}(\vec{t})) \Big|\Big| !\,\mathbf{tell}(\mathtt{out}'(stop'))$$

We notice that $R'_i \,||\, R'_j \not\longrightarrow$ and it is a process that can only output the constraint $\mathtt{out}'(x)$ where x is a local variable. By appealing to Proposition 3.2.8 we conclude $[\![Q_i]\!] || [\![Q_j]\!] \Longrightarrow\!\!\sim^{obs} (\mathbf{local}\ k)\,([\![Q'_i]\!] || [\![Q'_j]\!])$.

(d) The cases using the rules Label and Pass can be proven similarly as the case for link.

2. *Completeness*. Given the encoding and the structure of P, we have a utcc process $R = [\![P]\!]$ s.t.

$$R \equiv_u (\mathbf{local}\ \vec{x})\,([\![Q_1]\!] \Big|\Big| \ldots \Big|\Big| [\![Q_n]\!])\,.$$

Let $R_i = [\![Q_i]\!]$ for $1 \leq i \leq n$. By an analysis on the structure of R, if $R_i \longrightarrow R'_i$ then it must be the case that either (a) $R_i = \mathbf{when}\ e\ \mathbf{do}\ \mathbf{next}[\![Q'_i]\!]$ and $R'_i = \mathbf{next}[\![Q'_i]\!]$ or (b) $\langle R_i, c\rangle \longrightarrow \langle R'_i, c \wedge d\rangle$ where d is a constraint of the form $\mathtt{req}(\cdot)$, $\mathtt{sel}(\cdot)$, $\mathtt{out}(\cdot)$, or $\mathtt{outk}(\cdot)$. In both cases we shall show that there exists a R''_i such that $R_i \longrightarrow^* R''_i \not\longrightarrow$ such that $Q_i \longrightarrow_h Q'_i$ and $R''_i = \mathbf{next}[\![Q'_i]\!]$.

(a) Assume that $R_i = \mathbf{when}\ e \downarrow \mathrm{tt}\ \mathbf{do}\ \mathbf{next}[\![Q'_i]\!]$ for some Q'_i. Then it must be the case that $Q_i = \mathbf{if}\ e\ \mathbf{then}\ Q'_i\ \mathbf{else}\ Q''_i$. If $e \downarrow \mathrm{tt}$ we then have $R''_i = \mathbf{next}[\![Q'_i]\!]$. The case when $e \downarrow \mathrm{ff}$ is similar by considering $R_i = \mathbf{when}\ e \downarrow \mathrm{ff}\ \mathbf{do}\ Q'_i$.

(b) Assume now that $\langle R_i, c\rangle \longrightarrow \langle R'_i, c \wedge d\rangle$ where d is of the form $\mathtt{req}(\cdot)$, $\mathtt{sel}(\cdot)$, $\mathtt{out}(\cdot)$ or $\mathtt{outk}(\cdot)$. We proceed by case analysis of the constraint d. Let us consider only the case $d = \exists_k(\mathtt{req}(a,k))$; the cases in which d takes the form $\mathtt{sel}(\cdot)$, $\mathtt{out}(\cdot)$, or $\mathtt{outk}(\cdot)$ are handled similarly. If $d = \exists_k(\mathtt{req}(a,k))$ for some a, then we must have that $Q_i \equiv_h \mathbf{request}\ a(k)\ \mathbf{in}\ Q'_i$ for some i. If there exists j such that $Q_j \equiv_h \mathbf{accept}\ a(x)\ \mathbf{in}\ Q'_j$, one can show a derivation similar to the case of the rule Link in soundness to prove that $R_i \,||\, R_j \longrightarrow^* \!\!\sim^o (\mathbf{local}\ k)\,(\mathbf{next}[\![Q'_i]\!] || \mathbf{next}[\![Q'_j]\!])$. If there is no Q_j such that $Q_i \,| Q_j$ forms a redex, then one can show by using (1) in Proposition 3.2.8 that $R_i \Longrightarrow\!\!\sim^{obs} R_i$.

□

$$
\begin{array}{rl}
P ::= \textbf{request}\ a(k)\ \textbf{during}\ m\ \textbf{in}\ P & \text{(Timed Session Request)} \\
\mid \textbf{accept}\ a(k)\ \textbf{given}\ c\ \textbf{in}\ P & \text{(Declarative Session Acceptance)} \\
\mid \cdots & \text{(the other constructs, as in Def. 3.2.1)} \\
\mid \textbf{kill}\ c_k & \text{(Session Abortion)}
\end{array}
$$

Figure 3.4: HVK$^{\mathsf{T}}$: Syntax of the language

3.4 A Timed Extension of HVK

We now propose an extension to HVK in which a bundled treatment of time is explicit and session closure is considered. More precisely, the HVK$^{\mathsf{T}}$ language arises as the extension of HVK processes (Def. 3.2.1) with refined constructs for session request and acceptance, as well as with a construct for session abortion:

Definition 3.4.1 (A timed language for sessions). HVK$^{\mathsf{T}}$ *processes are given by the grammar in figure 3.4:*

The intuition behind these three operators is the following: **request** $a(k)$ **during** m **in** P will request a session k over the service name a during m time units. Its dual construct is **accept** $a(k)$ **given** c **in** P: it will grant the session key k when requested over the service name a provided by a session and a successful check over the constraint c. Notice that c stands for a precondition for agreement between session request and acceptance. In c, the duration m of the corresponding session key k can be referenced by means of the variable dur_k. In the encoding we syntactically replace it by the variable corresponding to m. Finally, **kill** c_k will remove c_k from the valid set of sessions.

Adapting the encoding in Figure 3.3 to consider HVK$^{\mathsf{T}}$ processes is remarkably simple (see Figure 3.5). Indeed, modifications to the encoding of session request and acceptance are straightforward. The most evident change is the addition of the parameter m within the constraint $\mathtt{req}(a, k, m)$. The duration of the requested session is suitably represented as a bounded replication of the process defining the activation of the session k represented as the constraint $\mathtt{act}(k)$. The execution of the continuation $[P]$ is guarded by the constraint $\mathtt{act}(k)$ (i.e. P can be executed only when the session k is valid). Thus, in the encoding we use the function $\mathcal{G}_d(P)$ to denote the process behaving as P when the constraint d can be entailed from the current store, doing nothing otherwise. More precisely:

$$
\begin{aligned}
[\textbf{request } a(k) \textbf{ during } m \textbf{ in } P] = (\textbf{local } k)\, \underline{\texttt{tell}}(\texttt{req}(a,k,m)) \Big|\Big| \\
\textbf{whenever } \texttt{acc}(a,k) \textbf{ do next} (\\
\textbf{tell}(\texttt{act}(k)) \Big|\Big| \\
\mathcal{G}_{\texttt{act}(k)}([P]) \Big|\Big| \\
!_{[m]}\textbf{unless } \texttt{kill}(k) \textbf{ next tell}(\texttt{act}(k)))
\end{aligned}
$$

$$
\begin{aligned}
[\textbf{accept } a(k) \textbf{ given } c \textbf{ in } P] = (\underline{\textbf{wait}}\ k; \texttt{req}(a,k,m) \wedge c[m/dur_k])\ \textbf{do} \\
(\textbf{tell}(\texttt{acc}(a,k)) \Big|\Big| \textbf{next}\, \mathcal{G}_{\texttt{act}(k)}([P])) \\
[\textbf{kill } k] = !\, \textbf{tell}(\texttt{kill}(k))
\end{aligned}
$$

Figure 3.5: Encoding of HVK$^{\mathsf{T}}$. $\mathcal{G}_d(P)$ is in Definition 3.4.2.

Definition 3.4.2. *Let $\mathcal{G} : \mathcal{C} \to Procs \to Procs$ be defined as:*

$$
\begin{aligned}
&\mathcal{G}_d(\textbf{skip}) && = \textbf{skip} \\
&\mathcal{G}_d(P_1 \Big|\Big| P_2) && = \mathcal{G}_d(P_1) \Big|\Big| \mathcal{G}_d(P_2) \\
&\mathcal{G}_d(\textbf{tell}(c)) && = \textbf{when } d \textbf{ do tell}(c) \\
&\mathcal{G}_d(!\, Q) && = !\, \mathcal{G}_d(Q) \\
&\mathcal{G}_d(\textbf{next } Q) && = \textbf{when } d \textbf{ do next}\, \mathcal{G}_d(Q) \\
&\mathcal{G}_d((\textbf{abs } \vec{x}; c)\, Q) && = (\textbf{abs } \vec{x}; c)\, \mathcal{G}_d(Q) \quad \textit{if } \vec{x} \notin \mathrm{fv}(d) \\
&\mathcal{G}_d(\textbf{unless } c \textbf{ next } Q) && = \textbf{when } d \textbf{ do unless } c \textbf{ next}\, \mathcal{G}_d(Q) \\
&\mathcal{G}_d((\textbf{local } \vec{x}; c)\, Q) && = (\textbf{local } \vec{x}; c)\, \mathcal{G}_d(Q) \quad \textit{if } \vec{x} \notin \mathrm{fv}(d)
\end{aligned}
$$

On the side of session acceptance, the main novelty is the introduction of $c[m/dur_k]$. As explained before, we syntactically replace the variable dur_k by the corresponding duration of the session m. This is a generic way to represent the agreement that should exist between a service provider and a client; for instance, it could be the case that the client is requesting a session longer than what the service provider can or want to grant.

3.4.1 Case Study: Electronic booking

Here we present an example that makes use of the constructs introduced in HVK$^{\mathsf{T}}$.

Let us consider an electronic booking scenario. On one side, consider a company AC which offers flights directly from its website. On the other side, there is a customer looking for the best offers. In this scenario, the customer establishes a timed session with AC and asks for a flight proposal given a set of constraints (dates allowed, destination, etc.). After receiving an offer from AC, the customer can refine

$$Customer = \textbf{request}\ ob(k)\ \textbf{during}\ m\ \textbf{in}\ (k![bookingdata]; Select(k))$$

$$Select(k) = k?(offer)\,\textbf{in}\,(\textbf{if}\,(offer.price \leq 1500)\ \textbf{then}\ k \lhd Contract;\ \textbf{else}\ Select(k)\,)$$

$$AC = \textbf{accept}\ ob(k)\ \textbf{given}\ dur_k \leq \text{MAX\_TIME}\ \textbf{in}\ (k?(bookingData)\,\textbf{in}\ (\nu u)k![u]; k \rhd \left\{ Contract : \overline{Accept} \,\middle|\middle|\, Reject : \textbf{kill}\,k \right\})$$

Figure 3.6: Online booking example with two agents.

the selection further (e.g. by checking that the prices are below a given threshold) and loops until finding a suitable option, that he will accept by starting the booking phase. One possible HVK$^{\mathsf{T}}$ specification of this scenario is described in Figure 3.6.

In a second stage, the customer uses an online broker to mediate between him and a set of airlines acting as service providers. Let n be the number of service providers, and consider two vectors of fixed length: *Offers*, which contains the list $[Offers_0, \ldots, Offers_i, \ldots, Offers_n]$ of offers received by a customer, and SP, which contains the list of trusted services. First, the customer establishes a session with the broker for a given period m; later on, he/she starts requesting for a flight by providing the details of his/her trip to the broker. On the other side, the broker will look into his pool of trusted service providers for the ones that can supply flights that suit the customer's requirements. All possible offers are transferred back to the customer, who will invoke a local procedure Sel (not specified here) that selects one of the offers by performing an output on name a. Once an offer is selected, the broker will allow a final interaction between the customer and the selected service. He does so by delegating to the customer the session key used previously between him and the chosen service provider. Finally, the broker proceeds to cancel all those sessions concerning the discarded services. An HVK$^{\mathsf{T}}$ specification of this scenario is given in Figure 3.7 where, for the sake of readability, processes denoting post-processing activities are abstracted from the specification.

A notable advantage in using HVK$^{\mathsf{T}}$ as a modeling language is the possibility of exploiting timed constructs in the specification of service enactment and service cancellation. In the above scenario it is possible to see how HVK$^{\mathsf{T}}$ allows (i) to effectively take explicit account on the maximal times accepted by the customer: the composition of nested services can take different speeds but the service broker will ensure that customers with low speeds are ruled out of the communication; and (ii) to have a more efficient use of the available resources: since there is not need to maintain interactions with discarded services, the service broker will free those resources by sending kill signals.

$$
\begin{aligned}
\text{Customer} = \ & \textbf{request}\ ob(k)\ \textbf{during}\ m\ \textbf{in}\ (k![bookingdata];\\
& k?(n)\ \textbf{in}\ (\\
& \textstyle\prod_{i\in n}(k?(Offers_i)\ \textbf{in}\ (\\
& Sel(Offers); a?(x)\ \textbf{in}\ k![x];\\
& \ \textbf{catch}\ k(k')\ \textbf{in}\\
& \ k'![PaymentDetails]; \textbf{inact}))))
\end{aligned}
$$

$$
\begin{aligned}
\text{SP} = \ & \textbf{accept}\ SP_i(k_i')\ \textbf{given}\ N \leq 300ms\ \textbf{in}\ (\\
& k_i'?(bookingData)\ \textbf{in}\\
& \ k_i'![offer];\\
& \ k_i'?(paymentDetails)\ \textbf{in}\ \textbf{inact})
\end{aligned}
$$

$$
\begin{aligned}
\text{Broker} = \ & \textbf{accept}\ ob(k)\ \textbf{given}\ m \leq 500ms\ \textbf{in}\ (\\
& \ k?(bookingData)\ \textbf{in}\ k![|SP|];\\
& \ (\nu u)\ \textstyle\prod_{i\in|SP|}(\textbf{request}\ SP_i(k_i')\ \textbf{during}\ N\ \textbf{in}\\
& \quad k_i'![bookingData];\\
& \quad k_i'?(offer_i)\ \textbf{in}\ (u![offer_i]; \textbf{inact}\,||\,S(u,k)))\\
& \ k?(y)\ \textbf{in}\ \textbf{def}\ X(Offers, k_1', \ldots, k_n') = P\ \textbf{in}\\
& \quad \textstyle\prod_{i\in|SP|}(\textbf{if}\ (y = offers_i)\ \textbf{then}\ (\textbf{throw}\ k[k_i']; PostProc)\ \textbf{else}\\
& \qquad \textbf{kill}\ k_i'\,||\,P(X - \{offers_i, k_i'\})))
\end{aligned}
$$

$$
\text{S(u,k)} = \ \textstyle\prod_{i\in|SP|}(u?(offer_i)\ \textbf{in}\ \textbf{inact}\,||\,k![offer_i]; \textbf{inact})
$$

Figure 3.7: Online booking example with online broker.

3.4.2 Exploiting the Logic Correspondence

To exploit the logic correspondence we can draw inspiration from the *constraint templates* put forward in [Pesic & van der Aalst 2006], a set of LTL formulas that represent desirable/undesirable situations in service management. Such templates are divided in three types: *existence constraints*, that specify the number of executions of an activity; *relation constraints*, that define the relation between two activities to be present in the system; and *negation constraints*, which are essentially the negated versions of relation constraints.

By appealing to Theorem 3.2.6, our framework allows for the verification of existence and relation constraints over HVK^{T} programs. Assume a HVK^{T} program P and let $F = \text{TL}[\![P]\!]$ (i.e., the FLTL formula associated to the utcc representation of P). For existence constraints, assume that P defines a service accepting requests on channel a. If the service is eventually active, then it must be the case that $F \Vdash \Diamond\exists_k(\mathtt{acc}(a,k))$ (recall that the encoding of **accept** adds the constraint $\mathtt{acc}(a,k)$ when the session k is accepted). A slight modification to the encoding of **accept** would allow us to

take into account the number of accepted sessions and then support the verification of properties such as $F \Vdash \Diamond(N_{sessions}(a) = N)$, informally meaning that the service a has accepted N sessions. This kind of formulas correspond to the existence constraints in [Pesic & van der Aalst 2006, Figure 3.1.a–3.1.c]. Furthermore, making use of the guards associated to ask statements, we can verify relation constraints as eventual consequences over the system. Take for instance the specification in Figure 3.6. Let $\overline{Accept}$ be a process that outputs "ok" through a session h. We then may verify the formula $F \Vdash \exists_u(u.price < 1.500 \Rightarrow \mathtt{out}(h, ok))$. This is a responded existence constraint describing how the presence of an offer with price less or equal than 1.500 would lead to an acceptance state.

3.5 Concluding Remarks

We have argued for a timed CCP language as a suitable foundation for analyzing structured communications. We have presented an encoding of the language for structured communication in [Honda *et al.* 1998] into utcc, as well as an extension of such a language that considers explicitly elements of partial information and session duration. To the best of our knowledge, a unified framework where behavioral and declarative techniques converge for the analysis of structured communications has not been proposed before.

Languages for structured communication and CCP process calculi are conceptually very different. We have dealt with some of these differences (notably, determinacy) when stating an operational correspondence property for the declarative interpretation of HVK processes. We believe there are at least two ways of achieving more satisfactory notions of operational correspondence. The first one involves considering extensions of utcc with (forms of) non-determinism. This would allow to capture some scenarios of session establishment in which the operational correspondence presented here falls short. The main consequence of adding non-determinism to utcc is that the correspondence with FLTL as stated in Theorem 3.2.6 would not longer hold. This is mainly because non-deterministic choices cannot be faithfully represented as logical disjunctions (see, e.g., [Nielsen *et al.* 2002]). While a non-deterministic extension to tcc with a tight connection with temporal logic has been developed (**ntcc** [Nielsen *et al.* 2002]), it does not provide for representations of mobile links. Exploring whether there exists a CCP language between **ntcc** and utcc combining both non-determinism and mobility while providing logic-based reasoning techniques is interesting on its own and appears challenging. The second approach consists in defining a type system for HVK and HVK$^{\mathsf{T}}$ processes better suited to the nature of utcc processes. This would imply enriching the original type system in [Honda *et al.* 1998] with e.g., stronger typing rules for dealing with session establishment. The definition of such a type system is delicate and needs care, as one would not like to rule out too many processes as a result of too stringent typing rules. An advantage of a type system "tuned" in this way is that one could aim at obtaining a correspondence between well-typed processes and logic formulas, similarly as the

given by Theorem 3.2.6. In these lines, plans for future work include the investigation of effective mechanisms for the seamless integration of new type disciplines and reasoning techniques based on temporal logic within the elegant framework provided by (timed) CCP languages.

The timed extension to HVK presented here includes notions of time that involve only session engagement processes. A further extension could involve the inclusion of time constraints over input/output actions. Such an extension might be useful to realistically specify scenarios in which factors such as, e.g, network traffic and long-lived transactions, prevent interactions between services from occurring instantaneously. Properties of interest in this case could include, for instance, the guarantee that a given interaction has been fired at a valid time, or that the nested composition of services does not violate a certain time frame. We plan to explore case studies of structured communications involving this kind of timed behavior, and extend/adjust HVK^{T} accordingly.

Acknowledgments. We are grateful to Marco Carbone and Thomas Hildebrandt for insightful discussions on the topics of this paper. We also grateful to Roberto Zunino who provided useful remarks on a previous version of this document.

Chapter 4

Types for Security and Mobility in Universal CCP

Abstract: The fundamental primitives of Concurrent Constraint Programming (CCP), *tell* and *ask*, respectively adds knowledge to and infers knowledge from a shared constraint store. These features, and the elegant use of the constraint system to represent the abilities of attackers, make concurrent constraint programming and timed CCP (tcc) interesting candidates for modeling and reasoning about security protocols. However, they lack primitives for the communication of secrets (or local names as in the π-calculus) between agents. The recently proposed *universal* tcc (utcc) introduces a universally quantified ask operation that makes it possible to infer knowledge which is local to other agents. However, it allows agents to guess knowledge even if it is encrypted or communicated on secret channels, simply by quantifying over both the encryption key (or channel) and the message simultaneously. We present a secure utcc (utcc$_s$) based on: (i) a simple type system for constraints allowing to distinguish between restricted (secure) and non-restricted (universally quantifiable) variables in constraints, and (ii) a generalization of the universally quantified ask operation to allow the assumption of local knowledge. We illustrate the use of the utcc$_s$ calculus with examples on communication of local names (as in the π-calculus) and for giving semantics to secure pattern matching in a prototypical security language.

Contents

4.1 Introduction

A number of variants of process calculi and logical approaches have been proposed for the analysis of security protocols, including [Abadi & Gordon 1999, Crazzolara & Winskel 2001, Corin & Etalle 2002, Fiore & Abadi 2001, Blanchet 2001, Miller 2003, Buchholtz *et al.* 2004, Olarte & Valencia 2008a]. The approaches have generally two features in common: The first is the use of some kind of logical inference/pattern matching/unification to represent the ability of attackers and principals to infer what has been communicated, and from that knowledge construct new messages. The second is a way of representing and communicating local knowledge (such as keys or nonces in security protocols).

The combination of these two features calls for some means to control the ability to infer knowledge which is supposed to be inaccessible, e.g. a message encrypted by a key unknown to the attacker or the key itself. Typically, this takes the form of a restriction on the rules for inference of knowledge/pattern matching, designed particularly for the considered setting of security protocols. Sometimes the restriction is enforced by the language, as e.g. in [Buchholtz *et al.* 2004], however in many cases the restriction must be maintained in the specification of the attacker and the protocol under analysis.

In the present paper we propose a more general solution to representing this kind of restriction. Even though we believe that the solution is broadly applicable, in this paper we focus on the setting of concurrent constraint programming (CCP). This is due to the fact that our work was directly triggered by the interesting recent proposal of the calculus of *universal* timed concurrent constraint programming (utcc)[Olarte & Valencia 2008a], which extends timed concurrent constraint programming [Saraswat *et al.* 1994] to include a universally quantified abstraction (ask) operation. Intuitively, the new operation added in utcc, written $(\lambda\ \vec{x}; c)\ P$, spans a copy of the residual process $P[\vec{t}/\vec{x}]$ for all possible inferences of $c[\vec{t}/\vec{x}]$. This adds the ability to extend the scope of local knowledge which is not possible in CCP [López *et al.* 2006]. In particular it was illustrated in [Olarte & Valencia 2008a] how to model a notion of *link mobility* as found in the pi-calculus and to use the universal abstraction operator for communication of messages in security protocols.

However, the universal quantification in utcc is completely unrestricted. This means that in the proposed representations of link mobility and security protocols in utcc, every agent may guess channel names and encrypted values by universal quantification. It is thus necessary to enforce a restriction on the allowed processes to make sure that this is not possible.

As a general solution for making exactly such restrictions, we propose a simple type system for constraints used as patterns in abstractions, which essentially allow to distinguish between universally abstractable and secure variables in predicates. We also propose a novel notion of *abstraction under local knowledge*, which gives a general way to model that a process (principal) knows a key and can use it to decrypt a message encrypted with this key without revealing the key.

We exemplify the type system on πcalculus-like mobility of local names and

for giving semantics to a novel security protocol language called Security Protocol Concurrent Constraint Programming language (SPCCP), combining the best features of the the Security CCP (SCCP) language proposed by Olarte and Valencia [Olarte & Valencia 2008a] and the Security Protocol Language (SPL) by Crazzolara and Winskel [Crazzolara & Winskel 2001].

The foregoing document is divided as follows: Section 4.3 introduces the type system for utcc the new abstraction rule over local knowledge, as well as termination and subject-reduction results over the type system proposed. In Section 4.4 we give more details on the use of the utcc with secure patterns. Finally, concluding remarks and future work are described in Section 4.5.

4.2 Preliminaries

This section provides the interested reader the main concepts of Temporal Concurrent Constraint Programming (tcc) and its universal extension (utcc), following the presentation of [Olarte & Valencia 2008a].

In CCP-based calculi all the (partial) information is *monotonically* accumulated in a so-called *store*. The store keeps the knowledge about the system in terms of *constraints*, or statements defining the possible values a variable can take (e.g., $x + y \geq 42$). Concurrent agents (i.e., processes) that are part of the system interact with each other using the store as a shared communication medium. They have two basic capabilities over the store, represented by *tell* and *ask* operations. While the former *adds* a piece of information about the system, the latter *queries* the store to determine if some piece of information can be inferred from its current content. Tell operations can act concurrently refining the information in the store while asks can serve as a general synchronization mechanism, that will be blocked if there is not enough information into the store to answer its query.

A fundamental notion in CCP-based calculi is that of a *constraint system*. Basically, a constraint system provides a signature from which syntactically denotable objects in the language called *constraints* can be constructed, and an entailment relation ($\Vdash$) specifying interdependencies among such constraints. More precisely,

Definition 4.2.1 (Constraint System). *A constraint system is a pair* $CS = (\Sigma, \Delta)$ *where* Σ *is a signature of function (*F*) and predicate (*P*) symbols, and* Δ *is a decidable theory over* Σ *(i.e., a decidable set of sentences over* Σ *with at least one model). The underlying language* $\mathcal{L}$ *of* (Σ, Δ) *contains the symbols* $\neg, \wedge, \Rightarrow, \exists$ *denoting logical negation, conjunction, implication, existential quantification.* Constants, *such as* true *and* false *denote the usual always true and always false values, respectively.* Constraints, *denoted by* $c, d, \ldots$ *are first-order formulae over* $\mathcal{L}$*. We say that* c entails d *in* Δ*, written* $c \Vdash_\Delta d$ *(or just* $c \Vdash d$ *when no confusion arises), if* $c \Rightarrow d$ *is true in all models of* Δ*. For operational reasons we shall require* $\Vdash$ *to be decidable.*

tcc arises as the extension of CCP for timed-systems: Including the notion of discrete time intervals (time units), a computation can be described as the interaction

of a tcc process with the environment: At the instant i a tcc process P receives the store c as an initial stimulus, and when it reaches a quiescent point, it outputs d as the resulting constraint store with a residual process Q that will be executed in the instant $i+1$. Here it is where one of the most important differences between ccp and tcc resides, as whilst the refinement of c during the execution of P at i is monotonic, d is not necessarily a refinement of c (that is, constraints can be forgotten).

Definition 4.2.2 (tcc **process syntax).** *Processes $P, Q, \ldots \in Proc$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by the following syntax.*

$$
\begin{array}{rcl}
P, Q \ldots & ::= & \mathbf{skip} \\
& | & \mathbf{tell}\,(\mathrm{c}) \\
& | & \mathbf{when}\ \mathrm{c}\ \mathbf{do}\ P \\
& | & P \,||\, Q \\
& | & (\mathbf{local}\ \vec{x}; c)P \\
& | & \mathbf{next}\,(P) \\
& | & \mathbf{unless}\ c\ \mathbf{next}\,(P) \\
& | & !P
\end{array}
$$

Intuitively, the process **skip** does nothing, **tell** (c) adds a new constraint c into the store, while **when** c **do** P asks if c is present into the store in order to execute P. (**local** $\vec{x}$; c)P binds a set of variables $\vec{x}$ in P by defining their existence under the constraint c. The operators associated with time allow the process to go one time unit in the future (**next** (P)) or to define time-outs: if at the current time unit it is not possible to entail the constraint c then the process **unless** c **next** P will execute P at the next time unit. We will often use **next**^{n} (P) as a shorter version of **next** (**next** (. . . **next** (P))) n-times. Finally, $P \,||\, Q$ denotes the usual parallel execution and $!P$ denotes timed replication; that is, $!P = P \,||\, \mathbf{next}\,(!P)$ executes P at the current time and replicates its behaviour over the next time period.

utcc [Olarte & Valencia 2008a] is an extension of the tcc calculus with a general *ask* defining a model of synchronization. While in tcc an ask **when** c **do** P is blocked if there is not enough information to entail c from the store, utcc inspires its synchronization mechanism on the notion of abstraction in functional programming languages. ($\lambda\ \vec{x}$; c) P can be seen as the dual version of (**local** $\vec{x}$; c) P in which the variables are *abstracted* with respect to the constraint c and the process P. The operational semantics provides the intuitions on how utcc processes interact. In principle, a configuration is represented by the tuple $\langle P, \mathrm{c}\rangle$ where P denotes a set of processes and c a constraint store. P can evolve to a further process P' during an *internal transition* ($\rightarrow$) where the constraint store c is monotonically refined, or can execute an *observable transition* ($\Longrightarrow$), producing the result of the future function of P and the constraint store d. The set of operational rules is presented in Figure 4.1, where $\langle P, \mathrm{c}\rangle$ denotes a configuration, and $F(P)$ denotes the *future function of* P.

$$\mathrm{R_T}\ \frac{}{\langle \mathbf{tell}\,(\mathrm{d}), \mathrm{c}\rangle \longrightarrow \langle \mathbf{skip}, \mathrm{c} \wedge \mathrm{d}\rangle} \qquad \mathrm{R_P}\ \frac{\langle P, \mathrm{c}\rangle \longrightarrow \langle P', \mathrm{c}'\rangle}{\langle P \,||\, Q, \mathrm{c}\rangle \longrightarrow \langle P' \,||\, Q, \mathrm{c}'\rangle} \qquad \mathrm{R_S}\ \frac{\gamma_1' \longrightarrow \gamma_2'}{\gamma_1 \longrightarrow \gamma_2}\ \text{if}\ \begin{matrix}\gamma_1 \equiv \gamma_1',\\ \gamma_2 \equiv \gamma_2'\end{matrix}$$

$$\mathrm{R_R}\ \frac{}{\langle !P, \mathrm{c}\rangle \longrightarrow \langle P \,||\, \mathbf{next}\,(!P), \mathrm{c}\rangle} \qquad \mathrm{R_U}\ \frac{d \Vdash c}{\langle \mathbf{unless}\ \mathrm{c}\ \mathbf{next}\ P, \mathrm{d}\rangle \longrightarrow \langle \mathbf{skip}, \mathrm{d}\rangle}$$

$$\mathrm{R_L}\ \frac{\langle P, (\exists \tilde{\mathrm{x}} \mathrm{d}) \wedge \mathrm{c}\rangle \longrightarrow \langle P', (\exists \tilde{\mathrm{x}} \mathrm{d}) \wedge \mathrm{c}'\rangle}{\langle (\mathbf{local}\ \vec{x}; \mathrm{c})\ P, \mathrm{d}\rangle \longrightarrow \langle (\mathbf{local}\ \vec{x}; \mathrm{c}')\ P', (\exists \tilde{\mathrm{x}} \mathrm{c}') \wedge \mathrm{d}\rangle}$$

$$\mathrm{R_A}\ \frac{d \Vdash c[\vec{t}/\vec{x}] \quad |\vec{t}| = |\vec{x}|}{\langle (\lambda\ \vec{x}; \mathrm{c})\ P, \mathrm{d}\rangle \longrightarrow \langle P[\vec{t}/\vec{x}] || (\lambda\ \vec{x}; \mathrm{c} \wedge (\tilde{\mathrm{x}} \neq \tilde{\mathrm{t}}))\ P, \mathrm{d}\rangle}$$

$$\mathrm{R_O}\ \frac{\langle P, \mathrm{c}\rangle \longrightarrow^* \langle Q, \mathrm{d}\rangle \not\longrightarrow}{P \overset{(c,d)}{\Longrightarrow} F(Q)}$$

$$\text{Where } F(Q) = \begin{cases} \mathbf{skip} & \text{if } Q = \mathbf{skip} \\ F(Q_1) \,||\, F(Q_2) & \text{if } Q = Q_1 \,||\, Q_2 \\ R & \text{if } Q = \mathbf{next}\,(R) \\ \mathbf{skip} & \text{if } Q = (\lambda\ \vec{x}; \mathrm{c})\ R \\ (\mathbf{local}\ \vec{x})\ F(R) & \text{if } Q = (\mathbf{local}\ \vec{x}; \mathrm{c})\ R \\ R & \text{if } Q = \mathbf{unless}\ \mathrm{c}\ \mathbf{next}\ R \end{cases}$$

Figure 4.1: Transition System for utcc: Internal and Observable transitions

Definition 4.2.3 (Structural Congruence). *Structural congruence (denoted by $\equiv$) is defined for* utcc *by the axioms:*

(i) $P \equiv Q$ *if they are* α*-equivalent.*

(ii) $P \,||\, \mathbf{skip} \equiv P$.

(iii) $P \,||\, Q \equiv Q \,||\, P$.

(iv) $P \,||\, (Q \,||\, R) \equiv (P \,||\, Q) \,||\, R$.

(v) $(\mathbf{local}\ \vec{x}; \mathrm{c})\ \mathbf{skip} \equiv \mathbf{skip}$.

(vi) $P \,||\, (\mathbf{local}\ \vec{x}; \mathrm{c})\ Q \equiv (\mathbf{local}\ \vec{x}; \mathrm{c})\ (P \,||\, Q)$ *if* $\vec{x} \notin fv(P)$.

(vii) $\langle P, c\rangle \equiv \langle Q, c\rangle$ *iff* $P \equiv Q$. *and*

(viii) $\mathbf{next}\,(P \,||\, \mathbf{next}\,(Q)) \equiv \mathbf{next}\,(P) \,||\, \mathbf{next}^2\,(Q)$.

Intuitively, the operational rules of utcc behaves almost in the same way as its counterpart in tcc, excepting by the general treatment of asks in utcc. Here we will describe the operational consequence of this change, we refer to [Olarte & Valencia 2008a] for further details on the operational semantics. Rule R_A describes the behavior of the abstraction $(\lambda\ \vec{x}; c)\ P$: a configuration here considers two stores, being c and d *local* and *global* stores respectively. If d entails $c[\tilde{t}/\tilde{x}]$ then $P[\vec{t}/\vec{x}]$ is executed. Moreover, the abstraction persists in time, allowing any other process to match with $\vec{x}$ in P while no other replacements of $\vec{x}$ with $\vec{t}$ will occur, as d is augmented with a constraint disallowing this. The notion of *local information* can be evidenced in R_L, considering a process $P = (\textbf{local}\ \vec{x}; c)\ Q$, we have to consider: (i) that the information about $\vec{x}$ locally for P subsumes any other information present for the same set of variables in the global store; therefore, $\vec{x}$ is hidden by the use of an existential quantifier over $\tilde{x}$ in d. (ii) that the information about $\vec{x}$ that P can produce after the reduction is still local, so we hide it by existentially quantifying $\vec{x}$ in c' before publishing it to the global store. After the reduction, c' will be the new local store of the evolution of internal processes. Finally, observable behaviour is described by R_o: after having used the internal transitions in a process P to evolve to a process Q with a quiescent-point (in which no more information can be added/inferred), the reduction will continue by executing the future function of Q with the resulting constraint store.

We will use the following auxiliar lemmas further on in our proofs:

Lemma 4.2.4. $\forall Q, \exists d$ *such that* $\langle Q, d\rangle \not\longrightarrow$ *then* $F(Q)$ *is defined.*

Proof. The proof proceeds by induction on the structure of Q and the definition of $\longrightarrow$ and $F(Q)$. □

Lemma 4.2.5 (Monotonicity). $\forall P, c; \exists Q, d$ *such that* $\langle P, c\rangle \longrightarrow^* \langle Q, d\rangle \not\longrightarrow$ *and* $d \Vdash c$.

Proof. The proof proceeds by induction on the length of the inference of $\longrightarrow$ as defined in Figure 4.1. □

4.3 utcc and Secure Pattern Matching

As described in Section 4.2, one of the main advantages of utcc with respect to tcc is that the universal abstraction operator allows for substitution of constraints for variables in processes. The extension has been proposed for the treatment of *mobile links* as present in the π-calculus [Milner *et al.* 1992] and pattern matching in modeling of security protocols. Below we will give two motivating examples for why a more refined abstraction operator is needed for modeling mobile *local* links and secret keys.

4.3.1 Motivating a refined universal abstraction in utcc

Our first example refers to the πcalculus-like mobility of local links. Consider the common scenario where a process P sends a request to a service offered by a process

Q and includes in the request a local link on which it expects the reply. This can be modeled in utcc using a constraint system $CS = (\Sigma, \Delta)$ where Σ includes the predicates req, rep, and res, and the constant 0. The processes P and Q are defined as

$$P = (\textbf{local}\ z)\ \big(\textbf{tell}\,(\mathrm{req(z)})\ \big|\big|(\lambda\ y; \mathrm{rep(z,y)})\ \textbf{next}\,(\textbf{tell}\,(\mathrm{res(y)}))\big)$$

and

$$Q = (\lambda\ x; \mathrm{req(x)})\ \textbf{tell}\,(\mathrm{rep(x,0)})$$

The predicates req and rep are used for the request and reply respectively, and the predicate res is used to report the result (and successful termination of P). The local operator is used to create a local variable z representing the local link.

The intention is that only the processes P and Q can synchronize via the local link z. However, the generality of abstraction in utcc makes it possible to violate this intention: Another process $E = (\lambda\ x, y; \mathrm{rep(x,y)})$ **skip** in parallel with the processes P and Q given above would be able to guess the link z (as well as the result) from the reply.

It is instructive to see how this could be avoided using the π-calculus, where the two processes could be modeled by

$$P = (\nu z)\big(\overline{\mathrm{req}}\langle z\rangle\ \big|\big|\ z\langle y\rangle.\overline{\mathrm{res}}\langle y\rangle\big) \quad \text{and} \quad Q = \mathrm{req}(x).\ \overline{x}\langle 0\rangle$$

In this case, the z and y are used differently in receiving the reply: The z is used as the communication channel and y is the binder for the received name. Another process in parallel would not be able to guess the channel z. As we will see below, our proposed type system for patterns allows to introduce this kind of distinction between the uses of variables in predicates.

Our second motivating example is from modeling of security protocols, where as pointed out in [Buchholtz *et al.* 2004] it it should be impossible for an agent to abstract variables if a one-way function has been applied to it. Consider a unary predicate o (used for output of messages to the network) and an encryption function enc(m, k) which represents the encryption of the variable m with the key k. A process P that sends out a local message n encrypted by a local key k can be represented by $P = (\textbf{local}\ k, n)\ \textbf{tell}\,(\mathrm{o(enc(n,k))})$. However, in utcc a spy process defined as $S = (\lambda\ x, z; \mathrm{o(enc(x,z))})\ \textbf{tell}\,(\mathrm{o(x)} \wedge \mathrm{o(z)})$, will succeed in retrieving and publishing both the key and the encrypted message.

As for the π-like channels, our proposed type system for patterns will allow us to rule out universal abstraction of variables to which a one-way function has been applied. Further, to be able to allow abstraction of the message when the key is locally known, we propose a novel kind of abstraction assuming local knowledge, which generalizes the universal abstraction of utcc.

4.3.2 Types for secure abstraction patterns in utcc

Based on the two motivating examples above, we argue that there are basically two sorts of arguments in functions and predicates: the ones that can be universally

quantifiable, which means that one would be able to use the abstraction operator for a variable in that argument in order to find a possible matching, and the ones that are not.

We will thus divide the arguments of predicates and functions in two sorts and write $P(\tilde{t};\tilde{t}')$ and $f(\tilde{t};\tilde{t}')$ for respectively the predicate P and function f where both $\vec{t}$ and $\vec{t'}$ are tuples of terms over the function signature F, and $\vec{t}$ denotes the restricted arguments and $\vec{t'}$ the unrestricted ones. We assume that both arguments of the equality predicate are restricted. If a predicate or function has either only restricted or unrestricted parameters and the sort is clear from the context, we will simply write $P(\tilde{t})$ and $f(\tilde{t})$.

The sorted predicates allow us to use a binary predicate piout(x; y) representing the π–like communication of y (the object) on the channel x (the subject). By defining that the subject is a restricted argument and the object an unrestricted argument we obtain the required asymmetry in the roles of the variables. The type rules for patterns should then forbid the abstraction $(\lambda\ x; \mathrm{piout}(x, y))\ P$, as it would allow us to identify all channels (also channels not known to us) containing a particular message y. However, they should *allow* the abstraction $(\lambda\ y; \mathrm{piout}(x, y))\ P$, reflecting that we can compute the possible messages on a channel x known to us. That is, we want to capture that if we *know* the values of the restricted variables, then we may abstract (i.e. compute all possible matches for) the unrestricted variables.

Similarly, sorted functions allow us to represent semantically that some functions are *one-way* functions such as the function enc(k, m) described above for encrypting the message m by the key k. Sorting both arguments as restricted will ensure that e.g. the abstractions $(\lambda\ \vec{x}; o(\mathrm{enc}(k, m)))\ P$ will be forbidden for any non-empty $\vec{x} \subseteq \{k, m\}$ Thus, even if the single argument of the o predicate is unrestricted (i.e. we can abstract all messages available on the network) then we can not compute the inverse of the encryption function. We may have functions for which an inverse is assumed to exist, such as a function $\mathrm{tup}_2(x, y)$ for making a pair of x and y. In that case it makes sense to allow abstractions over the two arguments by sorting them as unrestricted.

In general, patterns may be a conjunction of several predicates and thus variables may occur both restricted and unrestricted in the same pattern. An example of this is the abstraction $(\lambda\ y, z; c)\ P$, where $c = \mathrm{piout}(y, z) \wedge \mathrm{piout}(x, y)$. We argue that this pattern should be *allowed*, since it is possible first to match the unrestricted y in piout(x, y) and then subsequently, for the given y, match the unrestricted z in piout(y, z). Note that it is not enough simply to require the abstracted variables to occur unrestricted: Both variables x and y appear unrestricted in the abstraction $(\lambda\ x, y; \mathrm{piout}(x, y) \wedge \mathrm{piout}(y, x))\ P$, but neither of the two basic constraints can be matched without abstracting a restricted variable. As solution we define a set of type rules for constraints used as patterns in abstractions which capture that there exists an order of the basic constraints in which the first occurrence of each variable is unrestricted.

To allow abstractions in cases where the inverse key of the encryption is known we add a new rule $R_{A\rightarrow}$ given in Equation 4.1 in addition to the SOS rules pictured

$$\frac{}{\Gamma^R;\Gamma^U \vdash \mathsf{P}(\vec{t};\vec{t'}) : pat}\ \mathsf{T}_{\text{pred}}\ \Gamma^R = var(\vec{t}) \cup res(\vec{t'}) \text{ and } \Gamma^U = unr(\vec{t'})\backslash\Gamma^R$$

$$\frac{\Gamma \vdash c_1 \wedge (c_2 \wedge c_3) : pat}{\Gamma \vdash (c_1 \wedge c_2) \wedge c_3 : pat}\ \mathsf{T}_{\text{assoc}} \qquad \frac{\Gamma \vdash c_1 \wedge c_2 : pat}{\Gamma \vdash c_2 \wedge c_1 : pat}\ \mathsf{T}_{\text{commute}}$$

$$\frac{\Gamma_1^R;\Gamma_1^U \vdash c_1 : pat \quad \Gamma_2^R;\Gamma_2^U \vdash c_2 : pat}{\Gamma^R;\Gamma^U \vdash c_1 \wedge c_2 : pat}\ \mathsf{T}_{\text{comb}}\ \Gamma^R = (\Gamma_1^R \cup \Gamma_2^R)\backslash\Gamma_1^U \text{ and } \Gamma^U = (\Gamma_1^U \cup \Gamma_2^U)\backslash\Gamma_1^R$$

$$\frac{}{\vdash \mathbf{skip} : sec}\ \mathsf{T}_{\text{skip}} \qquad \frac{}{\vdash \mathbf{tell}\,(c) : sec}\ \mathsf{T}_{\text{tell}} \qquad \frac{\vdash P : sec}{\vdash\, !P : sec}\ \mathsf{T}_{\text{bang}}$$

$$\frac{\vdash P : sec \quad \vdash Q : sec}{\vdash P \,||\, Q : sec}\ \mathsf{T}_{\text{par}} \qquad \frac{\vdash P : sec}{\vdash \mathbf{next}\,(P) : sec}\ \mathsf{T}_{\text{next}}$$

$$\frac{\vdash P : sec}{\vdash (\mathbf{local}\ \vec{x}; c)\ P : sec}\ \mathsf{T}_{\text{loc}} \qquad \frac{\vdash P : sec}{\vdash \mathbf{unless}\, c\, \mathbf{next}\, P : sec}\ \mathsf{T}_{\text{unls}}$$

$$\frac{\vdash P : sec \quad \Gamma^R;\Gamma^U \vdash c : pat}{\vdash (\lambda\ \vec{x}; d \implies c)\ P : sec}\ \mathsf{T}_{\text{abs}}\quad \vec{x} \subseteq dom(\Gamma^U)\backslash fv(d)$$

Figure 4.2: Typing rules for secure patterns and processes

in Figure 4.1. $\mathsf{R}_{\mathsf{A}\rightarrow}$ allows for abstractions using constraints of the form $c \Rightarrow c'$, that is, assuming local knowledge c and a global store d, one can infer c'. The idea is to infer c' using c but without publishing it permanently to the store, as captured by the following operational rule:

$$\frac{d \wedge c \Vdash c'[\tilde{t}/\tilde{x}] \quad |\tilde{t}| = |\tilde{x}| \quad d \wedge c \Vdash ff \Rightarrow d \Vdash ff}{\langle(\lambda\ \vec{x}; c \Rightarrow c')\ P, d\rangle \longrightarrow \langle P[\vec{t}/\vec{x}] \,||\, (\lambda\ \vec{x}; c \Rightarrow (c' \wedge (\tilde{x} \neq \tilde{t}))\ P, d\rangle}\ \mathsf{R}_{\mathsf{A}\rightarrow} \tag{4.1}$$

The condition $d \wedge c \Vdash ff \Rightarrow d \Vdash ff$ ensures that local assumptions do not make the store inconsistent when combining with the constraint store.

The typing rules for secure patterns and processes are defined in Figure 4.2. For simplicity we assume patterns are simply conjunction of predicates applied to

terms over the function signature. The typing rules use an environment $\Gamma = \Gamma^R; \Gamma^U$, where Γ^R is the set of names used restricted and Γ^U is the set of names used unrestricted. When the distinction does not matter we simply write Γ. We employ three inductively defined functions on terms over the function signature: $unr(t)$, $res(t)$, and $var(t)$ yielding respectively the variables appearing unrestricted in t according to the sorting, the variables appearing restricted in t, and all variables appearing in t. We extend the functions to vectors of terms by $unr(\vec{t}) = \cup_{1 \leq i \leq |\vec{t}|} unr(t_i)$ (and similarly for res and var). Formally, the functions are given by $unr(x) = res(x) = var(x) = \{x\}$ for any variable x, and $unr(f(\vec{t}; \vec{t'})) = unr(\vec{t'})$, $res(f(\vec{t}; \vec{t'})) = res(\vec{t})$, and $var(f(\vec{t}; \vec{t'})) = var(\vec{t}) \cup var(\vec{t'})$. Note that obviously $var(t) = res(t) \cup unr(t)$ but also that $res(t) \cap unr(t)$ may be non-empty, i.e. a variable may appear both restricted and non-restricted.

The rule T_{Pred} captures that all variables in $\vec{t}$ as well as the variables occurring restricted in $\vec{t'}$ in the predicate $P(\vec{t}; \vec{t'})$ are restricted. The rest of the variables are unrestricted. The rules T_{asoc} and T_{commute} allow us to change the ordering of the basic constraints. Finally, the rule T_{comb} identifies the restricted and unrestricted variables in the joint pattern $c_1 \wedge c_2$ assuming that c_1 is matched first. That is, a variable is restricted if it appears restricted in either of the sub patterns c_1 and c_2 and not unrestricted in c_1. (If it appears unrestricted in c_1 it will be instantiated if c_1 is matched first, and thus it is allowed to appear restricted in c_2). Dually, the unrestricted variables in the joint pattern $c_1 \wedge c_2$ are the variables that appear unrestricted in either of the sub patterns c_1 and c_2, and do not appear restricted in c_1.

The objective of the type system is to determine the secure patterns, therefore typing rules over processes are rather simple. The only non-trivial rule is the rule T_{abs} for abstractions, which ensure that c is a valid pattern such that the abstracted variables are unrestricted, and no variables in the local d are abstracted.

Theorem 4.3.1 (Termination of type checking). *For any process P the type-checking process terminates.*

Proof. (Sketch) Follows from the fact that there are only finitely many permutations of basic constraints (predicates) in a pattern.

□

The following lemmas are used to prove subject reduction.

Lemma 4.3.2 (Constraint substitution does not affect pattern typing). *Given* $\Gamma^R; \Gamma^U \vdash c : pat$ *and* t *and* x*, then* $\Gamma^{R'}; \Gamma^{U'} \vdash c[t/x] : pat$ *and* $\Gamma^U \setminus (fv(t) \cup \{x\}) \subseteq \Gamma^{U'} \setminus (fv(t) \cup \{x\})$.

Proof. The proof proceeds by induction on the type inference of $\Gamma^R; \Gamma^U \vdash c : pat$.

- Case for $c = P(\tilde{t}, \tilde{t'})$: We have that $\Gamma^R = \left(var(\vec{t}) \cup rv(\vec{t'})\right); \Gamma^U = tv(\vec{t'}) \setminus \left(var(\vec{t}) \cup rv(\vec{t'})\right)$.

  If $x \notin fv(c)$ then $c[t/x] = c$, $\Gamma^{R'} = \Gamma^R$ and $\Gamma^{U'} = \Gamma^U$. Therefore, $\Gamma^{R'}; \Gamma^{U'} \vdash c[t/x] : pat$.

  When $\Gamma^{R'}; \Gamma^{U'} \vdash P(\tilde{t}, \tilde{t'})[t/x] : pat$ we have the following cases:

  - Case for $x \in fv(\mathrm{c})$: We have to check the use of x over the set of top variables of $\mathrm{P}(\tilde{\mathrm{t}}, \tilde{\mathrm{t}'})$. If $x \in tv(\vec{t'})$ then $\Gamma^{R'}; \Gamma^{U'} \vdash \mathrm{P}(\tilde{\mathrm{t}}, \tilde{\mathrm{t}'})[\mathrm{t}/\mathrm{x}] : pat$ where $\Gamma^{R'} = \Gamma^R \cup (fv(t) \backslash tv(t))$ and $\Gamma^{U'} = \left(\Gamma^U \backslash \{x\}\right) \cup tv(t)$. Moreover, $\Gamma^U \backslash (fv(t) \cup \{x\}) = \Gamma^{U'} \backslash (fv(t) \cup \{x\})$
  - Case for $x \notin tv(\vec{t'})$: Then $\Gamma^{U'} = tv(\vec{t'}) = \Gamma^U$ and $\Gamma^{R'} = \Gamma^R \cup \{fv(\vec{t}) \backslash tv(\vec{t'})\}$, from $\mathrm{T}_{\mathrm{Pred}}$ we get $\Gamma^{R'}; \Gamma^{U'} \vdash \mathrm{c}[t/x] : pat$. Then $\Gamma^{U'} \backslash (fv(t) \cup \{x\}) = \Gamma^U \backslash (fv(t) \cup \{x\})$ as $\Gamma^U = \Gamma^{U'}$.

- Case for $\mathrm{c} = \mathrm{c}_1 \wedge \mathrm{c}_2$: Assume $\Gamma^R; \Gamma^U \vdash \mathrm{c}_1 \wedge \mathrm{c}_2 : pat$. We have to show that $\Gamma^{R'}; \Gamma^{U'} \vdash (\mathrm{c}_1 \wedge \mathrm{c}_2)[t/x] : pat$ and $\Gamma^U \backslash (fv(t) \cup \{x\}) \subseteq \Gamma^{U'} \backslash (fv(t) \cup \{x\})$.

  From the rule $\mathrm{T}_{\mathrm{comb}}$ it follows that $\Gamma_1^R; \Gamma_1^U \vdash \mathrm{c}_1 : pat$ and $\Gamma_2^R; \Gamma_2^U \vdash \mathrm{c}_2 : pat$ with $\Gamma^R = \left(\Gamma_1^R \cup \Gamma_2^R\right) \backslash \Gamma_1^U$ and $\Gamma^U = \left(\Gamma_1^U \cup \Gamma_2^U\right) \backslash \Gamma_1^R$.

  From the induction hypothesis we have that, for $i \in \{1, 2\}$, $\Gamma_i^{R'}; \Gamma_i^{U'} \vdash \mathrm{c_i}[t/x] : pat$. Moreover,

$$\Gamma_i^U \backslash \left(fv(\vec{t}) \cup \{x\}\right) \subseteq \Gamma_i^{U'} \backslash \left(fv(\vec{t}) \cup \{x\}\right) \tag{4.2}$$

  and

$$\Gamma_i^{R'} \subseteq \Gamma_i^R \cup fv(\vec{t}) \tag{4.3}$$

  It follows from $\mathrm{T}_{\mathrm{comb}}$ that $\Gamma^{R'}; \Gamma^{U'} \vdash \mathrm{c}_1[t/x] \wedge \mathrm{c}_2[t/x] : pat$ for $\Gamma^{R'} = (\Gamma_1^{R'} \cup \Gamma_2^{R'}) \backslash \Gamma_1^{U'}$ and $\Gamma^{U'} = (\Gamma_1^{U'} \cup \Gamma_2^{U'}) \backslash \Gamma_1^{R'}$.

  We must show that

$$\Gamma^U \backslash (fv(t) \cup \{x\}) \subseteq \Gamma^{U'} \backslash (fv(t) \cup \{x\}) \tag{4.4}$$

  Substituting Γ^U by its respective definition it follows that

$$\left(\left(\Gamma_1^U \cup \Gamma_2^U\right) \backslash \Gamma_1^R\right) \backslash (fv(t) \cup \{x\}) \subseteq ((\Gamma_1^{U'} \cup \Gamma_2^{U'}) \backslash \Gamma_1^{R'}) \backslash (fv(t) \cup \{x\}) \tag{4.5}$$

  Which holds by substituting Γ^U and $\Gamma^{U'}$ by their definitions and using equations 4.2 and 4.3.

□

Lemma 4.3.3 (Constraint substitution does not affect process typing). *Given a typing judgment* $\vdash P' : sec$ *then* $\vdash P'[t/x] : sec$.

Proof. The proof proceeds by induction on the type inference of $\vdash P' : sec$

- Base case, rule $\mathrm{T}_{\mathrm{skip}}$: Let $P' =$ **skip** and $var(P') = \emptyset$, then $P'[t/x] : sec$ trivially.
- Base case, rule $\mathrm{T}_{\mathrm{tell}}$: Let $P' =$ **tell** (c) and $var(P') = \emptyset$, then $P'[t/x] =$ **tell** (c[t/x]), and applying $\mathrm{T}_{\mathrm{tell}}$ it follows that **tell** (c[t/x]) : sec.

- Inductive step, rule T_{par}: Let $P' = P \,||\, Q$, and $\vdash P \,||\, Q : sec$. We have to show that $\vdash P[t/x] \,||\, Q[t/x] : sec$. From T_{par} it follows that $\vdash P : sec$ and $\vdash Q : sec$. Moreover, from the Induction hypothesis we have that $\vdash P[t/x] : sec$ and $\vdash Q[t/x] : sec$. It follows from T_{par} that $\vdash P[t/x] \,||\, Q[t/x] : sec$.

- Inductive steps: rules $T_{next}, T_{unls}, T_{loc}$: We proceed similarly as for T_{par}.

- Inductive step: rule T_{abs}: Let $P' = (\lambda\ x; \mathrm{d} \Rightarrow \mathrm{c})\ P$ and $\vdash (\lambda\ x; \mathrm{d} \Rightarrow \mathrm{c})\ P : sec$, we have to show that $\vdash (\lambda\ x; \mathrm{d} \Rightarrow \mathrm{c[t/x]})\ P[t/x] : sec$. From T_{abs} it follows that $\vdash P : sec$ and $\Gamma^R; \Gamma^U \vdash \mathrm{c} : pat$ for $\vec{x'} \subseteq dom(\Gamma^U) \backslash fv(\mathrm{d})$. From the induction hypothesis we have that $\vdash P[t/x] : sec$ and from lemma 4.3.2 we have that $\Gamma^{R'}; \Gamma^{U'} \vdash \mathrm{c[t/x]} : pat$ with $\vec{x'} \subseteq dom(\Gamma^U) \backslash fv(\mathrm{d}) \subseteq dom(\Gamma^{U'}) \backslash (fv(\mathrm{d}) \cup \{x\})$. From the application of T_{abs} it follows that $\vdash (\lambda\ x; \mathrm{d} \Rightarrow \mathrm{c[t/x]})\ P[t/x] : sec$, which is what we had to prove.

□

Lemma 4.3.4 (Structural equivalence preserves typing). *Given P, Q processes, if $P \equiv Q$ and $\vdash P : sec$, then $\vdash Q : sec$.*

Proof. The proof proceeds by trivial case analysis over the structural congruence rules in Definition 4.2.3. □

Next we check that secure processes can not be made insecure during an internal transition step.

Lemma 4.3.5. *If $\langle P, \mathrm{c} \rangle \longrightarrow \langle Q, \mathrm{d} \rangle$ and $\vdash P : sec$, then $\vdash Q : sec$.*

Proof. The proof proceeds by induction on the depth of the inference $\langle P, \mathrm{c} \rangle \longrightarrow \langle Q, \mathrm{d} \rangle$ and using the definition of $\vdash P : sec$.

- Base case, rule R_T: Let $P = \mathbf{tell}\,(\mathrm{d'})$, $Q = \mathbf{skip}$ and $\mathrm{d} = \mathrm{c} \wedge \mathrm{d'}$. We want to show that if $\vdash P : sec$ then $\vdash Q : sec$, which follows trivially from T_{skip}.

- Base case, rule R_U: Let $P = \mathbf{unless}\,\mathrm{c'}\,\mathbf{next}\,P$ and $Q = \mathbf{skip}$ and $\mathrm{d} = \mathrm{c}$. If $\vdash P : sec$ we must show that $\vdash Q : sec$, which follows from T_{skip}.

- Inductive case, rule R_R: Let $P = !P'$ and $Q = P' \,||\, \mathbf{next}\,(!P')$ and $\mathrm{d} = \mathrm{c}$. From the induction hypothesis we get $\vdash P' : sec$. Using T_{bang} and $\vdash !P' : sec$ we get $\vdash P' : sec$. Similarly, from T_{next} and $\vdash !P' : sec$ we get $\vdash \mathbf{next}\,(!P' : sec)$. Finally, from T_{abs} we get $\vdash P' \,||\, \mathbf{next}\,(!P') : sec$.

- Inductive case, rule R_L: Let $P = (\mathbf{local}\ x; \mathrm{c'})\ P'$ and $Q = (\mathbf{local}\ x; \mathrm{c''})\ P''$ and $\mathrm{d} = (\exists \mathrm{x}; \mathrm{c''}) \wedge \mathrm{c}$. We assume that $\vdash P' : sec$, and by inductive hypothesis then $\vdash P'' : sec$. We have to show that $\vdash (\mathbf{local}\ x; \mathrm{c''})\ P'' : sec$, which follows from the application of T_{loc} and $\vdash P'' : sec$.

- Inductive case, rule R_S: we have that $\gamma_1 = \langle P_1, C_1\rangle$ and $\gamma_2 = \langle Q, d\rangle$. We assume that for $\gamma_1' = \langle P_1', c_1'\rangle$, $\vdash P_1' : sec$, then from the inductive hypothesis we get $\gamma_2' = \langle P_2', c_2'\rangle$, such that $\vdash P_2' : sec$. We have to show that if $\vdash P_1 : sec$, then there is a $\gamma_2 = \langle Q, d\rangle$ such that $\vdash Q : sec$, which follows from lemma 4.3.4.

- Inductive case, rule $R_{A\rightarrow}$: Let $P = (\lambda\ \vec{x}; c \Rightarrow c')\ P'$ and $Q = P'[\vec{t}/\vec{x}]\,||(\lambda\ \vec{x}; c \Rightarrow (c' \wedge (\tilde{x} \neq \tilde{t})))\ P'$ and $d = c$. We assume that $\vdash P' : sec$ and $\Gamma^R; \Gamma^U \vdash c' : pat$ and $x \subseteq dom(\Gamma^U)\backslash fv(c)$. We have to show that

$$\vdash P'[\vec{t}/\vec{x}]\,||(\lambda\ \vec{x}; c \Rightarrow (c' \wedge (\tilde{x} \neq \tilde{t})))\ P' : sec \tag{4.6}$$

  such that $\vec{x} \subseteq dom(\Gamma^{U'})\backslash fv(c)$.

  From T_{pred} we get that $\Gamma^{R''}; \Gamma^{U''} \vdash \vec{x} \neq \vec{t} : pat$ where inequality can be seen as a predicate between two set of variables and $\Gamma^{R''} = \{\vec{x}\} \cup var(\vec{t}); \Gamma^{U''} = \emptyset$. Using the assumption $\Gamma^R; \Gamma^U \vdash c' : pat$ and $\Gamma^{R''}; \Gamma^{U''} \vdash \vec{x} \neq \vec{t} : pat$ with T_{comb} we get

$$\Gamma^{R'}; \Gamma^{U'} \vdash c' \wedge \tilde{x} \neq \tilde{t} : pat \tag{4.7}$$

  with $\Gamma^{R'} = \left(\Gamma^R \cup \{\vec{x}\} \cup var(\vec{t})\right) \backslash \Gamma^U$ and $\Gamma^{U'} = \Gamma^U \backslash \Gamma^R = \Gamma^U$. Moreover, $\vec{x} \subseteq dom(\Gamma^{U'})\backslash fv(c)$ as $\vec{x} \subseteq dom(\Gamma^U)\backslash fv(c)$.

  From the assumption we have that $\vdash P' : sec$ and applying lemma 4.3.3 it follows that $\vdash P'[\vec{t}/\vec{x}] : sec$, Finally, from this last result together with equation 4.7 and T_{abs} we can derive
  $\vdash P'[\vec{t}/\vec{x}]\,||(\lambda\ \vec{x}; c \Rightarrow (c' \wedge (\tilde{x} \neq \tilde{t})))\ P' : sec$, which exactly the same as Equation 4.6, hence we are done.

□

Finally, we show that if a process P is well-typed, it can not perform any internal steps, and its future is defined then the future of P is also well-typed.

Lemma 4.3.6. *For all* $\vdash P : sec$, *if* $F(P)$ *is defined and* $\exists d.\langle P, d\rangle \not\longrightarrow$ *then* $\vdash F(P) : sec$.

Proof. The proof proceed by induction in the definition of $F(P)$.

- Base case, $P = \textbf{unless}\, c'\, \textbf{next}\, P'$: we have that $\vdash \textbf{unless}\, c'\, \textbf{next}\, P' : sec$. By T_{unls} we get $\vdash P' : sec$, and as $F(\textbf{unless}\, c'\, \textbf{next}\, P') = P'$, then $\vdash F(P) : sec$. This case is similar for $P = \textbf{next}\,(P')$.

- Inductive case, $P = P_1\,||\,P_2$: we have that $\vdash (P_1\,||\,P_2) : sec$. Then by T_{par} we get that for $i = \{1, 2\}$, then $\vdash P_i : sec$. By lemma 4.2.4 we have that for P_i, $F(P_i)$ is defined. We assume $F(P_i) = P_i'$ and $\vdash P_i' : sec$. We have that $F(P_1\,||\,P_2) = P_1'\,||\,P_2'$ and by T_{par} and the inductive hypothesis we get that $\vdash (P_1'\,||\,P_2') : sec$, then $\vdash F(P) : sec$.

- Inductive case, $P = (\textbf{local}\ x; c')\ P'$:, we have that $\vdash (\textbf{local}\ x; c')\ P' : sec$. Assuming $F(P') = P''$ is defined and $\vdash P'' : sec$ then by $\mathsf{T}_{\mathsf{loc}}$, and the inductive hypothesis, we get that $\vdash (\textbf{local}\ x; c')\ P'' : sec$, then $\vdash (\textbf{local}\ x; c')\ F(P') : sec$.

□

We now have all the ingredients to prove subject reduction.

Theorem 4.3.7 (Subject-reduction). *If* $P \stackrel{(c,d)}{\Longrightarrow} Q$ *and* $\vdash P : sec$ *, then* $\vdash Q : sec$ *.*

Proof. Assume $P \stackrel{(c,d)}{\Longrightarrow} Q$ and $\vdash P : sec$, then by rule R_o we get that $\langle P, c\rangle \longrightarrow^n \langle Q', d\rangle \not\longrightarrow$ and $Q = F(Q')$. We proceed by induction in n.

In the base case where $n = 0$, we have that $Q' = P$ and $c = d$. It follows from lemma 4.3.6 that $\vdash F(Q') : sec$.

For the induction step, assume $\langle P, c\rangle \longrightarrow^1 \langle P', c'\rangle \longrightarrow^n \langle Q', d\rangle \not\longrightarrow$. Then $\vdash P' : sec$ by lemma 4.3.5 and thus we get by induction that $\vdash F(Q') : sec$.

□

4.4 Applications

This section illustrates the use of the type system with some examples in mobility and security.

4.4.1 Mobility & Access Control

First, let us return to the π-calculus example. We assume the syntactic sugar $x\langle y\rangle$ stands for the binary predicate piout(x; y) and represents the use of the (restricted) channel x with the (unrestricted) message y. The following type inference shows that we can quantify over either x or y for the pattern $y\langle x\rangle \wedge x\langle y\rangle$:

$$\dfrac{\dfrac{}{x; y \vdash x\langle y\rangle : pat}\ \mathsf{T}_{\mathsf{pred}} \quad \dfrac{}{y; x \vdash y\langle x\rangle : pat}\ \mathsf{T}_{\mathsf{pred}}}{x; y \vdash x\langle y\rangle \wedge y\langle x\rangle : pat}\ \mathsf{T}_{\mathsf{comb}}$$

The way to read the first inference is that we can abstract y if we know x. Conversely, a second inference from the same pattern can lead to a typing of the form $y; x \vdash y\langle x\rangle \wedge x\langle y\rangle : pat$

$$\dfrac{\dfrac{}{y; x \vdash x\langle y\rangle : pat}\ \mathsf{T}_{\mathsf{pred}} \quad \dfrac{}{x; y \vdash y\langle x\rangle : pat}\ \mathsf{T}_{\mathsf{pred}}}{y; x \vdash x\langle y\rangle \wedge y\langle x\rangle : pat}\ \mathsf{T}_{\mathsf{comb}}$$

capturing the fact that one can abstract x if we know y. However, note that we can not infer $\varepsilon; x, y \vdash x\langle y\rangle \wedge y\langle x\rangle : pat$, and thus we are not allowed to simultaneously quantify over x and y.

It is interesting to note the resemblance of the type environments and type rules herewith presented and classical definitions of information flow policies based on lattice models [Bell & LaPadula 1973, Denning 1976, Biba 1977, Sandhu 1993]. The unrestricted and restricted typing environments map to high and low security levels, while the composition of the type rules given by $\mathsf{T}_{\mathsf{pred}}$ and $\mathsf{T}_{\mathsf{comb}}$ allow for a flow of data from high to low security clearances, just as the high-low lattice model in [Sandhu 1993]. This is the same kind of behaviour exhibited in confidentiality policies, like the one proposed by [Bell & LaPadula 1973], where data labelled with a high level of confidentiality will decrease its level when composed with a minor confidentiality level. We can see an example on how this works in the variable extrusion of the process $(\lambda\ x, z; \mathsf{y}\langle\mathsf{x}\rangle \wedge \mathsf{x}\langle\mathsf{z}\rangle)$ **skip**:

$$\dfrac{\dfrac{}{\vdash \textbf{skip} : sec}\ \mathsf{T}_{\mathsf{skip}} \qquad \dfrac{\dfrac{}{y; x \vdash \mathsf{y}\langle\mathsf{x}\rangle : pat}\ \mathsf{T}_{\mathsf{pred}} \qquad \dfrac{}{x; z \vdash \mathsf{x}\langle\mathsf{z}\rangle : pat}\ \mathsf{T}_{\mathsf{pred}}}{y; x, z \vdash \mathsf{y}\langle\mathsf{x}\rangle \wedge \mathsf{x}\langle\mathsf{z}\rangle : pat}\ \mathsf{T}_{\mathsf{comb}}}{\vdash (\lambda\ x, z; \mathsf{y}\langle\mathsf{x}\rangle \wedge \mathsf{x}\langle\mathsf{z}\rangle)\ \textbf{skip} : sec}\ \mathsf{T}_{\mathsf{abs}}$$

Here, the security label of channel x in $x\langle z\rangle$, originally restricted (secret in nature) gets decreased when it is composed with the communication $y\langle x\rangle$. It is easy to imagine that this type system could accommodate other security hierarchies used in real life scenarios, such as Smith's lattice based access control model for military applications [Smith 1990].

4.4.2 Security Protocols

To illustrate the application of utcc_s in the security domain, we follow the lines of the Security Protocol Language (SPL) [Crazzolara & Winskel 2001] and SCCP [Olarte & Valencia 2008b] to define a specification language for security protocols that we have called the Security Protocol Concurrent Constraint Programming (SPCCP) language. The SPCCP embeds utcc_s in a syntax suitable for defining security protocols, capturing process specifications with respect to input and output events over a global network. The SPCCP language combines the best ideas from SPL and SCCP by having a simple notion of pattern matching as in SPL and using the constraint system to model the attackers ability to combine and split messages as in SCCP. Hereto we add the new concept of *pattern matching under local knowledge*, which allow us to syntactically guarantee that only message parts inferable from the available keys are extracted, which can not be guaranteed in SPL nor in SCCP.

Definition 4.4.1 (SPCCP**).** *The Secure Concurrent Constraint Programming language* SCCP *[Olarte & Valencia 2008b] is redefined by the grammar in Figure 4.3, where* $\vec{x}$ *range over a set of variables and the subscript* $\vec{k}$ *in* $\mathbf{in}_{\forall\vec{x}}[N]_{\vec{k}}.R$ *is a set of keys.*

We define the semantics of SPCCP by giving a translation into utcc_s with a security constraint system given by the signature Σ with a single (unrestricted) unary predicate o(t) used for message output, function symbols $\mathsf{F} = \{\mathsf{enc}, \mathsf{pub}, \mathsf{priv},$

Values	v,v′	=	x
			$\mid k$
Keys	k	=	$pub(x)$
			$\mid priv(x)$
			$\mid sym(x)$
Messages and patterns	M,N	=	v
			$\mid (M_1, \ldots, M_n)$
			$\mid \{M\}_k$
Processes	R	=	**nil**
			$\mid$ **local**(x) **in** R
			$\mid$ **out**(M) $.R$
			$\mid$ **in**$_{\forall \vec{x}[N]_{\vec{k}}}.R$
			$\mid !R$
			$\mid R \,\|\|\, R$

Figure 4.3: SPCCP : Process syntax

$\mathrm{sym}, \mathrm{tup}_n\}$, and entailment relation given in Figure 4.4 inspired on the requirements stated by Dolev and Yao in [Dolev & Yao 1981].

The binary function enc takes two unrestricted arguments: a key and a message. The key is intended to be either a symmetric, private, or public key generated by the (restricted) unary functions $sym(x)$, $priv(x)$, or $pub(x)$ respectively. Letting $\mathrm{k} \in \{pub, priv, sym\}$ and defining $sym^{-1} = sym$, and $pub^{-1} = priv$, the entailment rule scheme $\mathrm{E}_{\mathrm{k-dec}}$ for decryption expresses how enc acts as symmetric or asymmetric encryption. The n-ary (unrestricted) tupling functions tup_n allow to create n-ary tuples, from which the individual elements can be projected as expressed by the entailment rule $\mathrm{E}_{\mathrm{proj}}$. As usual, the rules $\mathrm{E}_{\mathrm{enc}}, \mathrm{E}_{\mathrm{k-key}}$, and $\mathrm{E}_{\mathrm{tup}_n}$ express that the output of any function of known output values can be inferred.

The messages/patterns of SPCCP are mapped to the terms generated by the corresponding function symbols and variables in the security constraint system, using the usual notation $(M_1, \ldots, M_n)$ for n-tuples and $\{M\}_k$ for $\mathrm{enc}(\mathrm{k}, M)$. For a message M of SPCCP let $v(M)$ denote the set of variables in M. For a set of values $\vec{v} = \{v_1, v_2, \ldots, v_i\}$ let $o(\vec{v})$ be short for $o(v_1) \wedge o(v_2) \wedge \ldots \wedge o(v_i)$, and in particular $o(\emptyset) = \mathrm{tt}$.

We are now ready to define the encoding of SPCCP in utcc_s.

Definition 4.4.2 (SPCCP encoding). *Let* $[\cdot]_{\mathrm{utcc}}$ *be the mapping from* SPCCP *to* utcc_s *processes, given by:*

$$\frac{c \Vdash o(k^{-1}(x)) \quad c \Vdash o(enc(k(x), m))}{c \Vdash o(m)}\ E_{k-dec} \quad \text{for } k \in \{sym, pub\}, sym^{-1} = sym, \text{ and } pub^{-1} = priv$$

$$\frac{c \Vdash o(x) \qquad c \Vdash o(y)}{c \Vdash o(enc(x, y))}\ E_{enc} \qquad \frac{c \Vdash o(x)}{c \Vdash o(k(x))}\ E_{k-key}, \text{ for } k \in \{sym, pub, priv\}$$

$$\frac{c \Vdash o(i_1) \quad \ldots \quad c \Vdash o(i_n)}{c \Vdash o(tup_n(i_1, \ldots, i_n))}\ E_{tup_n} \qquad \frac{c \Vdash o(tup_n(i_1, \ldots, i_n))}{c \Vdash o(i_j)}\ E_{proj} \quad j \in \{1, \ldots, n\}$$

Figure 4.4: Entailment relation for a security constraint system.

$$[R]_{utcc} \quad : \quad \begin{cases} \textbf{skip} & \text{if } R = \textbf{nil} \\ (\textbf{local}\, x)\ [R']_{utcc} & \text{if } R = \textbf{local}(x)\, \textbf{in}\, R' \\ \textbf{tell}\,(o(M))\ \ ||\ \ \textbf{next}\,([R']_{utcc}) & \text{if } R = \textbf{out}(M)\,.R' \\ (\lambda\ \vec{x}; o(\tilde{k}) \Rightarrow o(N) \wedge o(\tilde{x}))\ \textbf{next}\,([R']_{utcc}) & \text{if } R = \textbf{in}_{\forall \vec{x}}[N]_{\vec{k}}.R' \\ ![R']_{utcc} & \text{if } R = !R' \\ [R']_{utcc}\, |||\, [R'']_{utcc} & \text{if } R = R' \,||\, R'' \end{cases}$$

We will focus on outlining process constructions for pattern matching and network output. The remaining process constructions are mapped directly to the corresponding construct in utcc$_s$: **nil**, $R \,||\, R'$ and $!R$ have the usual meaning of inaction, parallel composition and replication in process calculi; $\textbf{out}(M).R$ adds the constraint o(M) to the constraint store and subsequently in the next time period behaves as (the encoding of) R.

SPCCP differs from SCCP in the treatment of keys and the input operation: priv(x), pub(x), and sym(x) yields respectively the private, public and symmetric key from generator x. The input operator written as $\textbf{in}_{\forall \vec{x}}[N]_{\vec{k}}.P$ should be read as "for all possible messages $\vec{m}$ (available under the assumption of knowing the keys $\vec{k}$) such that $N[\vec{m}/\vec{x}]$ is available as message at the network evolve into $P[\vec{m}/\vec{x}]$". Intuitively, the idea is to check if $\vec{m}$ is available as knowledge assuming locally that the keys in $\vec{k}$ are available as knowledge, and if so, bind the variables in P occurring in the pattern N with the corresponding values in $\vec{m}$. The pattern matching resembles the pattern matching construct in SPL. The key difference is that it proceed for all possible matches, and that we employ the new rule for universal abstraction under local knowledge introduced in the previous section to allow the use of private keys as local information to perform the decryption of messages. Note that we also require that all the abstracted values can be inferred as output. This guarantees that secret values are not abstracted, and result in well-typeness of the encoding.

Proposition 4.4.3 (SPCCP maps to well-typed utcc$_s$ processes). *For any SPCCP process* P, $\vdash [P]_{utcc} : sec$.

$$
\begin{array}{ll}
(1) & A \to B \,:\, \{m, A\}_{pub(B)} \\
(2) & B \to A \,:\, \{m, n, B\}_{pub(A)} \\
(3) & A \to B \,:\, \{n\}_{pub(B)}
\end{array}
$$

Figure 4.5: Needham-Schröeder-Lowe protocol with public-key encryption

$$
\begin{array}{lcl}
Init(A, B, k_A, p_B) & = & \mathbf{new}(m)\,\mathbf{out}(\{m, A\}_{p_B}). \\
& & \quad \mathbf{in}_{\forall} x[\{m, x, B\}_{pub(k_A)}]_{priv(k_A)}. \\
& & \quad\ \mathbf{out}(\{x\}_{p_B}).\,\mathbf{nil} \\
\\
Res(A, B, k_B, p_A) & = & \mathbf{in}_{\forall} y[\{y, A\}_{pub(k_B)}]_{priv(k_B)}. \\
& & \quad \mathbf{new}(n)\,\mathbf{out}(\{y, n, B\}_{p_A}). \\
& & \quad\ \mathbf{in}_{\forall}[\{n\}_{pub(k_B)}]_{priv(k_B)}.\mathbf{nil} \\
\\
System(A, B) & = & \mathbf{new}(k_A)\,\mathbf{new}(k_B)\,(Init(A, B, k_A, pub(k_B)) \\
& & \quad \| \, Res(A, B, k_B, pub(k_A)))
\end{array}
$$

Figure 4.6: NSL protocol in SPCCP

Proof. (Sketch) The proof proceeds by trivial induction on the definition of $[P]_{\text{utcc}}$ and the typing rules in Figure 4.2 □

Example: Needham-Schröeder-Lowe with public-key encryption In Figure 4.5 we recall the protocol steps of the Needham-Schröeder-Lowe protocol [Lowe 1995] (herewith referred as NSL) used as example in [Crazzolara & Winskel 2001].

The NSL protocol describes the interaction between agents A and B. First A sends to B a nonce along its agent name, encrypted with B's public key. Then B decrypts the message with his own private key extracting A's nonce. Next, B sends a message to A containing the proof of reception along with a fresh name encrypted under A's public key. Finally, A decrypts B's message and sends to B the name challenge received in the previous message encrypted with B's public key. The SPCCP version of the protocol is given in Figure 4.6.

SPCCP share some similarities with other languages for the description of security protocols, in particular with the approaches in LySA^{NS} [Buchholtz *et al.* 2004], SCCP , and the SPL calculus. Particularly, observe that there is no need to explicitly define the communication channels in which agents are transmitting messages. The underlying model acts as an open network in which every actor can access all the messages posted provided that he has the proper keys to decrypt its the message. We assume a disclosure of public keys for every agent, while the private keys are kept secret for each principal. The key difference between the approach in SPCCP to the approaches in SPL and SCCP is that the abstraction of the contents of a message encrypted with a key is only allowed if one possesses the corresponding key for decryption. This is similar to the approach in the LySA^{NS} calculus, except that we

$$
\begin{aligned}
Init(A,B,k_A,p_B) = &(\textbf{local}\ m)\ \textbf{tell}\,(o(\{\langle m, A\rangle\}_{p_B}) \\
&\Big|\Big|\,\textbf{next}\,(((\lambda\ x; o(priv(k_A)) \Rightarrow (o(\{m, x, B\}_{pub(k_A)}) \wedge o(x)))\ ) \\
&\Big|\Big|\,\textbf{next}\,(\textbf{tell}\,(o(\{x\}_{pub_B}))\Big|\Big|\,\textbf{next}\,(\textbf{skip})))
\end{aligned}
$$

$$
\begin{aligned}
Res(A,B,k_B,p_A) = &(\lambda\ y; o(priv(k_B)) \Rightarrow (o(\{y, A\}_{pub(k_B)}) \wedge o(y))) \\
&\Big|\Big|\,\textbf{next}\,(((\textbf{local}\ n)\ \textbf{tell}\,(o(\{y, n, B\}_{p_A}))) \\
&\Big|\Big|\,\textbf{next}\,((\lambda\ \emptyset; o(priv(k_B)) \Rightarrow (o(\{n\}_{pub(k_B)})))\Big|\Big|\,\textbf{next}\,(\textbf{skip})))
\end{aligned}
$$

$$
\begin{aligned}
System(A,B) = &(\textbf{local}\ k_A)\ (\textbf{local}\ k_B)\ Init(A,B,k_A,pub(k_B)) \\
&\Big|\Big|\,Res(A,B,k_B,pub(k_A))
\end{aligned}
$$

Figure 4.7: NSL protocol: Translation into $utcc_s$

employ the constraint system and local knowledge instead of tailoring the pattern matching with a notion of key pairs. Figure 4.7 exemplifies the translation into $utcc_s$.

4.5 Discusion and Future Work

We have illustrated that the introduction of universal quantification to CCP for modeling mobile communication and security protocols introduce the problem that information which should be local can be obtained by universal quantification. As a way to remedy the problems we have proposed a simple type system for constraints used as patterns in abstractions which allows us to guarantee semantically that e.g. channel names and encrypted values are only extracted by agents that are able to infer the channel or non-encrypted value from the store. Furthermore, we proposed a novel kind of abstraction allowing abstraction under the assumption of local knowledge. The latter can be applied to infer the plain text of encrypted messages under the assumption of knowledge of the key, without adding the key permanently to the global store. We exemplified the type system by examples of mobility of local links (in the context of the π-calculus) and provided a new language for security protocols combining the key features of the Security CCP (SCCP) language and the SPL calculus, but adding the ability to syntactically constraining the ability to decrypting secret values inspired by the LySA^{NS} calculus.

The present work is only in its first stage. However, we believe that the proposed distinction between variables that can be universally quantified and variables that can not is an elegant way to remedy the problems we have illustrated connected to the universal quantification to CCP. A next step will be to perform a detailed investigation of the proposed new variant of the SCCP calculus and applications to

model security protocols. Here we delve into initial ideas for analysing security attacks using SCCP .

4.5.1 Further comments on Secrecy

A secrecy property under utcc should consider the existence of an attacker with the capabilities of previously defined in [Dolev & Yao 1981]. First of all, it is necessary to provide a persistent network model, with the possibility of injecting noisy messages when desired. This behaviour can be represented as:

$$Network =! \left(\begin{array}{l} \mathbf{new}(x)\,\mathbf{out}(x).\,\mathbf{nil} \\ \quad ||\,\mathbf{in}_{\forall} N[M]_{pk(System)}.\mathbf{out}(N).\,\mathbf{nil} \end{array} \right) \tag{4.8}$$

In this model, the network can always create new noise and output for others. It can also use the public knowledge of the system (the set of public session keys) along with the security constraint system to decompose mesages, and output the resulting messages back to the environment. Along with the security constraint system given in figure 4.4, and the definition of network model, an attacker will be able to use its pattern matching capabilities along with the set of keys publicly available as well as his own set of session keys in his search for a given message s, as defined below:

Definition 4.5.1 (Spy). *A spy searching for key s in* SPCCP *is defined as the process*

$$Spy(s,a) = \left(\; Network \;||\; !\mathbf{in}_{\forall} s[M]_{keys(Spy)\wedge pk(System)}.\mathbf{out}(a).\,\mathbf{nil} \; \right) \tag{4.9}$$

Where $keys$ is the set of keys for a given agent and $pk(System)$ the set of public keys of the system.

The Spy in this way has prior knowledge of the kind s of message he is searching for (not of his contents). After a successful decription of s in the pattern M, he will output a message acknowledging success similar to the tests in [De Nicola & Hennessy 1984].

Formally, we will use the notion of input-output equivalences [Valencia 2002] in order to describe an operational view of secrecy:

Definition 4.5.2 (Observable behaviour). *Given $\alpha = c_1.c_2.\ldots$ and $\alpha = c'_1.c'_2.\ldots$. We write $P \xrightarrow{(\alpha,\alpha')} \ldots$ for $P = P_1 \xrightarrow{(c_1,c'_1)} P_2 \xrightarrow{(c_2,c'_2)} \ldots$. The set $io(P) = \{(\alpha,\alpha') | P \xrightarrow{(\alpha,\alpha')} \ldots\}$ denotes the input-output behaviour of P. Two processes P and Q are input-output equivalent ($\sim_{io}$) if $io(P) = io(Q)$. Moreover, we say that P eventually publishes c, written $P \Downarrow c$, if $P \xrightarrow{(tt,\alpha')} \ldots$ and $\alpha'(i) = c$ for some $i \geq 0$. Finally, we say that P discloses c', written $P \downarrow c'$, if $P \Downarrow c$ and $c \vdash c'$.*

Therefore, a security property can use the notion of equivalence presented above to describe an attack. Intuitively, a security attack can be shown if a protocol specification interacting with a network model is different than the same protocol interacting with a spy, in the style of [Abadi & Gordon 1999]. In the case of secrecy, the security

property will be granted iff for any system plugged in parallel with a spy, there is no attacks involving the leakage of a secret. Formally:

Definition 4.5.3 (Secrecy, operationally). *Process P is defined to have a secrecy attack under key s if $\exists a$ s.t. $P \,||\, Network \not\sim_{io} P \,||\, Spy(s,a)$ and $P \,||\, Spy(s,a) \downarrow a$.*

We can exploit the correspondence of utcc processes with linear temporal logic formulae to specify security properties in a logical way. First lets recall the logical correspondence definition in [Olarte & Valencia 2008a].

Theorem 4.5.4 (FLTL Characterization and Logic Correspondence[Olarte & Valencia 2008a]). *Let $[\cdot]$ a map from* utcc *processes to FLTL formulae given by:*

$$
\begin{array}{lcrlcr}
[\mathbf{skip}] & = & \mathrm{tt} & [\mathbf{tell}\,(\mathrm{c})] & = & \mathrm{c} \\
[(\lambda\ \vec{x}; \mathrm{c})\ P] & = & \forall \vec{x}\,(\mathrm{c} \Rightarrow [\mathrm{P}]) & [P \,||\, Q] & = & [P] \wedge [Q] \\
[(\mathbf{local}\ \vec{x}; \mathrm{c})\ P] & = & \exists \vec{x}\,(\mathrm{c} \wedge [\mathrm{P}]) & [\mathbf{next}\,(P)] & = & \circ[P] \\
[\mathbf{unless}\ \mathrm{c}\ \mathbf{next}\ P] & = & \mathrm{c} \vee \circ[\mathrm{P}] & [!P] & = & \Box[P]
\end{array}
$$

Moreover, If P is a well-terminated process and c *a state formula then*

$$P \Vdash \diamond c \quad \textit{iff} \quad P \Downarrow \mathrm{c} \tag{4.10}$$

We can then reformulate a secrecy attack in terms of a temporal property, such as:

Definition 4.5.5 (Secrecy, logically). *A secrecy attack is shown to be present in a process P under the secret s if $\exists a$ s.t. $P \wedge Spy(s,a) \Vdash \diamond a$.*

Finally, we sketch a relation between the secrecy attack evicenced operationally and the exhibition of a message using the logical equivalence. We conjecture that $P \,||\, Network \not\sim_{io} P \,||\, Spy(s,a)$ iff $P \wedge Spy(s,a) \Vdash \diamond a$ for any given s.

4.5.2 Future Work

It is important to remark the importance of the current proposal with respect to other analysis techniques for security protocols. In [Blanchet 2001], a framework for the analysis of secrecy properties is proposed with logic programming as its underlying mechanism. The specification language follows the line of the equational theory presented in the Applied π-calculus [Abadi & Fournet 2001], encoding constructor and destructor functions by means of deduction rules in the framework. Here, pattern-matching is being used to encode the abilities of an attacker to abstract away information from the facts present in the store. Given that the attacker can apply the set of rules in a given specification, the correctness of the analysis relies on the power we give on the inference system. For instance, a rule attacker(sign(m, sk)) $\rightarrow$ attacker(sk) could be specified and the attacker would be able to extract away the secret key from a signature. We believe that a type system similar to the one proposed in this paper can be applied here to limit the extra expressive power of the rule-based approach by

allowing only to abstract only variables over unrestricted parts of the predicates, ruling out the example given above by declaring *sk* a restricted variable over sign(m, sk). Similar considerations can be applied to other systems that base their analysis on pattern-matching techniques, like the extended strand-space approach in [Corin & Etalle 2002] and Miller's linear logic approach for security protocols [Miller 2003].

As also pointed out in the text the local operator of utcc does not really correspond to the generation of new names in nominal calculi. This has already been noticed by Palamidessi et al. [Palamidessi *et al.* 2006], where a logical characterization of name restriction using the existential quantifier does not ensure uniqueness in the fragment of the π-calculus with mismatch. The same occurs in utcc: a process (**local** x) (**local** y) P can hide both x and y from the store, but the current logical formulation does not ensure the uniqueness of x and y, as one may wish when dealing with nonces for security protocols. We leave for future work to study variants of the local operator ensuring uniqueness.

Acknowlegments The authors would like to thank the anonymous reviewers for their suggestions for improvement of this paper. We would also like to thank Jorge A. Pérez for his comments on earlier versions of this document. This research has been partially supported by the Trustworthy Pervasive Healthcare Services (TrustCare) project. Danish Research Agency, Grant # 2106-07-0019 (www.TrustCare.eu).

Chapter 5

A Logic for Choreographies

Abstract: We explore logical reasoning for the global calculus, a coordination model based on the notion of choreography, with the aim to provide a methodology for specification and verification of structured communications. Starting with an extension of Hennessy-Milner logic, we present the global logic ($\mathcal{GL}$), a modal logic describing possible interactions among participants in a choreography. We illustrate its use by giving examples of properties on service specifications. Finally, we show that, despite $\mathcal{GL}$ is undecidable, there is a significant decidable fragment which we provide with a sound and complete proof system for checking validity of formulae.

Contents

5.1 Introduction

Due to the continuous growth of technologies, software development is recently shifting its focus on communication, giving rise to various research efforts for proposing new methodologies dealing with higher levels of complexity. A new software paradigm, known as *choreography*, has emerged with the intent to ease programming of communication-based protocols. Intuitively, a choreography is a description of the global flow of execution of a system where the software architect just describes which and in what order interactions *can* take place. This idea differs from the standard approach where the communication primitives are given for each single entity separately. A good illustration can be seen in the way a soccer match is planned: the

coach has an overall view of the team, and organises (a priori) how players will interact in each play (the rôle of a choreography); once in the field, each player performs his role by interacting with each of the members of his team by throwing/receiving passes. The way each player synchronise with other members of the team represents the rôle of an orchestration.

The work in [Carbone *et al.* 2006] formalises the notion of choreography in terms of a calculus, dubbed the *global calculus*, which pinpoints the basic features of the choreography paradigm. Although choreography provides a good abstraction of the system being designed allowing to *forget* about common problems that can arise when programming communication (e.g. races over a channel), it can still have complex structures hence being often error prone. Additionally, choreography can be non-flexible in early design stages where the architect might be interested in designing only parts of a system as well as specifying only parts of a protocol (e.g. initial and final interactions). In this view, we believe that a logical approach can allow for more modularity in designing systems e.g. providing partial specification of a system using the choreography paradigm.

In order to illustrate the approach proposed in this work, let us consider an online booking scenario. On one side, consider an airline company AC which offers flights directly from its website. On the other side, there is a customer looking for the best offers. We can informally describe the interaction protocol in terms of a sequence of allowed interactions (as in a choreography) as follows:

1. Customer establishes a communication with AC;
2. Customer asks AC for a flight proposal given a set of constraints;
3. AC establishes a communication with partner AC' serving the destination asked by the costumer;
4. AC forwards the request made by the customer;
5. AC' sends an offer to AC;
6. AC forwards the offer to the customer

Note that each step above represents a communication. In the same way that a choreographical specification describes each of the interactions between participants, a logical characterisation of choreographies denotes formulae describing the evolution of such interactions. However, a logical characterisation gives extra flexibility to the specification of interactions: When writing a logical property describing specific communication patterns we focus on describing **only** the sequence of *key interactions*, leaving room for implementations that include extra behaviour that does not compromise the fulfilment of the property. For instance, in the above example, one can describe a property leaving out the details on the forward of the request to the airline partner, in a statement like *"given an interaction between the customer and AC featuring a booking request, then there is an eventual response directed to the customer with an offer matching the original session"* (in this case, the offer is not necessarily from the airline originally contacted but from one of its partners).

In this document, we provide a link between choreographies and logics. Starting with an extension of Hennessy-Milner logic [Hennessy & Milner 1980], we provide the syntax and the semantics of a logic for the global calculus as well as several examples

of choreographical properties. On decidability issues, we found out that the whole set of the logic is undecidable on the global calculus with recursion. As a result, we focus our studies in a decidable fragment, providing a proof system that allows for property verification of choreographies and show that it is sound and complete, in the sense that all and only valid formulae specified in the global logic can be provable in the proof system. Moreover, we can conclude that the proof checking algorithm using this proof system is terminating.

Overview of the document First, in Section 5.2 we recall the formal foundations of the global calculus, and equip it with a labelled transition semantics. A logic characterisation of the calculus and several examples of the use of the logic are presented in Section 5.3. We proceed with the study of undecidability for the logic in Section 5.4, and a proof system relating the logical characterisation and the global calculus for a decidable fragment of the language is presented in Section 5.5. Finally, concluding remarks are presented in Section 5.6.

5.2 The Global Calculus

The Global Calculus (GC) [Carbone *et al.* 2006, Carbone *et al.* 2007] originates from the Web Service Choreography Description Language (WS-CDL) [Kavantzas *et al.* 2004], a description language for web services developed by W3C. Terms in GC describe choreographies as interactions between participants by means of message exchanges. The description of such interactions is centred on the notion of a *session*, in which two interacting parties first establish a private connection via some public channel and then interact through it, possibly interleaved with other sessions. More concretely, an interaction between two parties starts by the creation of a fresh session identifier, that later will be used as a private channel where meaningful interactions take place. Each session is fresh and unique, so each communication activity will be clearly separated from other interactions. In this section, we provide an operational semantics for GC in terms of a label transition systems (LTS) [Plotkin 1981] describing how global descriptions evolve, and relate to the type discipline that describes the structured sequence of message exchanges between participants from [Carbone *et al.* 2007].

5.2.1 Syntax

Let $\mathcal{C}, \mathcal{C}', \ldots$ denote *terms* of the calculus, often called *interactions* or *choreographies*; $A, B, C, \ldots$ range over *participants*; $k, k', \ldots$ are *linear channels*; $a, b, c, \ldots$ *shared channels*; $v, w, \ldots$ variables; $X, Y, \ldots$ process variables; $l, l_i, \ldots$ labels for branching; and finally $e, e', \ldots$ over unspecified arithmetic and other first-order expressions. We write $e@A$ to mean that the expression e is evaluated using the variable related to participant A in the store.

Definition 5.2.1. *The syntax of the global calculus [Carbone et al. 2006] is given by the following grammar:*

$$
\begin{array}{rll}
\mathcal{C} ::= & \mathbf{0} & \text{(inaction)} \\
\mid & A{\to}B{:}a(k).\ \mathcal{C} & \text{(init)} \\
\mid & A{\to}B : k\langle e, y\rangle.\ \mathcal{C} & \text{(com)} \\
\mid & A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I} & \text{(choice)} \\
\mid & \mathcal{C}_1 \mid \mathcal{C}_2 & \text{(par)} \\
\mid & \textbf{if } e@A \textbf{ then } \mathcal{C}_1 \textbf{ else } \mathcal{C}_2 & \text{(cond)} \\
\mid & X & \text{(recvar)} \\
\mid & \mu X.\ C & \text{(recursion)}
\end{array}
$$

Intuitively, the term (inaction) denotes a system where no interactions take place. (init) denotes a session initiation by A via B's service channel a, with a fresh session channel k and continuation $\mathcal{C}$. Note that k is bound in $\mathcal{C}$. (com) denotes an in-session communication of the evaluation (at A's) of the expression e over a session channel k. In this case, y does not bind in $\mathcal{C}$ (our semantics will treat y as a variable in the store of B). (choice) denotes a labelled choice over session channel k and set of labels I. In (par), $\mathcal{C}_1 \mid \mathcal{C}_2$ denotes the parallel product between $\mathcal{C}_1$ and $\mathcal{C}_2$. (cond) denotes the standard conditional operator where $e@A$ indicates that the expression e has to be evaluated in the store of participant A. In (recursion), $\mu X.\ \mathcal{C}$ is the minimal fix point operation for recursion, where the variable X of (recvar) is bound in $\mathcal{C}$. The free and bound session channels and term variables are defined in the usual way. The calculus is equipped with a standard structural congruence $\equiv$, defined as the minimal congruence relation on interactions $\mathcal{C}$, such that $\equiv$ is a commutative monoid with respect to $\mid$ and $\mathbf{0}$, it is closed under alpha equivalence $\equiv_\alpha$ of terms, and it is closed under the recursion unfolding, i.e., $\mu X.\mathcal{C} \equiv \mathcal{C}[\mu X.\mathcal{C}/X]$.

Remark 5.2.2 (Differences with the approach in [Carbone *et al.* 2007]). Excluding the lack of local assignment, we argue that this monadic version of GC is, to some extent, as expressive as the one Global Calculus originally reported in [Carbone *et al.* 2007]. In particular, note that $A{\to}B : k\langle \mathrm{op}, e, y\rangle$ in [Carbone *et al.* 2007] captures both selection and message passing which are instead disentangled in our case (mainly for clarity reasons). The absence of op in the interaction process $A{\to}B : k\langle e, y\rangle$ can be easily encoded with the existing operators. In fact, $\Sigma_{i\in I} A{\to}B : k\langle op_i, e, y\rangle.\ \mathcal{C}'_i$ can be decomposed into $A{\to}B{:}k[op_i : \mathcal{C}''_i]_{i\in I}$ where $\mathcal{C}''_i = A{\to}B : k\langle e, y\rangle.\ \mathcal{C}'_i$ (although we lose atomicity).

5.2.2 Semantics

We give the operational semantics in terms of configurations $(\sigma, \mathcal{C})$, where σ represents the state of the system and $\mathcal{C}$ the choreography actually being executed. The state σ contains a set of variables labelled by participants. As described in the previous subsection, a variable x located at participant A is written as $x@A$. The

$$\text{(G-Init)}\ \frac{h \text{ fresh}}{(\sigma, A{\to}B{:}a(k).\ \mathcal{C}) \xrightarrow{\mathsf{init}\ A{\to}B\ \mathsf{on}\ a(h)} (\sigma, \mathcal{C}[h/k])}$$

$$\text{(G-Com)}\ \frac{\sigma(e@A) \Downarrow v}{(\sigma, A{\to}B : k\langle e, x\rangle.\ \mathcal{C}) \xrightarrow{\mathsf{com}\ A{\to}B\ \mathsf{over}\ k} (\sigma[x@B \mapsto v], \mathcal{C})}$$

$$\text{(G-Choice)}\ \frac{}{(\sigma, A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}) \xrightarrow{\mathsf{sel}\ A{\to}B\ \mathsf{over}\ k{:}l_i} (\sigma, \mathcal{C}_i)}$$

$$\text{(G-Par)}\ \frac{(\sigma, \mathcal{C}_1) \xrightarrow{\ell} (\sigma', \mathcal{C}_1')}{(\sigma, \mathcal{C}_1 \mid \mathcal{C}_2) \xrightarrow{\ell} (\sigma', \mathcal{C}_1' \mid \mathcal{C}_2)}$$

$$\text{(G-Struct)}\ \frac{\mathcal{C} \equiv \mathcal{C}' \quad (\sigma, \mathcal{C}') \xrightarrow{\ell} (\sigma', \mathcal{C}'') \quad \mathcal{C}'' \equiv \mathcal{C}'''}{(\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}''')}$$

$$\text{(G-IfT)}\ \frac{\sigma(e@A) \Downarrow \mathsf{tt} \quad (\sigma, \mathcal{C}_1) \xrightarrow{\ell} (\sigma', \mathcal{C}_1')}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2\ ) \xrightarrow{\ell} (\sigma', \mathcal{C}_1')}$$

$$\text{(G-IfF)}\ \frac{\sigma(e@A) \Downarrow \mathsf{ff} \quad (\sigma, \mathcal{C}_2) \xrightarrow{\ell} (\sigma', \mathcal{C}_2')}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2\ ) \xrightarrow{\ell} (\sigma', \mathcal{C}_2')}$$

Table 5.1: Operational Semantics for the Global Calculus

same variable name labelled with different participant names denotes different variables (hence $\sigma(x@A)$ and $\sigma(x@B)$ may differ). Formally, the operational semantics is defined as a labelled transition system (LTS). A transition $(\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')$ says that a choreography $\mathcal{C}$ in a state σ executes an action (or label) ℓ and evolves into $\mathcal{C}'$ with a new state σ'. Actions are defined as $\ell = \{\mathsf{init}\ A \to B\ \mathsf{on}\ a(k), \mathsf{com}\ A \to B\ \mathsf{over}\ k, \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l_i\}$, denoting initiation, in-session communication and branch selection, respectively. We write $(\sigma, \mathcal{C}) \longrightarrow (\sigma', \mathcal{C}')$ when ℓ irrelevant, and $\longrightarrow^*$ denotes the transitive closure of $\longrightarrow$. The transition relation $\longrightarrow$ is defined as the minimum relation on pairs state/interaction satisfying the rules in Table 5.1.

Intuitively, transition (G-Init) describes the evolution of a session initiation: after A initiates a session with B on service channel a, A and B share the fresh channel h locally. (G-Com) describes the main interaction rule of the calculus: the expression e is evaluated into v in the A-portion of the state σ and then assigned to the variable x located at B resulting in the new state $\sigma[x@B \mapsto v]$. (G-Choice) chooses the evolution of a choreography resulting from a labelled choice over a session key k. (G-IfT) and (G-IfF) show the possible paths that a deterministic evolution of a choreography can

produce. (G-Par) and (G-Struct) behave as the standard rules for parallel product and structural congruence, respectively.

Remark 5.2.3 (Global Parallel). Parallel composition in the global calculus differs from the notion of parallel found in standard concurrency models based on input/output primitives [Milner 1999]. In the latter, a term $P_1 \mid P_2$ may allow *interactions* between P_1 and P_2. However, in the global calculus, the parallel composition of two choreographies $\mathcal{C}_1 \mid \mathcal{C}_2$ concerns two parts of the described system where *interactions* may occur in $\mathcal{C}_1$ and $\mathcal{C}_2$ but never across the parallel operator $\mid$. This is because an interaction $A \to B \ldots$ abstracts from the actual end-point behaviour, i.e., how A sends and B receives. In this model, dependencies between two choreographies can be expressed by using variables in the state σ.

In its original presentation [Carbone *et al.* 2007], GC comes equipped with a reduction semantics unlike the one presented in Table 5.1. Our LTS semantics has the advantage of allowing to observe changes on the behaviour of the system, which will prove useful when relating to the logical characterisation in Section 5.3. We conjecture that our proposed LTS semantics and the reduction semantics of the global calculus originally presented in [Carbone *et al.* 2007] coincide (taking into account the considerations in Remark 5.2.2).

Example 5.2.4 (Online Booking). We consider the example presented in the introduction, i.e., a simplified version of the on-line booking scenario presented in [López *et al.* 2010]. Here, the customer (Cust) establishes a session with the airline company (AC) using service (on-line booking, shorted as ob) and creating the session key k_1. Once the session is established, the customer will request the company about a flight offer with his booking data, along the session key k_1. The airline company will process the customer request and, after requesting another airline company (AC') for the service, will send a reply back with an offer. The customer will eventually accept the offer, sending back an acknowledgment to the airline company using k_1. The following specification in the GC represents the protocol:

$$\begin{aligned}\mathcal{C}_{\mathrm{OB}} = {} & \mathrm{Cust}{\to}\mathrm{AC}{:}\mathrm{ob}(k_1).\ \mathrm{Cust}{\to}\mathrm{AC} : k_1\langle \mathrm{booking}, x\rangle.\ \mathrm{AC}{\to}\mathrm{AC}'{:}\mathrm{ob}(k_2). \\ & \mathrm{AC}{\to}\mathrm{AC}' : k_2\langle \mathrm{x}, x'\rangle.\ \mathrm{AC}'{\to}\mathrm{AC} : k_2\langle \mathrm{offer}, y\rangle.\ \mathrm{AC}{\to}\mathrm{Cust} : k_1\langle \mathrm{y}, y''\rangle.\ \mathrm{Cust}{\to}\mathrm{AC} : k_1\langle \mathrm{accept}, z\rangle.\ \mathbf{0}\end{aligned} \tag{OB}$$

5.2.3 Session Types for the Global Calculus

We use a generalisation of session types [Honda *et al.* 1998] for global interactions, first presented in [Carbone *et al.* 2007]. Session types in GC are used to structure sequence of message exchanges in a session. Their syntax is as follows:

$$\alpha = \uparrow(\theta).\alpha \mid \downarrow(\theta).\alpha \mid \&\{l_i : \alpha_i\}_{i\in I} \mid \oplus\{l_i : \alpha_i\}_{i\in I} \mid \mathrm{end} \mid \mu\mathbf{t}.\ \alpha \mid \mathbf{t} \tag{5.1}$$

where $\theta, \theta', \ldots$ range over value types $\mathrm{bool}, \mathrm{string}, \mathrm{int}, \ldots$. $\alpha, \alpha', \ldots$ are session types. The first four types are associated with the various communication operations. $\downarrow(\theta).\alpha$ and $\uparrow(\theta).\alpha$ are the input and output types respectively. Similarly, $\&\{l_i : \alpha_i\}_{i\in I}$

$$
\begin{array}{rcll}
\phi, \chi & ::= & \exists t.\ \phi & \text{(f-exists)} \\
& | & \phi \wedge \chi & \text{(f-and)} \\
& | & \neg \phi & \text{(f-neg)} \\
& | & \langle \ell \rangle \phi & \text{(f-action)} \\
& | & \mathsf{end} & \text{(f-termination)} \\
& | & e_1@A = e_2@B & \text{(f-equality)} \\
& | & \phi \mid \chi & \text{(f-parallel)} \\
& | & \Diamond \phi & \text{(f-may)}
\end{array}
\qquad
\begin{array}{rcll}
\ell & ::= & \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) & \text{(l-init)} \\
& | & \mathsf{com}\ A \to B\ \mathsf{over}\ k & \text{(l-com)} \\
& | & \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l & \text{(l-branch)}
\end{array}
$$

Table 5.2: $\mathcal{GL}$: Syntax of formulae

is the branching type while $\oplus\{l_i : \alpha_i\}_{i \in I}$ is the selection type. The type end indicates session termination and is often omitted. $\mu \mathbf{t}.\ \alpha$ indicates a recursive type with $\mathbf{t}$ as a type variable. $\mu \mathbf{t}.\ \alpha$ binds the free occurrences of $\mathbf{t}$ in α. We take an *equi-recursive* view on types, not distinguishing between $\mu \mathbf{t}.\ \alpha$ and its unfolding $\alpha[\mu \mathbf{t}.\ \alpha/\mathbf{t}]$.

A typing judgment has the form $\Gamma \vdash \mathcal{C} : \Delta$, where Γ, Δ are *service type* and *session type* environments, respectively. Typically, Γ contains a set of type assignments of the form $a@A : \alpha$, which says that a service a located at participant A may be invoked and run a session according to type α. Δ contains type assignments of the form $k[A, B] : \alpha$ which says that a session channel k identifies a session between participants A and B and has session type α when seen from the viewpoint of A. The typing rules are omitted, and we refer to [Carbone *et al.* 2009] for the full account of the type discipline noting that the observations made in Remark 5.2.2 will require extra typing rules.

Returning to the specification (OB) in Example 5.2.4, the service type of the airline company AC at channel *ob* can be described as:

$$\mathsf{ob@AC} : (k_1, k_2)\ k_1 \downarrow \mathsf{booking(string)}.\ k_2 \uparrow \mathsf{x(string)}.\ k_2 \downarrow \mathsf{offer(int)}.\ k_1 \uparrow \mathsf{y(int)}.\ k_1 \downarrow \mathsf{accept(int)}.\ \mathsf{end}\,.$$

Assumption 5.2.5. In the sequel, we only consider choreographies that satisfy the typing discipline.

5.3 $\mathcal{GL}$: A Logic for the Global Calculus

In this section, we introduce a logic for choreographies, inspired by the modal logic for session types presented in [Berger *et al.* 2008]. The logical language comprises assertions for equality and value/name passing.

5.3.1 Syntax

The grammar of assertions is given in Table 5.2. Choreography assertions (ranged over by $\phi, \phi', \chi, \ldots$) give a logical interpretation of the global calculus introduced

in the previous section. The logic includes the standard First Order Logic (FOL) operators $\wedge$, $\neg$, and $\exists$. In $\exists t.\ \phi$, the variable t is meant to range over service and session channels, participants, labels for branching and basic placeholders for expressions. Accordingly, it works as a binder in ϕ. In addition to the standard operators, the operator (f-action) represents the execution of a labelled action ℓ followed by the assertion ϕ. Those labels in ℓ match the ones in the LTS of GC, i.e., they are (l-init), (l-com), and (l-branch). The formula (f-termination) represents the process termination. We also include an unspecified, but decidable, (f-equality) operator on expressions as in [Berger *et al.* 2008]. (f-may) denotes the standard eventually operators from Linear Temporal Logic (LTL) [Emerson 1991]. The spatial operator (f-parallel) denotes composition of formulae: because of the unique nature of parallel composition in choreographies, we have used the symbol $\mid$ (as in separation logic [Reynolds 2002] and spatial logic [Caires & Cardelli 2001]) in order to stress the fact that there is no interference between two choreographies running in parallel.

Notation 5.3.1 (Existential quantification over action labels). In order to simplify the readability, we introduce the concept of existential quantification over action labels as a short-cut to mean the following:

$$\begin{aligned}\exists \ell.\ \langle \ell \rangle \phi \stackrel{\text{def}}{=}\ & \exists A, B, a, k.\ \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle.\ \phi \vee \\ & \exists A, B, k.\ \langle \mathsf{com}\ A \to B\ \mathsf{over}\ k \rangle.\ \phi \vee \\ & \exists A, B, k, l.\ \langle \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l \rangle.\ \phi\,.\end{aligned}$$

Remark 5.3.2 (Derived Operators). We can get the full account of the logic by deriving the standard set of strong modalities from the above presented operators. In particular, we can encode the constant true (tt) and false (ff), the next ($\circ\phi$) and the always operators ($\Box\phi$) from LTL.

$$\begin{array}{lll} \mathsf{tt} \stackrel{\text{def}}{=} (0@A = 0@A) & \mathsf{ff} \stackrel{\text{def}}{=} (0@A = 1@A) & (e_1 \neq e_2) \stackrel{\text{def}}{=} \neg(e_1 = e_2) \\ \forall x.\ \phi \stackrel{\text{def}}{=} \neg\exists x.\ \neg\phi & \phi \vee \chi \stackrel{\text{def}}{=} \neg(\neg\phi \wedge \neg\chi) & \phi \Rightarrow \chi \stackrel{\text{def}}{=} \neg\phi \vee \chi \\ \Box\phi \stackrel{\text{def}}{=} \neg\Diamond\neg\phi & [\ell]\phi \stackrel{\text{def}}{=} \neg\langle\ell\rangle\neg\phi & \circ\phi \stackrel{\text{def}}{=} \exists \ell.\ \langle \ell \rangle \phi\,. \end{array}$$

In the rest of this section, we illustrate the expressiveness of our logic through a sequence of simple, yet illuminating examples, giving an intuition of how the modalities introduced plus the existential operator $\exists$ allow to express properties of choreographies.

Example 5.3.3 (Availability, Service Usage and Coupling). The logic above allows to express that, given a service invoker (known as A in this setting) requesting the service a, there exists another participant (called B in the example) providing a with A invoking it. This can be formulated in $\mathcal{GL}$ as follows:

$$\exists B.\ \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle \mathsf{tt}\,.$$

Example 5.3.5 (Response Abstraction).

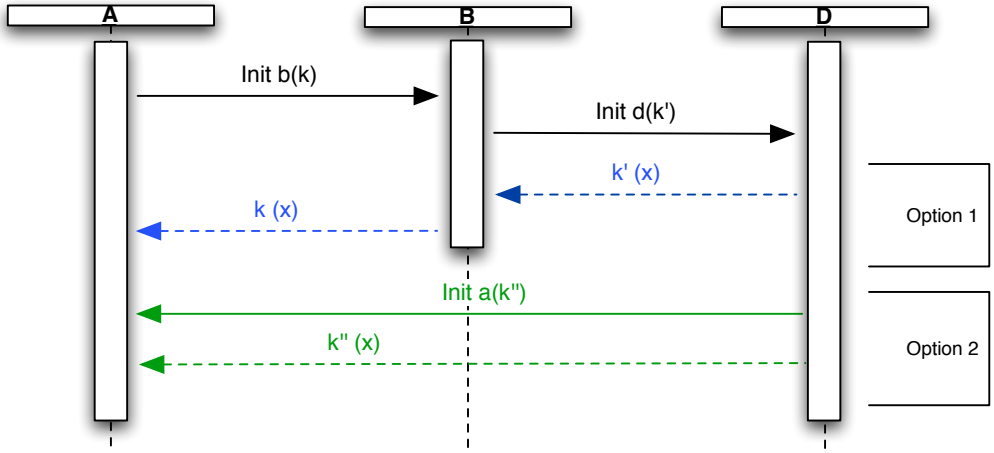


Figure 5.1: Diagram of a partial specification.

Assume now, that we want to ensure that services available are actually used. We can use the dual property for availability, i.e., for a service provider B offering a, there exists someone invoking a:

$$\exists A.\ \langle \textsf{init}\ A \to B\ \textsf{on}\ a(k)\rangle \mathsf{tt}\,.$$

Verifying that there is a service pairing two different participants in a choreography can be done by existentially quantifying over the shared channels used in an initiation action. A formula in $\mathcal{GL}$ representing this can be the following one:

$$\exists a.\ \langle \textsf{init}\ A \to B\ \textsf{on}\ a(k)\rangle \mathsf{tt}\,.$$

Example 5.3.4 (Causality Analysis). The modal operators of the logic can be used to perform studies of the causal properties that our specified choreography can fulfil. For instance, we can specify that given an expression e evaluated to true at participant A, there is an eventual firing of a choreography that satisfies property ϕ_1, whilst ϕ_2 will never be satisfied. Such a property can be specified as follows:

$$(e@A = \mathsf{tt}) \wedge \Diamond(\phi_1) \wedge \Box\neg\phi_2\,.$$

An interesting aspect of our logic is that it allows for the declaration of partial specification properties regarding the interaction of the participants involved in a choreography. Take for instance the interaction diagram in Figure 5.1. The participant A invokes service b at B's and then B invokes D's service d. At this point, D can send the content of variable x to A in two different ways: either by using those originally established sessions or by invoking a new service at A's. However, at the end of both computation paths, variable z (located at A's) will contain the value of x. In the global

calculus, this two optional behaviour can be modelled as follows:

$$C_1 = A{\to}B{:}b(k).\ B{\to}D{:}d(k').\ D{\to}B : k'\langle x, y_B\rangle.\ B{\to}A : k\langle y_B, z\rangle.\ \mathbf{0} \qquad \text{(Option 1)}$$
$$C_2 = A{\to}B{:}b(k).\ B{\to}D{:}d(k').\ D{\to}A{:}a(k'').\ D{\to}A : k''\langle x, z\rangle\ .\ \mathbf{0}\,. \qquad \text{(Option 2)}$$

We argue that, under the point of view of A, both options are sufficiently good if, after an initial interaction with B is established, there is an eventual response that binds variable z. Such a property can be expressed by the $\mathcal{GL}$ formula:

$$\exists X, k''.\ \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k)\rangle\Diamond\Big(\langle \mathsf{com}\ X \to A\ \mathsf{over}\ k''\rangle(z@A = x@D)\Big).\ \mathsf{end}\,.$$

Notice that both the choreographies (Option 1) and (Option 2) *satisfy* the partial specification above. This will be clear in Section 5.3.2 where we introduce the semantics of logic.

Also note that a third option for the protocol at hand is to use *delegation* (the ability of communicating session keys to third participants not involved during session initiation). However, the current version of the global calculus does not feature such an operation and we leave it as future work.

Example 5.3.6 (Connectedness). The work in [Carbone *et al.* 2007] proposes a set of criteria for guaranteeing a safe end-point projection between global and local specifications (note that the choreography in the previous example does not respect such properties). Essentially, a valid global specification has to fulfil three different criteria, namely Connectedness, Well-threadedness and Coherence. It is interesting to see that some of these criteria relate to global and local causality relations between the interactions in a choreography, and can be easily formalised as properties in the choreography logic presented here. Below, we consider the notion of connectedness and leave the other cases as future work. Connectedness dictates a global causality principle among interactions: any two consecutive interactions $\ldots A \to B.\ C \to D \ldots$ in a choreography are such that $B = C$. In the following, let $\mathsf{Interact}(A, B)\phi$ be true whenever $\langle\ell\rangle\phi$ holds for some ℓ with an interaction from A to B. Connectedness can be specified as:

$$\forall A, B.\ \Box\Big(\mathsf{Interact}(A, B)\mathsf{tt} \Rightarrow \exists C.\ \big(\mathsf{Interact}(A, B)\mathsf{Interact}(B, C)\mathsf{tt} \vee \mathsf{Interact}(A, B)\neg\exists\ell\langle\ell\rangle\mathsf{tt}\big)\Big)\,.$$

5.3.2 Semantics

We now give a formal meaning to the assertions introduced above with respect to the semantics of the global calculus introduced in the previous section. In particular, we introduce the notion of satisfaction. We write $\mathcal{C} \models_\sigma \phi$ whenever a state σ and a choreography $\mathcal{C}$ satisfy a $\mathcal{GL}$ formula ϕ. The relation $\models_\sigma$ is defined by the rules given in Table 5.3. In the $\exists t.\ \phi$ case, w should be an appropriate value according to the type of t, e.g., a participant if t is a participant placeholder.

Definition 5.3.7 (Satisfiability, Validity and Logical Equivalence).

$$
\begin{array}{lcl}
\mathcal{C} \models_\sigma \text{end} & \stackrel{\text{def}}{=} & \mathcal{C} \equiv \mathbf{0} \\
\mathcal{C} \models_\sigma (e_1@A = e_2@B) & \stackrel{\text{def}}{=} & \sigma(e_1@A) \Downarrow v \text{ and } \sigma(e_2@B) \Downarrow v \\
\mathcal{C} \models_\sigma \langle \ell \rangle \phi & \stackrel{\text{def}}{=} & (\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}') \text{ and } \mathcal{C}' \models_{\sigma'} \phi \\
\mathcal{C} \models_\sigma \phi \wedge \chi & \stackrel{\text{def}}{=} & \mathcal{C} \models_\sigma \phi \text{ and } \mathcal{C} \models_\sigma \chi \\
\mathcal{C} \models_\sigma \neg \phi & \stackrel{\text{def}}{=} & \mathcal{C} \not\models_\sigma \phi \\
\mathcal{C} \models_\sigma \exists t.\ \phi & \stackrel{\text{def}}{=} & \mathcal{C} \models_\sigma \phi[w/t] \text{ (for some appropriate } w) \\
\mathcal{C} \models_\sigma \Diamond \phi & \stackrel{\text{def}}{=} & (\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}') \text{ and } \mathcal{C}' \models_{\sigma'} \phi \\
\mathcal{C} \models_\sigma \phi \mid \chi & \stackrel{\text{def}}{=} & \mathcal{C} \equiv \mathcal{C}_1 \mid \mathcal{C}_2 \text{ such that } \mathcal{C}_1 \models_\sigma \phi \text{ and } \mathcal{C}_2 \models_\sigma \chi
\end{array}
$$

Table 5.3: Assertions of the Choreography Logic

- *A formula ϕ is* satisfiable *if there exists some configuration under which it is true, that is, $\mathcal{C} \models_\sigma \phi$ for some $(\mathcal{C}, \sigma)$.*
- *A formula ϕ is* valid *if it is true in every configuration, that is, $\mathcal{C} \models_\sigma \phi$ for every $(\sigma, \mathcal{C})$.*
- *A formula χ is a* logical consequence *of a formula ϕ (or ϕ* logically implies *χ), denote with an abuse of notation as $\phi \models \chi$, if every configuration $(\sigma, \mathcal{C})$ that makes ϕ true also makes χ true.*
- *We say that a formula ϕ is* logical equivalent *to a formula χ, written $\phi \equiv_\models \chi$, if $\phi \models \chi$ iff $\chi \models \phi$.*

5.4 Undecidability of Global Logic

In this section we focus on the undecidability of the global logic for the global calculus with recursion given in Section 5.2. In order to prove that the global logic is undecidable, we use a reduction from the Post Correspondence Problem (PCP) [Post 1944] similarly to the one proposed in [Charatonik & Talbot 2001]. The idea is to encode in the global calculus a "program" which simulates the construction of PCP. We first give a formal definition of the PCP. In the sequel, $\cdot$ denotes word concatenation.

Definition 5.4.1 (PCP). *Let $s, t, \ldots$ range over Σ^* where $\Sigma = \{0, 1\}$ and let ε be the empty word. An instance of PCP is a set of pairs of words $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ over $\Sigma^* \times \Sigma^*$. The Post Correspondence Problem is to find a sequence $i_0, i_1, \ldots, i_k$ ($1 \leq i_j \leq n$ for all $0 \leq j \leq k$) such that $s_{i_0} \cdot \ldots \cdot s_{i_k} = t_{i_0} \cdot \ldots \cdot t_{i_k}$.*

Intuitively, PCP consists of finding some string in Σ^* which can be obtained by the concatenation $s_{i_0} \cdot \ldots \cdot s_{i_k}$ as well as by $t_{i_0} \cdot \ldots \cdot t_{i_k}$. Such a problem has been proved to be undecidable [Post 1944]. Our goal is to find a GC term that takes a

random pair of words from an instance of PCP and append them to an "incremental pair" of words which encodes the current state of the sequences $s_{i_0} \cdot \ldots \cdot s_{i_k}$ and $t_{i_0} \cdot \ldots \cdot t_{i_k}$. Technically, we need a choreography that assigns randomly a natural number in $\{1, \ldots, n\}$ to a variable r in some participant B, and another choreography that picks a pair of words from the PCP instance, accordingly to value in the variable $r@B$, and then appends them to the "incremental pair" of words in A. Formally,

Definition 5.4.2 (Encoding of PCP). *Let $A_1, \ldots, A_n, A, B$ be participants and let a, b be shared names for sessions, then define the two choreographies as shown below:*

$$\begin{aligned} \mathit{Random}(A_1, \ldots, A_n, B, a) \stackrel{def}{=} \quad & \mu X.\ A_1 \to B{:}a(k).\ A_1 \to B : k\langle 1, r\rangle.\ X \\ \mid\ & \mu X.\ A_2 \to B{:}a(k).\ A_2 \to B : k\langle 2, r\rangle.\ X \\ \mid\ & \ldots \\ \mid\ & \mu X.\ A_n \to B{:}a(k).\ A_n \to B : k\langle n, r\rangle.\ X \end{aligned}$$

$$\begin{aligned} \mathit{Append}(A, B, b) \stackrel{def}{=}\ & \mu X.\ A \to B{:}b(k).\ A \to B : k\langle str1, tmp1\rangle.\ A \to B : k\langle str2, tmp2\rangle. \\ & \textbf{if } r@B = 1 \textbf{ then} \\ & \quad B \to A : k\langle tmp1 \cdot s_1, str1\rangle.\ B \to A : k\langle tmp2 \cdot t_1, str2\rangle.\ X \\ & \textbf{else if } r@B = 2 \textbf{ then} \\ & \quad B \to A : k\langle tmp1 \cdot s_2, str1\rangle.\ B \to A : k\langle tmp2 \cdot t_2, str2\rangle.\ X \\ & \textbf{else if } r@B = 3 \textbf{ then} \\ & \quad \vdots \\ & \textbf{else if } r@B = n \textbf{ then} \\ & \quad B \to A : k\langle tmp1 \cdot s_n, str1\rangle.\ B \to A : k\langle tmp2 \cdot t_n, str2\rangle.\ X \\ & \textbf{else } X \end{aligned}$$

We define the initial configuration $(\sigma, \mathcal{C})$ to be formed by the choreography and the state below:

$$\mathcal{C} \stackrel{def}{=} \mathit{Random}(A_1, \ldots, A_n, B, a) \mid \mathit{Append}(A, B, b)$$

$$\sigma \stackrel{def}{=} [str1@A \mapsto \varepsilon,\ str2@A \mapsto \varepsilon,\ tmp1@B \mapsto \varepsilon,\ tmp2@B \mapsto \varepsilon,\ r@B \mapsto 1].$$

For encoding the PCP existence question ($s_{i_0} \cdot \ldots \cdot s_{i_k} = t_{i_0} \cdot \ldots \cdot t_{i_k}$) we can encode it as a $\mathcal{GL}$ formula:

$$\phi \stackrel{def}{=} \Diamond\Big((str1@A = str2@A) \wedge (str1@A \neq \varepsilon) \wedge (str2@A \neq \varepsilon)\Big).$$

Above, each participant A_i (with $i \in \{1, \ldots, n\}$) recursively opens a session with participant B and writes in the variable $r@B$ the value i. Moreover, the participant B stores the knowledge of all the word pairs (s_i, t_i), while the participant A takes randomly a word pair from B and then append it to his incremental pair of words: $(str1, str2)$. Next, the formula ϕ states that there exists a computational path from the initial configuration to a configuration which stores in $str1$ and $str2$ two equal non-empty strings.

Theorem 5.4.3. *The global logic is undecidable on the global calculus with recursion.*

Proof. (Sketch) The statement $\mathcal{C} \models_\sigma \phi$ holds iff the encoded PCP has a solution. Indeed, if the initial configuration $(\sigma, \mathcal{C})$ satisfies the formula ϕ then it means there exists a configuration $(\sigma', \mathcal{C}')$ where $(str1@A = str2@A) \wedge (str1@A \neq \varepsilon) \wedge (str2@A \neq \varepsilon)$ holds. Hence, there is a sequence of $i_0, \ldots, i_k$ such that $str1 = s_{i_0} \cdot \ldots \cdot s_{i_k} = t_{i_0} \cdot \ldots \cdot t_{i_k} = str2$, that is, the instance of PCP has a solution. □

Remark 5.4.4. The undecidability result presented in this section shows that the global calculus is considerably expressive, despite the choreography approach offers a simplification in the specification of concurrent communicating systems as argued in [Carbone *et al.* 2007]. The encoding in Definition 5.4.2 shows that allowing state variables (hence local variables that can be accessed by various threads) increases the expressive power of the language. Indeed, we could just look at GC as a simple concurrent language with a "shared" store where assignment to variables is just in-session communication. In this view, we conjecture that removing variables and focusing only on communication would make the logic decidable.

5.5 Proof System for Recursion-free Choreographies

In this section, we present a model checking algorithm (in the form of a proof system) to decide whether a global logic formula is satisfied by a recursion-free configuration of the global calculus. Indeed, similarly to [Charatonik & Talbot 2001], it turns out that the logic is decidable on the recursion-free choreographies.[1] We also prove the soundness and completeness of the proposed proof system w.r.t. the assertion semantics.

In order to reason about judgments $\mathcal{C} \models_\sigma \phi$, we propose a proof (or inference) system for assertions of the form $\mathcal{C} \vdash_\sigma \phi$. Intuitively, we want $\mathcal{C} \vdash_\sigma \phi$ to be as approximate as possible to $\mathcal{C} \models_\sigma \phi$ (ideally, they should be equivalent). We write $\mathcal{C} \vdash_\sigma \phi$ for the provability judgement where $(\sigma, \mathcal{C})$ is a configuration and ϕ is a formula.

Notation 5.5.1. We define the set of continuations configuration after an action ℓ and the reachable configurations, both starting from a configuration $(\sigma, \mathcal{C})$, as follows:

$$\mathsf{Next}(\sigma, \mathcal{C}, \ell) \stackrel{\text{def}}{=} \{(\sigma', \mathcal{C}') \mid (\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')\}$$
$$\mathsf{Reachable}(\sigma, \mathcal{C}) \stackrel{\text{def}}{=} \{(\sigma', \mathcal{C}') \mid (\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')\}\,.$$

Normalisation is required by the proof system to infer equality of choreographies up to structural equivalence (Especially for the $[\cdot] \mid [\cdot]$ operator). We define $\mathsf{Norm}(\mathcal{C})$ to be a normalisation function from recursion-free choreographies into multi-sets of

[1] Removing recursion yields a decidability result orthogonal to the conjecture formulated in Remark 5.4.4

choreographies:

$$\mathsf{Norm}(A{\to}B : k\langle e, y\rangle.\ \mathcal{C}) \stackrel{\text{def}}{=} [A{\to}B : k\langle e, y\rangle.\ \mathcal{C}]$$

$$\mathsf{Norm}(A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}) \stackrel{\text{def}}{=} [A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}]$$

$$\mathsf{Norm}(A{\to}B{:}a(k).\ \mathcal{C}) \stackrel{\text{def}}{=} [A{\to}B{:}a(k).\ \mathcal{C}]$$

$$\mathsf{Norm}(\textbf{if } e@A \textbf{ then } \mathcal{C}_1 \textbf{ else } \mathcal{C}_2\,) \stackrel{\text{def}}{=} [\textbf{if } e@A \textbf{ then } \mathcal{C}_1 \textbf{ else } \mathcal{C}_2\,]$$

$$\mathsf{Norm}(\mathbf{0}) \stackrel{\text{def}}{=} [\,]$$

$$\mathsf{Norm}(\mathcal{C}_1 \mid \mathcal{C}_2) \stackrel{\text{def}}{=} [P_1, \ldots, P_n, Q_1, \ldots, Q_m] \quad \text{if} \quad \begin{array}{l} \mathsf{Norm}(\mathcal{C}_1) = [P_1, \ldots, P_n] \quad \text{and} \\ \mathsf{Norm}(\mathcal{C}_2) = [Q_1, \ldots, Q_m] \quad . \end{array}$$

Lemma 5.5.2 (Normalisation preserves structural equivalence). *Let $\mathcal{C}$ be a recursion-free choreography and $\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n]$, then $\mathcal{C} \equiv \prod_{i=1}^{n} P_i$.*

Proof. By induction on the structure of the choreography $\mathcal{C}$.

Case $\mathcal{C} = \mathbf{0}$: We have $\mathsf{Norm}(\mathbf{0}) = [\,]$, and $\prod_{i=1}^{0} P_i = \mathbf{0} \equiv \mathbf{0}$.

Case $\mathcal{C} = \mathcal{C}_1 \mid \mathcal{C}_2$: We have that $\mathsf{Norm}(\mathcal{C}_1) = [P_1, \ldots, P_n]$, $\mathsf{Norm}(\mathcal{C}_2) = [Q_1, \ldots, Q_m]$, and $\prod_{i=1}^{n} P_i \equiv \mathcal{C}_1$, $\prod_{j=1}^{m} Q_j \equiv \mathcal{C}_2$ by induction hypothesis. Then, we can derive that $\prod_{i=1}^{n} P_i \mid \prod_{j=1}^{m} Q_j \equiv \mathcal{C}_1 \mid \mathcal{C}_2$.

All the other cases: Trivially we have that $\mathsf{Norm}(\mathcal{C}) = [P_1]$, where $P_1 = \mathcal{C}$, then $\prod_{i=1}^{1} P_i \equiv \mathcal{C}$. □

Definition 5.5.3 (Entailment). *We say that a choreography $\mathcal{C}$* entails *a formula ϕ under a state σ, written $\mathcal{C} \vdash_\sigma \phi$, iff the assertion $\mathcal{C} \vdash_\sigma \phi$ has a proof in the proof system given in Table 5.4.*

Let us now describe some of the inference rules of the proof system. The rule $\mathsf{P}_{\mathsf{end}}$ relates the inaction terms with the termination formula. The rules $\mathsf{P}_{\mathsf{and}}$ and $\mathsf{P}_{\mathsf{neg}}$ denote rules for conjunction and negation in classical logic, respectively. The rule for parallel composition is represented in $\mathsf{P}_{\mathsf{par}}$; it does not indicate the behaviour of a given choreography, but hints information about the structure of the process: $\mathsf{P}_{\mathsf{par}}$ juxtaposes the behaviour of two processes and combines their respective formulae by the use of a separation operator. The next rule, $\mathsf{P}_{\mathsf{action}}$ requires that the process P in the configuration σ can perform an action labelled ℓ, so we must search for a continuations of $(\sigma, \mathcal{C})$ after an action ℓ and find a configuration which satisfies the rest of the formula, i.e., ϕ. Analogously, $\mathsf{P}_{\mathsf{may}}$ looks for a continuation in the reachable configuration of $(\sigma, \mathcal{C})$ in oder to satisfy ϕ. The rule $\mathsf{P}_{\exists}$ says that in order to satisfy an $\exists t.\ \phi$, it is sufficient to find a value w for t in the free names used by the choreography $\mathcal{C}$ or in the free names used by the formula ϕ. Finally, the rule $\mathsf{P}_{\mathsf{exp}}$ denotes evaluation of expressions.

We now proceed to prove the soundness of the proof system with respect to the semantics of assertions presented before.

$$\frac{}{\mathcal{C} \vdash_\sigma \text{end}}\ \mathsf{P}_{\text{end}} \quad \mathsf{Norm}(\mathcal{C}) = [\,] \qquad \frac{\mathcal{C} \vdash_\sigma \phi \quad \mathcal{C} \vdash_\sigma \chi}{\mathcal{C} \vdash_\sigma \phi \wedge \chi}\ \mathsf{P}_{\text{and}} \qquad \frac{\mathcal{C} \not\vdash_\sigma \phi}{\mathcal{C} \vdash_\sigma \neg\phi}\ \mathsf{P}_{\text{neg}}$$

$$\frac{\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n] \quad \exists I, J.\ I \cup J = \{1, \ldots, n\} \wedge I \cap J = \emptyset \wedge \prod_{i \in I} P_i \vdash_\sigma \phi_1 \wedge \prod_{j \in J} P_j \vdash_\sigma \phi_2}{\mathcal{C} \vdash_\sigma \phi_1 \mid \phi_2}\ \mathsf{P}_{\text{par}}$$

$$\frac{\exists (\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell).\ \mathcal{C}' \vdash_{\sigma'} \phi}{\mathcal{C} \vdash_\sigma \langle \ell \rangle \phi}\ \mathsf{P}_{\text{action}} \qquad \frac{\exists (\sigma', \mathcal{C}') \in \mathsf{Reachable}(\sigma, \mathcal{C}).\ \mathcal{C}' \vdash_{\sigma'} \phi}{\mathcal{C} \vdash_\sigma \Diamond \phi}\ \mathsf{P}_{\text{may}}$$

$$\frac{\exists w \in fn(\mathcal{C}) \cup fn(\phi).\ \mathcal{C} \vdash_\sigma \phi[w/t]}{\mathcal{C} \vdash_\sigma \exists t.\ \phi}\ \mathsf{P}_{\exists} \qquad \frac{\sigma(e_1@A) \Downarrow v \quad \sigma(e_2@B) \Downarrow v}{\mathcal{C} \vdash_\sigma (e_1@A = e_2@B)}\ \mathsf{P}_{\text{exp}}$$

Table 5.4: Proof system for the Global Calculus.

Lemma 5.5.4 (Structural congruence preserves satisfability). *If $\mathcal{C} \equiv \mathcal{C}'$ and $\mathcal{C} \models_\sigma \phi$, then $\mathcal{C}' \models_\sigma \phi$.*

Proof. (Sketch) It follows from structural induction over ϕ. □

Theorem 5.5.5 (Soundness). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, if $\mathcal{C} \vdash_\sigma \phi$ then $\mathcal{C} \models_\sigma \phi$.*

Proof. It follows by induction on the derivation of $\vdash_\sigma$.

Case P_{end}: Straight consequence of Lemmas 5.5.2 and 5.5.4, indeed $\mathcal{C} \equiv \mathbf{0}$ and $\mathcal{C} \models_\sigma$ end.

Case P_{and}: By induction hypothesis and conjunction.

Case P_{neg}: We have that $\mathcal{C} \vdash_\sigma \neg\phi$, so by P_{neg} we get $\mathcal{C} \not\vdash_\sigma \phi$. By induction hypothesis we have that $\mathcal{C} \not\models_\sigma \phi$, which is the necessary condition to deduce $\mathcal{C} \models_\sigma \neg\phi$.

Case P_{par}: We have that $\mathcal{C} \vdash_\sigma \phi_1 \mid \phi_2$, then $\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n]$, and there exist I, J such that $I \cup J = \{1, \ldots, n\}$, $I \cap J = \emptyset$, $\prod_{i \in I} P_i \vdash_\sigma \phi_1$, and $\prod_{j \in J} P_j \vdash_\sigma \phi_2$. By induction hypothesis we know that $\prod_{i \in I} P_i \models_\sigma \phi_1$ and $\prod_{j \in J} P_j \models_\sigma \phi_2$, then by Lemma 5.5.2 we have $\mathcal{C} \equiv \prod_{i \in I} P_i \mid \prod_{j \in J} P_j$, hence it is immediate to prove that $\mathcal{C} \models_\sigma \phi_1 \mid \phi_2$.

Case $\mathsf{P}_{\text{action}}$: We have that $\mathcal{C} \vdash_\sigma \langle \ell \rangle \phi$ and by $\mathsf{P}_{\text{action}}$ then $\mathcal{C}' \vdash_{\sigma'} \phi$ and $(\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell)$. From the induction hypothesis we have that $\mathcal{C}' \models_{\sigma'} \phi$, then we have to show that $\mathcal{C} \models_\sigma \langle \ell \rangle \phi$. From the assertion semantics we know that $C \models_\sigma \langle \ell \rangle \phi$ iff $(\sigma, \mathcal{C}') \xrightarrow{\ell} (\sigma', \mathcal{C}')$ and $\mathcal{C}' \models_{\sigma'} \phi$, which holds immediately by the selection of $(\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell)$ and the induction hypothesis.

Case $\mathsf{P}_{\mathsf{may}}$: We have that $\mathcal{C} \vdash_\sigma \Diamond\phi$ and by $\mathsf{P}_{\mathsf{may}}$ then $\mathcal{C}' \vdash_{\sigma'} \phi$ and $(\sigma', \mathcal{C}') \in \mathsf{Reachable}(\sigma, \mathcal{C})$. From the induction hypothesis we have that $\mathcal{C}' \models_{\sigma'} \phi$, then we have to show that $\mathcal{C} \models_\sigma \Diamond\phi$. From the assertion semantics we know that $C \models_\sigma \Diamond\phi$ iff $(\sigma, \mathcal{C}') \longrightarrow^* (\sigma', \mathcal{C}')$ and $\mathcal{C}' \models_{\sigma'} \phi$, which holds immediately by the selection of $(\sigma', \mathcal{C}') \in \mathsf{Reachable}(\sigma, \mathcal{C})$ and the induction hypothesis.

Case $\mathsf{P}_\exists$: We have that $\mathcal{C} \vdash_\sigma \exists t.\phi$ and by $\mathsf{P}_\exists$ we have that $\exists w \in fn(\mathcal{C}) \cup fn(\phi)$ and $\mathcal{C} \vdash_\sigma \phi[w/t]$. By induction hypothesis we know that $C \models_\sigma \phi[w/t]$ with appropriate $w \in fn(\mathcal{C}) \cup fn(\phi)$, then $\mathcal{C} \models_\sigma \exists t.\phi$ follows from the definition of the assertion semantics.

Case $\mathsf{P}_{\mathsf{exp}}$: It holds trivially by checking if $\sigma(e_1@A) \Downarrow v$ and $\sigma(e_2@B) \Downarrow v$. □

Lemma 5.5.6. *For every configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion free, and every formula $\exists t.\ \phi$, if $\{n_1, \ldots, n_k\} = fn(\mathcal{C}) \cup fn(\phi)$, then $\mathcal{C} \models_\sigma \exists t.\ \phi$ iff $\exists m \in \{n_1, \ldots, n_k\}$ such that $\mathcal{C} \models_\sigma \phi[m/t]$.*

Proof. (Sketch) By induction on the structure of ϕ. It is similar to the proof of [Cardelli & Gordon 2000, Lemma 5.3(3)]. □

Theorem 5.5.7 (Completeness). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, if $\mathcal{C} \models_\sigma \phi$ then $\mathcal{C} \vdash_\sigma \phi$.*

Proof. By rule induction on the derivation of $\models_\sigma$.

Case $\mathcal{C} \models_\sigma$ end: We have that $\mathcal{C} \equiv \mathbf{0}$ and hence $\mathsf{Norm}(\mathcal{C}) = [\,]$ by Lemma 5.5.2. Now, the thesis follows immediately from the application of $\mathsf{P}_{\mathsf{end}}$.

Case $\mathcal{C} \models_\sigma (e_1@A = e_2@B)$: It follows immediately by the application of $\mathsf{P}_{\mathsf{exp}}$.

Case $\mathcal{C} \models_\sigma \langle\ell\rangle\phi'$: Take $(\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')$ and $\mathcal{C}' \models_{\sigma'} \phi'$, we have by induction hypothesis that $\mathcal{C}' \vdash_{\sigma'} \phi'$. Now, we have to show that $\mathcal{C} \vdash_\sigma \langle\ell\rangle\phi'$. By the fact that $(\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')$, we have that $(\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell)$, hence, we can apply rule P_{action} and we are done.

Case $\mathcal{C} \models_\sigma \phi \wedge \chi$: We have that $\mathcal{C} \models_\sigma \phi$ and $\mathcal{C} \models_\sigma \chi$. From the induction hypothesis we have that $\mathcal{C} \vdash_\sigma \phi$ and $\mathcal{C} \vdash_\sigma \chi$. The application of $\mathsf{P}_{\mathsf{and}}$ lead to $\mathcal{C} \vdash_\sigma \phi \wedge \chi$ as desired.

Case $\mathcal{C} \models_\sigma \neg\phi$: From the definition of the assertion semantics we have that $\mathcal{C} \models_\sigma \neg\phi$ iff $\mathcal{C} \not\models_\sigma \phi$. We have to show that $\mathcal{C} \vdash_\sigma \neg\phi$. We proceed by contradiction. Take a $(\phi, \mathcal{C})$ such that $\mathcal{C} \vdash_\sigma \phi$, then from Theorem 5.5.5 we have that $\mathcal{C} \models_\sigma \phi$, which is a contradiction to $\mathcal{C} \models_\sigma \neg\phi$.

Case $\mathcal{C} \models_\sigma \exists t.\ \phi$: We have that $\mathcal{C} \models_\sigma \exists t.\phi$ and by the definition in the assertion semantics we have that $\mathcal{C} \models_\sigma \phi[w/t]$ for an appropriate w. By induction hypothesis we know that $\mathcal{C} \vdash_\sigma \phi[w/t]$. Lemma 5.5.6 guarantees that there exists $w \in fn(\mathcal{C}) \cup fn(\phi)$ in order to derive $\mathcal{C} \vdash_\sigma \exists t.\phi$ from $\mathsf{P}_\exists$.

Case $\mathcal{C} \models_\sigma \Diamond\phi$: Take $(\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')$ and $\mathcal{C}' \models_{\sigma'} \phi'$, we have by induction hypothesis that $\mathcal{C}' \vdash_{\sigma'} \phi'$. Now, we have to show that $\mathcal{C} \vdash_\sigma \Diamond\phi'$. By the fact that $(\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')$, we have that $(\sigma', \mathcal{C}') \in \mathsf{Reachable}(\sigma, \mathcal{C})$, hence, we can apply rule P_{may} and we are done.

Case $\mathcal{C} \models_\sigma \phi \mid \chi$: We have that $\mathcal{C} \equiv \mathcal{C}_1 \mid \mathcal{C}_2$ and $\mathcal{C}_1 \models_\sigma \phi \wedge \mathcal{C}_2 \models_\sigma \chi$. From the induction hypothesis $\mathcal{C}_1 \vdash_\sigma \phi$ and $\mathcal{C}_2 \vdash_\sigma \chi$. Now by Lemma 5.5.2 we have that $\mathcal{C}_1 \equiv \prod_{i\in I} P_i$ and $\mathcal{C}_2 \equiv \prod_{j\in J} P_j$ for some I, J. So, we can derive $\mathcal{C} \equiv \prod_{i\in I} P_i \mid \prod_{j\in J} P_j$, and hence $\mathsf{P}_{\mathsf{par}}$ leads to $\mathcal{C}_1 \mid \mathcal{C}_2 \vdash_\sigma \phi \mid \chi$. □

Theorem 5.5.8 (Termination). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, proof-checking algorithm terminates.*

Proof. First, notice that all the functions **Norm**, **Next**, and **Reachable** are total and computable. The proof is by induction over the structure of ϕ.

Case $\phi = \mathsf{end}$: $\mathcal{C} \vdash_\sigma \mathsf{end}$ iff $\mathsf{Norm}(\mathcal{C}) = [\,]$.

Case $\phi = \phi_1 \wedge \phi_2$: By conjunction and induction hypothesis on $\mathcal{C} \vdash_\sigma \phi_1$ and $\mathcal{C} \vdash_\sigma \phi_2$.

Case $\phi = \neg\phi'$: $\mathcal{C} \vdash_\sigma \phi$ iff $\mathcal{C} \vdash_\sigma \phi'$ does not hold. But by induction hypothesis we can construct a terminating proof or confutation for $\mathcal{C} \vdash_\sigma \phi'$. Hence the proof for $\mathcal{C} \vdash_\sigma \phi$ terminates as well.

Case $\phi = \phi_1 \mid \phi_2$: Suppose $\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n]$. Notice that there exists a finite number of possible partitioning of $\{1, \ldots, n\}$ in I, J. Hence, for every I, J we can compute $\prod_{i\in I} P_i \vdash_\sigma \phi_1$ and $\prod_{j\in J} P_j \vdash_\sigma \phi_2$, which both terminate by induction hypothesis. By applying Lemma 5.5.2 we prove the thesis.

Case $\phi = \langle\ell\rangle\phi'$: First, notice that the set $\mathsf{Next}(\sigma, \mathcal{C}, \ell)$ is finite, because the choreographies are finite, i.e., there are a finite number of actionable transition in a given configuration. For each configuration $(\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell)$, $\mathcal{C}' \vdash_{\sigma'} \phi'$ terminates by induction hypothesis.

Case $\phi = \Diamond\phi'$: As before, notice that the set $\mathsf{Reachable}(\sigma, \mathcal{C})$ is finite, because the choreographies are finite, i.e., the choreographies are recursion free. For each configuration $(\sigma', \mathcal{C}') \in \mathsf{Reachable}(\sigma, \mathcal{C})$, $\mathcal{C}' \vdash_{\sigma'} \phi'$ terminates by induction hypothesis.

Case $\phi = \exists t.\ \phi'$: To prove existence is sufficient to check every derivation by substituting t with a name $w \in fn(\mathcal{C}) \cup fn(\phi)$. Notice that $fn(\mathcal{C}) \cup fn(\phi)$ is finite, because both $\mathcal{C}$ and ϕ are so. So, for every w, we can construct a terminating derivation for $\mathcal{C} \vdash_\sigma \phi'[w/t]$ by induction hypothesis.

Case $\phi = (e_1@A = e_@@B) : \mathcal{C} \vdash_\sigma (e_1@A = e_@@B)$ iff $e_1@A \Downarrow v$ and $e_@@B \Downarrow v$. □

5.6 Conclusion and Related Work

The ideas hereby presented constitutes just the first step towards a verification framework for choreography. As a future work, our main concerns relate to integrate our framework into other end-point models and logical frameworks for the specification of sessions. In particular, our next step will focus on relating the logic to the end-point projection [Carbone *et al.* 2007], the process of automatically generating end-point code from choreography. Other improvements to the system proposed include the use of fixed points, essential for describing state-changing loops, and auxiliary axioms describing structural properties of a choreography.

This work can be fruitfully nourished by related work in types and logics for session-based communication. In [López *et al.* 2010] the authors proposed a mapping between the calculus of structured communications and concurrent constraint programming, allowing them to establish a logical view of session-based communication and formulae in First-Order Temporal Logic. In [Berger *et al.* 2008], Berger et al. presented proof systems characterising May/Must testing pre-orders and bisimilarities over typed π-calculus processes. The connection between types and logics in such system comes in handy to restrict the shape of the processes one might be interested, allowing us to consider such work as a suitable proof system for the calculus of end points. Finally, [Montangero & Semini 2006] studies a logic for choreographies in a model without services and sessions while [Bocchi *et al.* 2010] proposes notion of global assertion for enriching multiparty session types with simple formula describing changing in the state of a session.

Acknowledgements This research has been partially supported by the Trustworthy Pervasive Healthcare Services (TrustCare) and the Computer Supported Mobile Adaptive Business Processes (Cosmobiz) projects. Danish Research Agency, Grants # 2106-07-0019 (www.TrustCare.eu) and # 274-06-0415 (www.cosmobiz.org).

Chapter 6

Modal Logics for Structured Communications

Abstract: We present a framework integrating imperative and declarative views for structured communications. Starting from languages for the specification of services, we provide a modal logic characterization of the interactions ocurrying in a system, both at a from a global standpoint and from the views of each participant. The framework copes with two aims: exhibiting logical guarantees about the presence of an interaction, and model generation from logical specifications[1].

Contents

[1] This work is an extended version of Chapter 5. In particular, sections 6.2 – 6.4 contain simplified versions of the results in Chapter 5, and readers can refer to such chapter for its full explanation.

6.1 Introduction

Given the intrinsic complexity when analysing services in distributed environments, one normally use different abstractions to describe and analyse services. One of such abstractions deals with the the study of the concurrent nature of services. Process calculi are formal languages conceived for the description and analysis of concurrent systems. As such, the goal of a process calculus is to provide a rigorous framework where complex systems can be accurately analysed, including reasoning techniques (e.g.: type systems, specification logics) to verify essential properties about their behaviour. The term structured communications [Honda *et al.* 1998] refers to the branch of process calculi devoted to the analysis of interactions between services. On a calculus for structured communications, one considers the computation within a service as an atomic activity, and focus the core of the analysis in the interactions between services.

Despite of being such a young trend, different but interrelated views for the analysis of service oriented systems have been proposed. We can enclose such approaches in two dichotomies: global/local views of services, and imperative/declarative specifications. In the first dichotomy, either one describe the system as the exchange of messages between different participants, or one consider the system as the composition of the local behaviours of each participant. In this first view, known as *choreography* [Kavantzas *et al.* 2004], one considers the system as a whole, taking care only of the interfaces that participants use when interacting to the outside world. In the second view, known as *orchestration* [Misra & Cook 2006], one models the system as perceived by the eyes of each participant (so-called end-point), sending and receiving messages but not knowing which other actors are present in a communication. As recently presented [Carbone *et al.* 2007, Busi *et al.* 2006, Hongli *et al.* 2007], choreographies and orchestrations have close ties to each other, and one can project a choreography to generate distributed orchestrations that implements it, sometimes referred as an *end-point projection*.

The second dichotomy here considered refers to the approach used to construct the models. Descriptions can have imperative or declarative flavours: In an imperative approach, one explicitly defines the control flow of commands. Typical representatives of this approach are based on process calculi, and come with behavioural equivalences and type disciplines as their main analytic tools [Puhlmann & Weske 2005, Lapadula *et al.* 2007a, Boreale *et al.* 2006, Honda *et al.* 1998, Vieira *et al.* 2008]. On the

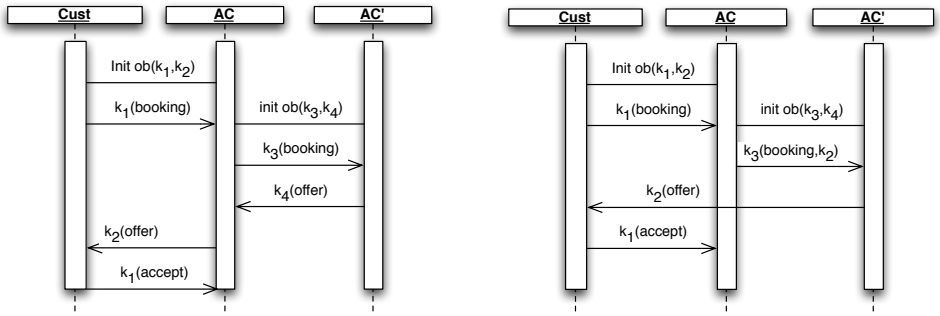
contrary, in a declarative approach the focus drifts to the specification of the set of constraints (causality relations, time constraints, quality of service) processes should fulfil in order to be considered correct [Pesic & van der Aalst 2006, van der Aalst & Pesic 2006, Lyng *et al.* 2008, Nørgaard *et al.* 2005]. Even if these two trends address similar concerns, we find that they have evolved rather independently from each other. Returning to our example, we might consider the specifications above presented imperative specifications, whereas a declarative specification will let parts of the process unspecified.

Contributions Here we present a framework integrating imperative and declarative views for structured communications. Building from previous research in calculi for the specification of services [Carbone *et al.* 2007], we provide modal logic characterisations of the interactions occurring in a system, both at a from a global standpoint and from the views of each participant. The framework cope with two aims: exhibiting logical guarantees about the presence of an interaction, and model generation from logical specifications. In particular, we present two logical languages for describing choreographies and orchestrations. First, $\mathcal{GL}$ is a logic describing possible interactions in a choreographical language (the global calculus): the correspondence between specifications in the calculus and the logic is tight, and one can go either from the logical characterisation to the process algebraic specification of a process, or prove that a choreographical description respects a formula in $\mathcal{GL}$. Second, $\mathcal{LL}$ is a logic inspired in the Hennessy-Milner with the aim of describing properties about the interactions in a language of orchestrations (the end-point calculus). The logic characterises typed bisimulation (pruning). As the main result, we show that there is a correspondence between imperative and declarative descriptions of the projections from choreographies to its end-points.

6.1.1 An Example

The notions of the framework are easily explained through an example describing each of the different visions we integrate.

Let us consider an electronic booking scenario. On one side, consider a company AC which offers flights directly from its website. On the other side, there is a customer looking for the best offers. In this scenario, the customer establishes a communication with AC and asks for a flight proposal given a set of constraints, such as the destination, the dates allowed, etc. After receiving a request and check its validity, AC establishes a communication with its partner AC' serving the destination asked by the customer, and forwards the request made by the customer to AC'. Once that AC' is able to process the request, he can contact the customer and provide him an offer. The ways AC' communicates the offer to the customer are *purposely* left unspecified: for instance, 1) AC' could reply back to AC, that later would contact customer with their previous established session, or 2) AC' receives from AC the session key of the communication established between the customer and AC, and use



(a) Interaction diagram (following classical session types)

(b) Interaction diagram (session types with delegation)

Figure 6.1: Electronic booking example

it to reply back to the customer (*delegation*)[2], or 3) AC' could create a new session with the customer, and send him the offer directly. A graphical specification of the interaction diagrams corresponding to such cases is illustrated in Figure 6.1.

A global specification focuses on the description of the interactions between participants $Cust, AC$ and AC'. For instance, Cust→AC:ob$_{AC}(k_1)$ initiates an interaction between the Customer and the service ob_{AC} located in the Airline company, labelled with a session identifier k_1. Similarly, the communication of the offer message between the Airline partner and the customer will be written as AC'→Cust : $k_2\langle \text{offer}, y\rangle$. Choreographies C_{OB-i} present possible specifications of message exchanges in Figure 6.1, where C_{OB-1} presents the alternative where messages travel back thru AC, C_{OB-2} the alternative creating a new session, and C_{OB-3} the alternative using session delegation. Notice that, in order for models like the ones described in C_{OB-2} and C_{OB-3} to respect the causality and coherence relations between interactions present in the theory of session types, we need languages to be expressive enough to support further capabilities, like delegation [Honda *et al.* 1998] and correlation sets [Lapadula *et al.* 2007a].

$$\begin{aligned} C_{OB-1} = \ & \text{Cust}\to\text{AC:ob}_{AC}(k_1, k_2).\ \text{Cust}\to\text{AC} : k_1\langle \text{booking}, x_1\rangle. \\ & \text{AC}\to\text{AC':ob}_{AC'}(k_3, k_4).\ \text{AC}\to\text{AC'} : k_3\langle x_1, y\rangle. \\ & \text{AC'}\to\text{AC} : k_4\langle \text{offer}, x_2\rangle.\ \text{AC}\to\text{Cust} : k_2\langle x_2, c\rangle. \\ & \text{Cust}\to\text{AC} : k_1\langle \text{accept}, z\rangle \end{aligned} \tag{6.1}$$

$$\begin{aligned} C_{OB-2} = \ & \text{Cust}\to\text{AC:ob}_{AC}(k_1, k_2).\ \text{Cust}\to\text{AC} : k_1\langle \text{booking}, x\rangle. \\ & \text{AC}\to\text{AC':ob}_{AC'}(k_3).\ \text{AC}\to\text{AC'} : k_3\langle k_2, x'\rangle. \\ & \text{AC'}\to\text{Cust} : k_2\langle \text{offer}, y\rangle.\ \text{Cust}\to\text{AC} : k_1\langle \text{accept}, z\rangle \end{aligned} \tag{6.2}$$

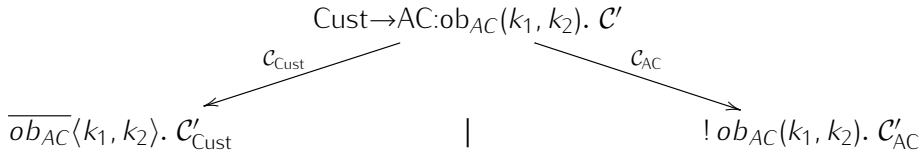
[2]This option, although interesting, will be refrained from consideration in our current study, as the choreography language utilised does not feature delegation of sessions

$$\begin{aligned} C_{OB-3} = \text{Cust}\rightarrow\text{AC:ob}_{AC}(k_1, k_2).\ \text{Cust}\rightarrow\text{AC} : k_1\langle\text{booking}, x_1\rangle.\\ \text{AC}\rightarrow\text{AC':ob}_{AC'}(k_3, k_4).\ \text{AC}\rightarrow\text{AC'} : k_3\langle x_1, y\rangle.\\ \text{AC'}\rightarrow\text{Cust:ob}_{Cust}(k_5, k_6).\ \text{AC'}\rightarrow\text{Cust} : k_5\langle\text{offer}, c\rangle.\\ \text{Cust}\rightarrow\text{AC} : k_1\langle\text{accept}, z\rangle \end{aligned} \tag{6.3}$$

In the same way that a choreographical specification describes each of the interactions between participants, a logical characterisation of choreographies denote formulae describing the evolution of such interactions. However, a logical characterisation gives more flexibility to the specification of interactions: One can forget about the addition of extraneous constructs of the language and define a simple policy about the behaviour of interactions. This policy can be described using logical specification over choreographies. This logical specification describes *only* the important parts of the message flow between participants. For instance, in the above presented specification, one can describe a property ensuring that, given a communication between the Customer and the Airline company with a booking message, there is an eventual response directed to the customer with an offer matching the same session identifier (in this case, not necessarily coming from the same participant the communication was initiated).

$$\begin{aligned} C_{OB-i} \models \exists A, k_r.\ \langle\textsf{init}\ Cust \rightarrow AC\ \textsf{on}\ ob_{AC}(k_1, k_2)\rangle.\ \langle\textsf{com}\ Cust \rightarrow AC\ \textsf{over}\ k_1(\text{booking})\rangle.\\ \Diamond\langle\textsf{com}\ A \rightarrow Cust\ \textsf{over}\ k_r(\text{offer})\rangle \end{aligned} \tag{6.4}$$

In a similar but orthogonal approach, the same specification in equation 6.1 can be seen as processes implementing each participant involved in the choreography. For instance, an interaction $\mathcal{C} = \text{Cust}\rightarrow\text{AC:ob}_{AC}(k_1, k_2).\ \mathcal{C}'$ describing session initiation can be decomposed to concurrent processes $\mathcal{C}_{\text{Cust}} \mid \mathcal{C}_{\text{AC}}$ implementing each side of the interaction:



The full set of projections realising the choreography in equation 6.1 might need to include participants Cust, AC and AC', and will need to guarantee that the ordering of the messages imposed in the global specification is still reflected in the projections. The resulting behaviour can be seen below:

$$\mathcal{C}_{\text{Cust}} = \overline{ob_{AC}}\langle k_1, k_2\rangle.\ k_1!\langle\text{booking}\rangle.\ k_2?(c)\ .\ k_1!\langle\text{accept}\rangle.\ \mathbf{0} \tag{6.5}$$

$$\mathcal{C}_{\text{AC}} = !\, ob_{AC}(k_1, k_2).\ k_1?(x_1)\ .\ \overline{ob_{AC'}}\langle k_3, k_4\rangle.\ k_3!\langle x_1\rangle.\ k_4?(x_2)\ .\ k_2!\langle x_2\rangle.\ k_1?(z)\ .\ \mathbf{0} \tag{6.6}$$

$$\mathcal{C}_{\text{AC'}} = !\, ob_{AC'}(k_3, k_4).\ k_3?(y)\ .\ k_4!\langle\text{offer}\rangle.\ \mathbf{0} \tag{6.7}$$

$$\mathcal{C} = \mathcal{C}_{\text{Cust}} \mid \mathcal{C}_{\text{AC}} \mid \mathcal{C}_{\text{AC'}} \tag{6.8}$$

A declarative vision of end points allows us to express properties regarding each of the participants involved. For instance, we can check that the end point representing the customer respects a property stating that there is an eventual reply back after having made a booking request. Hence, end point specification at Equation 6.5 needs to satisfy the following formula:

$$\mathcal{C}_{\text{Cust}} \models Cust[k_1!\langle \text{booking} \rangle].\ \Diamond\ Cust[k_2?(x)].\ \text{end} \tag{6.9}$$

Where $Cust[\psi]$ denotes the execution of an action ψ by participant $Cust$.

An interesting point here is the relation we can evidence between declarative specifications at global and local viewpoints. In principle, a declarative model describing the behaviour at the level of choreographies needs to be projected to formulae describing the behaviour of their end-points. Let ϕ_{OB} the formula in Equation 6.4, and its decomposition into end-point formula described below:

$$\begin{aligned}
[\phi_{OB}] = \exists A, k_r.(&Cust[ob_{AC} \uparrow k_1, k_2].\ Cust[k_1!\langle \text{booking} \rangle] \qquad (6.10)\\
&.\ (Cust[k_r?(\text{offer})] \vee \Diamond Cust[k_r?(\text{offer})]))\\
&\mid (AC[ob_{AC} \downarrow k_1, k_2].\ AC[k_1?(\text{booking})])\\
&\mid (A[k_r?(\text{offer})] \vee \Diamond A[k_r?(\text{offer})]) \qquad (6.11)
\end{aligned}$$

The formula projected *only* speaks about the projected behaviour from the global specification, and allows for multiple implementations of services to accomplish such specification. We will see later, that although the translation seems intuitive, it is far from trivial: end points can impliement many threads at the same time, and those have to be included in consideration on translations of the logical formulae. Also, the projections considered should be *meaningful*, in the sense that they respect the theory of end point projections. We show in further sections how this is accomplished.

6.1.1.1 Overview of the document

First, In Section 6.2 we recall the reader the formal foundations of a calculus of choreographies, the so-called global calculus. Its respective logic is presented in Section 6.3. A proof system relating the logical characterisation and the global calculus shown in Section 6.4. The semantics for the calculus of end-points is presented in Section 6.5, a logical characterisation of the end-point calculus is presented in Section 6.6, as well as the main contribution of this paper, namely the correspondence between the end-point projection and the logical projection between global and local formulae. Finally, concluding remarks are presented in Section 6.7.

6.2 The Global Calculus

The Global Calculus (GC) [Carbone *et al.* 2006, Carbone *et al.* 2007] originates from the Web Service Choreography Description Language (WS-CDL) [Kavantzas *et al.* 2004],

a description language for web services developed by W3C. Terms in GC describe choreographies as interactions between participants by means of message exchanges. The description of such interactions is centered on the notion of *session*, in which two interacting parties first establish a private connection via some public channel and then interact through it, possibly interleaved with other sessions. More concretely, an interaction between two parties starts by the creation of a fresh session identifier, that later will be used as a private channel where meaningful interactions take place. Each session is fresh and unique, so each communication activity will be clearly separated from other interactions. In this section, we provide an operational semantics for GC in terms of a label transition systems (LTS) [Plotkin 1981] describing how global descriptions evolve, and the type discipline that describes the structured sequence of message exchanges between participants from [Carbone *et al.* 2007].

6.2.1 Syntax

Let $\mathcal{C}, \mathcal{C}', \ldots$ denote *terms* of the calculus, often called *interactions* or *choreographies*; $A, B, C, \ldots$ range over *participants*; $k, k', \ldots$ are *linear channels*; $a, b, c, \ldots$ *shared channels*; $v, w, \ldots$ variables; $X, Y, \ldots$ process variables; $l, l_i, \ldots$ labels for branching; and finally $e, e', \ldots$ over unspecified arithmetic and other first-order expressions. We write $e@A$ to mean that the expression e is evaluated using the variable related to participant A in the store.

Definition 6.2.1. *The syntax of the global calculus [Carbone et al. 2006] is given by the following grammar:*

$$
\begin{array}{rll}
\mathcal{C} ::= & \mathbf{0} & \text{(inaction)} \\
\mid & A{\to}B{:}a(k).\ \mathcal{C} & \text{(init)} \\
\mid & A{\to}B : k\langle e, y\rangle.\ \mathcal{C} & \text{(com)} \\
\mid & A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I} & \text{(choice)} \\
\mid & \mathcal{C}_1 \mid \mathcal{C}_2 & \text{(par)} \\
\mid & \textbf{if } e@A \textbf{ then } \mathcal{C}_1 \textbf{ else } \mathcal{C}_2 & \text{(cond)} \\
\mid & X & \text{(recvar)} \\
\mid & \mu X.\ C & \text{(recursion)}
\end{array}
$$

Intuitively, the term (inaction) denotes a system where no interactions take place. (init) denotes a session initiation by A via B's service channel a, with a fresh session channel k and continuation $\mathcal{C}$. Note that k is bound in $\mathcal{C}$. (com) denotes an in-session communication of the evaluation (at A's) of the expression e over a session channel k. In this case, y does not bind in $\mathcal{C}$ (our semantics will treat y as a variable in the store of B). (choice) denotes a labelled choice over session channel k and set of labels I. In (par), $\mathcal{C}_1 \mid \mathcal{C}_2$ denotes the parallel product between $\mathcal{C}_1$ and $\mathcal{C}_2$. (cond) denotes the standard conditional operator where $e@A$ indicates that the expression e has to be evaluated in the store of participant A. In (recursion), $\mu X.\ \mathcal{C}$ is the minimal fix point operation for recursion, where the variable X of (recvar) is bound in $\mathcal{C}$. The

free and bound session channels and term variables are defined in the usual way. The calculus is equipped with a standard structural congruence $\equiv$, defined as the minimal congruence relation on interactions $\mathcal{C}$, such that $\equiv$ is a commutative monoid with respect to $\mid$ and $\mathbf{0}$, it is closed under alpha equivalence $\equiv_\alpha$ of terms, and it is closed under the recursion unfolding, i.e., $\mu X.\mathcal{C} \equiv \mathcal{C}[\mu X.\mathcal{C}/X]$.

Remark 6.2.2 (Differences with the approach in [Carbone *et al.* 2007]). The syntax in Definition 2.1.12 presents a simplified version of the global calculus without restriction, summation and local assignments. In its original presentation, restriction is used only during session initiation. We capture the requirement of fresh identifiers by using the operational rules in Figure 6.2. Excluding the lack of local assignment, we argue that our version of GC is, to some extent, as expressive as the one originally reported in [Carbone *et al.* 2007]. In particular, the interaction process $A{\to}B : k\langle \mathrm{op}, e, y\rangle$ as originally defined captures both selection and message passing which are instead disentangled in our case (mainly for clarity reasons). The absence of op in the interaction process $A{\to}B : k\langle e, y\rangle$ can be easily encoded with the existing operators. In fact, $A{\to}B : k\langle \mathrm{op}, e, y\rangle.\ \mathcal{C}'$ can be decomposed into $A{\to}B{:}k[op_i : \mathcal{C}'_i]_{i\in I}.\ A{\to}B : k\langle e, y\rangle.\ \mathcal{C}'$ with unary I (although we lose atomicity).

6.2.2 Semantics

We give the operational semantics in terms of configurations $(\sigma, \mathcal{C})$, where σ represents the state of the system and $\mathcal{C}$ the choreography actually being executed. The state σ contains a set of variables labelled by participants. As described in the previous subsection, a variable x located at participant A is written as $x@A$. The same variable name labelled with different participant names denotes different variables (hence $\sigma(x@A)$ and $\sigma(x@B)$ may differ). Formally, the operational semantics is defined as a labelled transition system (LTS). A transition $(\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')$ says that a choreography $\mathcal{C}$ in a state σ executes an action (or label) ℓ and evolves into $\mathcal{C}'$ with a new state σ'. Actions are defined as $\ell = \{\mathsf{init}\ A \to B\ \mathsf{on}\ a(k), \mathsf{com}\ A \to B\ \mathsf{over}\ k, \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l_i\}$, denoting initiation, in-session communication and branch selection, respectively. We write $(\sigma, \mathcal{C}) \longrightarrow (\sigma', \mathcal{C}')$ when ℓ irrelevant, and $\longrightarrow^*$ denotes the transitive closure of $\longrightarrow$. The transition relation $\longrightarrow$ is defined as the minimum relation on pairs state/interaction satisfying the rules in Figure 6.2.

Intuitively, transition (G-Init) describes the evolution of a session initiation: after A initiates a session with B on service channel a, A and B share the fresh channel h locally. (G-Com) describes the main interaction rule of the calculus: the expression e is evaluated into v in the A-portion of the state σ and then assigned to the variable x located at B resulting in the new state $\sigma[x@B \mapsto v]$. (G-Choice) chooses the evolution of a choreography resulting from a labelled choice over a session key k. (G-IfT) and (G-IfF) show the possible paths that a deterministic evolution of a choreography can produce. (G-Par) and (G-Struct) behave as the standard rules for parallel product and structural congruence, respectively.

Remark 6.2.3 (Global Parallel). Parallel composition in the global calculus differs

$$\mathsf{G}-\mathsf{Init}\quad \frac{h \text{ fresh}}{(\sigma, A{\to}B{:}a(k).\ \mathcal{C}) \xrightarrow{\mathsf{init}\ A\to B\ \mathsf{on}\ a(k)} (\sigma, \mathcal{C}[h/k])}$$

$$\mathsf{G}-\mathsf{Struct}\quad \frac{\mathcal{C} \equiv \mathcal{C}'' \quad (\sigma,\mathcal{C}) \xrightarrow{\ell} (\sigma',\mathcal{C}') \quad \mathcal{C}' \equiv \mathcal{C}'''}{(\sigma,\mathcal{C}'') \xrightarrow{\ell} (\sigma',\mathcal{C}''')}$$

$$\mathsf{G}-\mathsf{Choice}\quad \frac{}{(\sigma, A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}) \xrightarrow{\mathsf{sel}\ A\to B\ \mathsf{over}\ k:l_i} (\sigma,\mathcal{C}_i)}$$

$$\mathsf{G}-\mathsf{Par}\quad \frac{(\sigma,\mathcal{C}_1) \xrightarrow{\ell} (\sigma',\mathcal{C}_1')}{(\sigma,\mathcal{C}_1 \mid \mathcal{C}_2) \xrightarrow{\ell} (\sigma',\mathcal{C}_1' \mid \mathcal{C}_2)}$$

$$\mathsf{G}-\mathsf{IfF}\quad \frac{\sigma(e@A) \Downarrow \mathsf{ff} \quad (\sigma,\mathcal{C}_2) \xrightarrow{\ell} (\sigma',\mathcal{C}_2')}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2) \xrightarrow{\ell} (\sigma',\mathcal{C}_2')}$$

$$\mathsf{G}-\mathsf{IfT}\quad \frac{\sigma(e@A) \Downarrow \mathsf{tt} \quad (\sigma,\mathcal{C}_1) \xrightarrow{\ell} (\sigma',\mathcal{C}_1')}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2) \xrightarrow{\ell} (\sigma',\mathcal{C}_1')}$$

$$\mathsf{G}-\mathsf{Com}\quad \frac{\sigma(e@A) \Downarrow v}{(\sigma, A{\to}B : k\langle e, x\rangle.\ \mathcal{C}) \xrightarrow{\mathsf{com}\ A\to B\ \mathsf{over}\ k} (\sigma[x@B \mapsto v], \mathcal{C})}$$

Figure 6.2: Operational Semantics for the Global Calculus

from the notion of parallel found in standard concurrency models based on input/output primitives [Milner 1999]. In the latter, a term $P_1 \mid P_2$ may allow *interactions* between P_1 and P_2. However, in the global calculus, the parallel composition of two choreographies $\mathcal{C}_1 \mid \mathcal{C}_2$ concerns two parts of the described system where *interactions* may occur in $\mathcal{C}_1$ and $\mathcal{C}_2$ but never across the parallel operator $\mid$. This is because an interaction $A \to B \ldots$ abstracts from the actual end-point behaviour, i.e., how A sends and B receives. In this model, dependencies between two choreographies can be expressed by using variables in the state σ.

In its original presentation [Carbone *et al.* 2007], GC comes equipped with a reduction semantics unlike the one presented in Figure 6.2. Our LTS semantics has the advantage of allowing to observe changes on the behaviour of the system, which will prove useful when relating to the logical characterization in Section 6.3. We conjecture that our proposed LTS semantics and the reduction semantics of the global calculus originally presented in [Carbone *et al.* 2007] coincide (taking into account the considerations in Remark 6.2.2).

Lemma 6.2.4 (Reduction and LTS semantics coincide in GC). *Given $\mathcal{C}, \mathcal{C}'$ processes, $\to$ the reduction relation between global calculus processes in [Carbone et al. 2007] (Included in Appendix 6.A) and $\xrightarrow{\ell}$ the labelled transition relation in Figure 6.2. We can say that:*

Soundness *: If $(\sigma,\mathcal{C}) \to (\sigma',\mathcal{C}')$, then $\exists \ell$ s.t. $(\sigma,\mathcal{C}) \xrightarrow{\ell}{}^{*} (\sigma',\mathcal{C}'')$ and $\mathcal{C}' \equiv \mathcal{C}''$.*

Completeness *: For any ℓ, if $(\sigma,\mathcal{C}) \xrightarrow{\ell} (\sigma',\mathcal{C}')$ then $(\sigma,\mathcal{C}) \to (\sigma',\mathcal{C}')$.*

Proof. On (Soundness): The proof proceeds by induction on the length of derivation in $\rightarrow$.

On (Completeness): The proof proceeds by case analysis of the labels in ℓ over the transitions in $(\sigma, C) \xrightarrow{\ell} (\sigma', C')$. □

Example 6.2.5 (Online Booking). We consider the example presented in the introduction, i.e., a simplified version of the on-line booking scenario presented in [López *et al.* 2010]. Here, the customer (Cust) establishes a session with the airline company (AC) using service (on-line booking, shorted as ob) and creating session keys k_1, k_2. Once sessions are established, the customer will request the company about a flight offer with his booking data, along the session key k_1. The airline company will process the customer request and will send a reply back with an offer using the session key k_2. The customer will eventually accept the offer, sending back an acknowledgment to the airline company using k_1. The following specification in the global calculus represents the protocol:

$$\begin{aligned} \mathcal{C}_{\mathrm{OB}} = \mathrm{Cust}{\rightarrow}\mathrm{AC}{:}\mathrm{ob}(k_1, k_2).\ & \mathrm{Cust}{\rightarrow}\mathrm{AC} : k_1\langle \mathrm{booking}, x\rangle. \\ & \mathrm{AC}{\rightarrow}\mathrm{Cust} : k_2\langle \mathrm{offer}, y\rangle.\ \mathrm{Cust}{\rightarrow}\mathrm{AC} : k_1\langle \mathrm{accept}, z\rangle.\ \mathbf{0}\,. \end{aligned} \tag{OB}$$

6.2.3 Session Types for the Global Calculus

The Global Calculus comes accompanied wth a type discipline that ensures the proper control flow among interactions. It is built as a generalisation of session types [Honda *et al.* 1998] for global interactions, first presented in [Carbone *et al.* 2007]. Here we informally describe their use thru examples, and direct to their original presentation for a more formal view.

Session types in GC are used to structure sequence of message exchanges in a session. Their syntax is as follows:

$$\begin{aligned} \theta &= \ \mathrm{bool} \mid \mathrm{int} \mid \ \ldots \\ \alpha &= \ \uparrow(\theta).\alpha \mid \ \downarrow(\theta).\alpha \mid \&\{l_i : \alpha_i\}_{i\in I} \mid \ \oplus\{l_i : \alpha_i\}_{i\in I} \mid \alpha_1 \mid \alpha_2 \mid \mathrm{end} \mid \mu\mathbf{t}.\ \alpha \mid \mathbf{t} \end{aligned} \tag{6.12}$$

Here, θ range over standard data types bool, string, int, ... and α describe session types. We describe the forms of α.

- $\downarrow(\theta).\alpha$ and $\uparrow(\theta).\alpha$ are the input and output types and describe the reception (resp. emision) of a message with data type θ followed by a continuation α.
- Similarly, $\&\{l_i : \alpha_i\}_{i\in I}$ is the branching type while $\oplus\{l_i : \alpha_i\}_{i\in I}$ is the selection type.
- The type $\alpha_1 \mid \alpha_2$ is a parallel composition of session types α_1 and α_2.
- The type end indicates session termination and is often omitted.
- $\mu\mathbf{t}.\ \alpha$ indicates a recursive type with $\mathbf{t}$ as a type variable. $\mu\mathbf{t}.\ \alpha$ binds the free occurrences of $\mathbf{t}$ in α. We take an *equi-recursive* view on types, not distinguishing between $\mu\mathbf{t}.\ \alpha$ and its unfolding $\alpha[\mu\mathbf{t}.\ \alpha/\mathbf{t}]$.

Typing judgments in GC have the form $\Gamma \vdash \mathcal{C} \triangleright \Delta$, where Γ is a type environment describing *services*, and Δ the type environment describing *sessions*. Typically, Γ contains a set of type assignments of the form $a@A : \alpha$, which say that a service a located at participant A may be invoked and run a session according to type α. Δ contains type assignments of the form $k[A, B] : \alpha$ which say that a session channel k identifies a session between participants A and B and has session type α when seen from the viewpoint of A. There is no particular reason why one has to choose a strict direction when considering interactions, and one may as well consider $k[A, B] : \alpha$ from the viewpoint of B. We return to the specification (OB) in Example 6.2.5 to see how some of the typing rules work. One possible assignment for Δ is:

$$k_1, k_2[Cust, AC] : k_1 \downarrow \text{booking(string)}.\ k_2 \uparrow \text{offer(int)}.\ k_1 \downarrow \text{accept(string)}.\ \text{end}$$

Describing that k_1 and k_2 are names corresponding to the same session between participants $Cust$ and AC, and corresponds to the session type $\alpha = k_1 \downarrow$ booking(string). $k_2 \uparrow$ offer(int). $k_1 \downarrow$ accept(string). end when seeing it from the point of view of $Cust$.

We provide some examples on the typing rules for the GC. The full set typing rules derived from the original work are attached in Appendix 6.B. First, we comment the rule (G-TInit), which types the establishment of a new session between two participants.

$$\frac{\Gamma, a@B : (\vec{k})\alpha \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[B, A] : \alpha \quad A \neq B}{\Gamma, a@B : (\vec{k})\alpha \vdash A{\to}B{:}a(\vec{k}).\ \mathcal{C} \triangleright \Delta} \; \mathrm{G-TInit}$$

Here, the typing rule dictates some requirements on the structure of the choreography: first, the initalisation of a session between participants in $A{\to}B{:}a(\vec{k}).\ \mathcal{C}$ requires that sessions names in $\vec{k}$ correspond to a session type in the premise. Moreover, it checks that the service channel $a@B : (\vec{k})\alpha$ is declared in the service typing Γ. The rule (G-TCom) describes communication between participants:

$$\frac{\Gamma \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : \alpha \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash x@B : \theta \quad k \in \vec{k} \quad A \neq B}{\Gamma \vdash A{\to}B : k\langle e, x\rangle.\ \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : k \uparrow \theta.\ \alpha} \; \mathrm{G-TCom}$$

Here, the interaction $\mathcal{C} = A{\to}B : k\langle e, x\rangle.\ \mathcal{C}'$ will be typable with a session type $\Delta \cdot \vec{k}[A, B] : k \uparrow \theta.\ \alpha$ provided that: 1) The evaluation of the expression e at A and its recipient variable x at B correspond to the same value type, 2) the communication is performed between different participants A and B, and 3) the continuation $\mathcal{C}$ contains a session type between A and B sucht that its session names in $\vec{k}$ contain k. In the conclusion, we use an output type $k \uparrow \theta.\ \alpha$ describing the emision of value from the point of view of A. It is clear, that we could use a complementary rule to type the input of values from the point of view of B.

$$
\begin{array}{rcll}
\phi, \chi & ::= & \exists t.\ \phi & \text{(f-exists)} \\
& | & \phi \wedge \chi & \text{(f-and)} \\
& | & \neg \phi & \text{(f-neg)} \\
& | & \langle \ell \rangle \phi & \text{(f-action)} \\
& | & \mathsf{end} & \text{(f-termination)} \\
& | & e_1@A = e_2@B & \text{(f-equality)} \\
& | & \phi \mid \chi & \text{(f-parallel)} \\
& | & \Diamond \phi & \text{(f-may)} \\
\\
\ell & ::= & \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) & \text{(l-init)} \\
& | & \mathsf{com}\ A \to B\ \mathsf{over}\ k & \text{(l-com)} \\
& | & \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l & \text{(l-branch)}
\end{array}
$$

Figure 6.3: $\mathcal{GL}$: Syntax of formulae

Assumption 6.2.6 (Well-typedness). Henceforth we only consider well-typed terms for the Global calculus, unless otherwise specified.

6.3 $\mathcal{GL}$: A Logic for the Global Calculus

In this section, we introduce a logic for choreography. The logical language comprises assertions for equality and value/name passing.

6.3.1 Syntax

The grammar of assertions is given in Figure 6.3. Choreography assertions (ranged over by $\phi, \phi', \chi, \ldots$) give a logical interpretation of the global calculus introduced in the previous section. The logic includes the standard FOL operators $\wedge$, $\neg$, and $\exists$. In $\exists t.\ \phi$, the variable t is meant to range over service and session channels, participants, labels for branching and basic placeholders for expressions. Accordingly, it works as a binder in ϕ. In addition to the standard operators, the operator (f-action) represents the execution of a labelled action ℓ followed by the assertion ϕ. Those labels ℓ match the ones in the LTS of GC, i.e., they are (l-init), (l-com), and (l-branch). The formula (f-termination) represents the process termination. We also include an unspecified, but decidable, (f-equality) operator on expressions as in [Berger *et al.* 2008]. (f-may) denotes the standard eventually operators from Linear Temporal Logic (LTL) [Emerson 1991]. The spatial operator (f-parallel) denotes composition of formulae: because of the unique nature of parallel composition in choreographies, we have used the symbol | (as in separation logic [Reynolds 2002] and spatial logic [Caires & Cardelli 2001]) in order to stress the fact that there is no interference between two choreographies running in parallel.

Notation 6.3.1 (Existencial quantification over action labels). In order to simplify the readability, we introduce the concept of existential quantification over action labels as a short-cut to mean the following:

$$\begin{aligned}\exists \ell.\, \langle \ell \rangle \phi \stackrel{\text{def}}{=}\ & \exists A, B, a, k.\, \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle.\, \phi \vee \\ & \exists A, B, k.\, \langle \mathsf{com}\ A \to B\ \mathsf{over}\ k \rangle.\, \phi \vee \\ & \exists A, B, k, l.\, \langle \mathsf{sel}\ A \to B\ \mathsf{over}\ k : l \rangle.\, \phi\,.\end{aligned}$$

Remark 6.3.2 (Derived Operators). We can get the full account of the logic by deriving the standard set of strong modalities from the above presented operators. In particular, we can encode the constant true (tt) and false (ff); and the next ($\circ\phi$) and the always operators ($\Box\phi$) from LTL.

$$\mathsf{tt} \stackrel{\text{def}}{=} (0@A = 0@A) \qquad \mathsf{ff} \stackrel{\text{def}}{=} (0@A = 1@A) \qquad (e_1 \neq e_2) \stackrel{\text{def}}{=} \neg(e_1 = e_2)$$

$$\forall x.\, \phi \stackrel{\text{def}}{=} \neg\exists x.\, \neg\phi \qquad \phi \vee \chi \stackrel{\text{def}}{=} \neg(\neg\phi \wedge \neg\chi) \qquad \phi \Rightarrow \chi \stackrel{\text{def}}{=} \neg\phi \vee \chi$$

$$\Box\phi \stackrel{\text{def}}{=} \neg\Diamond\neg\phi \qquad [\ell]\phi \stackrel{\text{def}}{=} \neg\langle\ell\rangle\neg\phi \qquad \circ\phi \stackrel{\text{def}}{=} \exists\ell.\, \langle\ell\rangle\phi\,.$$

In the rest of this section, we illustrate the expressiveness of our logic through a sequence of simple, yet illuminating examples, giving an intuition of how the modalities introduced plus the existential operator $\exists$ allow to express properties of choreographies.

Example 6.3.3 (Availability, Service Usage and Coupling). The logic above allows to express that, given a service invoker (known as A in this setting) requesting the service a, there exists another participant (called B in the example) providing a with A invoking it. This can be formulated in $\mathcal{GL}$ as follows:

$$\exists B.\, \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle \mathsf{tt}\,.$$

Assume now, that we want to ensure that services available are actually used. We can use the dual property for availability, i.e., for a service provider B offering a, there exists someone invoking a:

$$\exists A.\, \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle \mathsf{tt}\,.$$

Verifying that there is a service pairing two different participants in a choreography can be done by existentially quantifying over the shared channels used in an initiation action. A formula in $\mathcal{GL}$ representing this can be the following one:

$$\exists a.\, \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k) \rangle \mathsf{tt}\,.$$

Example 6.3.4 (Causality Analysis). The modal operators of the logic can be used to perform studies of the causal properties that our specified choreography can fulfill. For instance, we can specify that given an expression e evaluated to true at participant A, there is an eventual firing of a choreography that satisfies property ϕ_1, whilst ϕ_2 will never be satisfied. Such a property can be specified as follows:

$$(e@A = \mathsf{tt}) \wedge \Diamond(\phi_1) \wedge \Box\neg\phi_2\,.$$

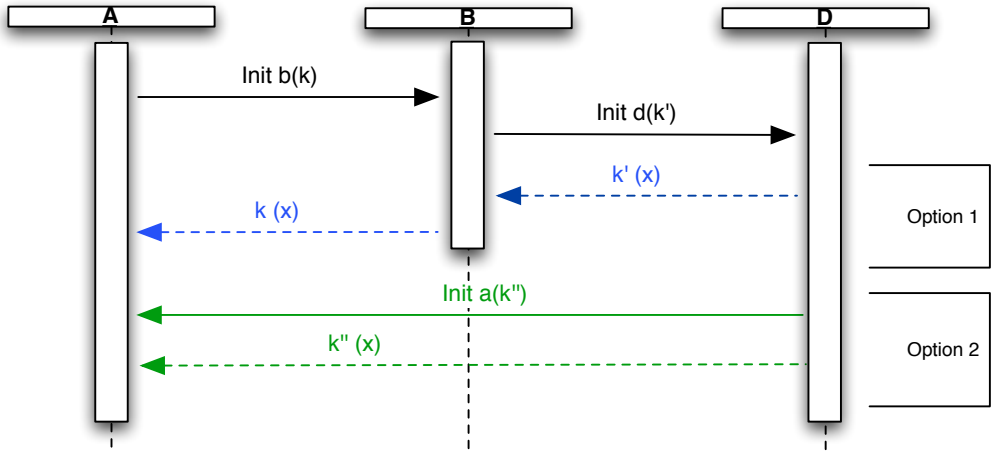


Figure 6.4: Diagram of a partial specification.

Example 6.3.5 (Response Abstraction). An interesting aspect of our logic is that it allows for the declaration of partial specification properties regarding the interaction of the participants involved in a choreography. Take for instance the interaction diagram in Figure 6.4. The participant A invokes service b at B's and then B invokes D's service d. At this point, D can send the content of variable x to A in two different ways: either by using those originally established sessions or by invoking a new service at A's. However, at the end of both computation paths, variable z (located at A's) will contain the value of x. In the global calculus, this two optional behaviour can be modelled as follows:

$$C_1 = A{\to}B{:}b(k).\ B{\to}D{:}d(k').\ D{\to}B : k'\langle x, y_B\rangle.\ B{\to}A : k\langle y_B, z\rangle.\ \mathbf{0} \qquad \text{(Option 1)}$$
$$C_2 = A{\to}B{:}b(k).\ B{\to}D{:}d(k').\ D{\to}A{:}a(k'').\ D{\to}A : k''\langle x, z\rangle\ .\ \mathbf{0}\,. \qquad \text{(Option 2)}$$

We argue that, under the point of view of A, both options are sufficiently good if, after an initial interaction with B is established, there is an eventual response that binds variable z. Such a property can be expressed by the $\mathcal{GL}$ formula:

$$\exists X, k''.\ \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k)\rangle\Diamond\Big(\langle \mathsf{com}\ X \to A\ \mathsf{over}\ k''\rangle(z@A = x@D)\Big).\ \mathsf{end}\,.$$

Notice that both the choreographies (Option 1) and (Option 2) *satisfy* the partial specification above. This will be clear in Section 6.3.2 where we introduce the semantics of logic.

Also note that a third option for the protocol at hand is to use *delegation* (the ability of communicating session keys to third participants not involved during session initiation). However, the current version of the global calculus does not feature such an operation and we leave it as future work.

$$
\begin{array}{lcl}
\mathcal{C} \models_\sigma \text{end} & \stackrel{\text{def}}{=} & \mathcal{C} \equiv \mathbf{0} \\
\mathcal{C} \models_\sigma (e_1@A = e_2@B) & \stackrel{\text{def}}{=} & \sigma(e_1@A) \Downarrow v \text{ and } \sigma(e_2@B) \Downarrow v \\
\mathcal{C} \models_\sigma \langle \ell \rangle \phi & \stackrel{\text{def}}{=} & (\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}') \text{ and } \mathcal{C}' \models_{\sigma'} \phi \\
\mathcal{C} \models_\sigma \phi \wedge \chi & \stackrel{\text{def}}{=} & \mathcal{C} \models_\sigma \phi \text{ and } \mathcal{C} \models_\sigma \chi \\
\mathcal{C} \models_\sigma \neg \phi & \stackrel{\text{def}}{=} & \mathcal{C} \not\models_\sigma \phi \\
\mathcal{C} \models_\sigma \exists t.\ \phi & \stackrel{\text{def}}{=} & \mathcal{C} \models_\sigma \phi[w/t] \text{ (for some appropriate } w) \\
\mathcal{C} \models_\sigma \Diamond \phi & \stackrel{\text{def}}{=} & (\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}') \text{ and } \mathcal{C}' \models_{\sigma'} \phi \\
\mathcal{C} \models_\sigma \phi \mid \chi & \stackrel{\text{def}}{=} & \mathcal{C} \equiv \mathcal{C}_1 \mid \mathcal{C}_2 \text{ such that } \mathcal{C}_1 \models_\sigma \phi \text{ and } \mathcal{C}_2 \models_\sigma \chi
\end{array}
$$

Figure 6.5: Assertions of the Choreography Logic

Example 6.3.6 (Connectedness). The work in [Carbone *et al.* 2007] proposes a set of criteria for guaranteeing a safe end-point projection between global and local specifications (note that the choreography in the previous example does not respect such properties). Essentially, a valid global specification have to fulfill three different criteria, namely Connectedness, Well-threadedness and Coherence. It is interesting to see that some of this criteria relate to global and local causality relations between the interactions in a choreography, and can be easily formalized as properties in the choreography logic here presented. Below, we consider the notion of connectedness and leave the other cases as future work. Connectedness dictates a global causality principle among interactions. If A initiates any action (say sending messages, assignment, etc) as a result of a previous event (e.g. message reception), then such a preceding event should have taken place at A. In the following, let $\mathsf{Interact}(A, B)\phi$ be a predicate which is true whenever $\langle \ell \rangle \phi$ holds for some ℓ with an interaction from A to B. Connectedness can then be specified as follows:

$$\forall A, B.\ \Box\Big(\mathsf{Interact}(A, B)\mathsf{tt} \Rightarrow \exists C.\ \big(\mathsf{Interact}(A, B)\mathsf{Interact}(B, C)\mathsf{tt} \vee \mathsf{Interact}(A, B)\neg\exists \ell \langle \ell \rangle \mathsf{tt}\big)\Big)\ .$$

6.3.2 Semantics

We now give a formal meaning to the assertions introduced above with respect to the semantics of the global calculus introduced in the previous section. In particular, we introduce the notion of satisfaction. We write $\mathcal{C} \models_\sigma \phi$ whenever a state σ and a choreography $\mathcal{C}$ satisfy a $\mathcal{GL}$ formula ϕ. The relation $\models_\sigma$ is defined by the rules given in Figure 6.5. In the $\exists t.\ \phi$ case, w should be an appropriate value according to the type of t, e.g., a participant if t is a participant placeholder.

Definition 6.3.7 (Satisfiability, Validity and Logical Equivalence in GL).

- *A formula ϕ is* satisfiable *if there exists some configuration under which it is true, that is, $\mathcal{C} \models_\sigma \phi$ for some $(\mathcal{C}, \sigma)$.*

- *A formula ϕ is* valid *if it is true in every configuration, that is, $\mathcal{C} \models_\sigma \phi$ for every $(\mathcal{C}, \sigma)$.*

- *A formula χ is a* logical consequence *of a formula ϕ (or ϕ* logically implies *χ), denote with an abuse of notation as $\phi \models \chi$, if every configuration $(\mathcal{C}, \sigma)$ that makes ϕ true also makes χ true.*

- *We say that a formula ϕ is* logical equivalent *to a formula χ, written $\phi \equiv_\models \chi$, if $\phi \models \chi$ iff $\chi \models \phi$.*

- *Given a set of formulae Φ and $\equiv_\models$, the equivalence class of $\phi \in \Phi$ is the subset of all elements in Φ such that are logically equivalent to ϕ:*

$$[\phi] = \{x \in \Phi | x \equiv_\models \phi\}$$

6.4 Proof System for $\mathcal{GL}$

In a previous version of this article [Carbone *et al.* 2010], we showed that $\mathcal{GL}$ is undecidable for the global calculus with recursion, and presented a model checking algorithm (in the form of a proof system) to decide when a global logic formula is satisfied by a recursion-free configuration of the global calculus. Indeed, similarly to [Charatonik & Talbot 2001], it turns out that the logic is decidable on the recursion-free choreographies[3]. We also prove the soundness and completeness of the proposed proof system w.r.t. the assertion semantics.

In order to reason about judgments $\mathcal{C} \models_\sigma \phi$, we propose a proof (or inference) system for assertions of the form $\mathcal{C} \vdash_\sigma \phi$. Intuitively, we want $\mathcal{C} \vdash_\sigma \phi$ to be as approximate as possible to $\mathcal{C} \models_\sigma \phi$ (ideally, they should be equivalent). We write $\mathcal{C} \vdash_\sigma \phi$ for the provability judgement where $(\sigma, \mathcal{C})$ is a configuration and ϕ is a formula.

Notation 6.4.1. We define the set of continuations configuration after an action ℓ and the reachable configurations, both starting from a configuration $(\sigma, \mathcal{C})$, as follows:

$$\mathsf{Next}(\sigma, \mathcal{C}, \ell) \stackrel{\text{def}}{=} \{(\sigma', \mathcal{C}') \mid (\sigma, \mathcal{C}) \xrightarrow{\ell} (\sigma', \mathcal{C}')\}$$
$$\mathsf{Reachable}(\sigma, \mathcal{C}) \stackrel{\text{def}}{=} \{(\sigma', \mathcal{C}') \mid (\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')\} .$$

We define $\mathsf{Norm}(\mathcal{C})$ to be the normalization of a recursion-free choreography $\mathcal{C}$:

[3] As described in [Carbone *et al.* 2010], removing recursion yields a decidability result in $\mathcal{GL}$

$$\frac{\mathsf{Norm}(\mathcal{C}) = [\,]}{\mathcal{C} \vdash_\sigma \mathsf{end}}\ \mathsf{P}_{\mathsf{end}} \qquad \frac{\mathcal{C} \vdash_\sigma \phi \quad \mathcal{C} \vdash_\sigma \chi}{\mathcal{C} \vdash_\sigma \phi \wedge \chi}\ \mathsf{P}_{\mathsf{and}} \qquad \frac{\mathcal{C} \nvdash_\sigma \phi}{\mathcal{C} \vdash_\sigma \neg\phi}\ \mathsf{P}_{\mathsf{neg}} \qquad \frac{\exists(\sigma', \mathcal{C}').\ \mathcal{C}' \vdash_{\sigma'} \phi}{\mathcal{C} \vdash_\sigma \Diamond\phi}\ \mathsf{P}_{\mathsf{may}}$$

$$\frac{\begin{array}{c}\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n] \\ \exists I, J.\ I \cup J = \{1, \ldots, n\} \wedge I \cap J = \emptyset \wedge \prod_{i\in I} P_i \vdash_\sigma \phi_1 \wedge \prod_{j\in J} P_j \vdash_\sigma \phi_2\end{array}}{\mathcal{C} \vdash_\sigma \phi_1 \mid \phi_2}\ \mathsf{P}_{\mathsf{par}}$$

$$\frac{\exists w \in fn(\mathcal{C}) \cup fn(\phi).\ \mathcal{C} \vdash_\sigma \phi[w/t]}{\mathcal{C} \vdash_\sigma \exists t.\ \phi}\ \mathsf{P}_{\exists} \qquad \frac{\sigma(e_1@A) \Downarrow v \quad \sigma(e_2@B) \Downarrow v}{\mathcal{C} \vdash_\sigma (e_1@A = e_2@B)}\ \mathsf{P}_{\mathsf{exp}}$$

$$\frac{\exists(\sigma', \mathcal{C}') \in \mathsf{Next}(\sigma, \mathcal{C}, \ell).\ \mathcal{C}' \vdash_{\sigma'} \phi}{\mathcal{C} \vdash_\sigma \langle\ell\rangle\phi}\ \mathsf{P}_{\mathsf{action}}$$

Table 6.1: Proof system for the Global Calculus.

$$\mathsf{Norm}(A{\to}B : k\langle e, y\rangle.\ \mathcal{C}) \stackrel{\text{def}}{=} [A{\to}B : k\langle e, y\rangle.\ \mathcal{C}]$$

$$\mathsf{Norm}(A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}) \stackrel{\text{def}}{=} [A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I}]$$

$$\mathsf{Norm}(A{\to}B{:}a(k).\ \mathcal{C}) \stackrel{\text{def}}{=} [A{\to}B{:}a(k).\ \mathcal{C}]$$

$$\mathsf{Norm}(\mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2) \stackrel{\text{def}}{=} [\mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2]$$

$$\mathsf{Norm}(\mathbf{0}) \stackrel{\text{def}}{=} [\,]$$

$$\mathsf{Norm}(\mathcal{C}_1 \mid \mathcal{C}_2) \stackrel{\text{def}}{=} [P_1, \ldots, P_n, Q_1, \ldots, Q_m] \quad \text{if} \quad \begin{array}{l}\mathsf{Norm}(\mathcal{C}_1) = [P_1, \ldots, P_n] \quad \text{and} \\ \mathsf{Norm}(\mathcal{C}_2) = [Q_1, \ldots, Q_m] \quad .\end{array}$$

Lemma 6.4.2 (Normalization preserves structural equivalence). *Let $\mathcal{C}$ be a recursion-free choreography and $\mathsf{Norm}(\mathcal{C}) = [P_1, \ldots, P_n]$, then $\mathcal{C} \equiv \prod_{i=1}^{n} P_i$.*

Proof. By induction on the structure of the choreography $\mathcal{C}$. □

Definition 6.4.3 (Entailment). *We say that a choreography $\mathcal{C}$* entails *a formula ϕ under a state σ, written $\mathcal{C} \vdash_\sigma \phi$, iff the assertion $\mathcal{C} \vdash_\sigma \phi$ has a proof in the proof system given in Table 6.1.*

Let us now describe some of the inference rules of the proof system. The rule $\mathsf{P}_{\mathsf{end}}$ relates the inaction terms with the termination formula. The rules $\mathsf{P}_{\mathsf{and}}$ and $\mathsf{P}_{\mathsf{neg}}$ denote rules for conjunction and negation in classical logic, respectively. The rule for parallel composition is represented in $\mathsf{P}_{\mathsf{par}}$; it does not indicate the behaviour of

a given choreography, but hints information about the structure of the process: $\mathsf{P}_{\mathsf{par}}$ juxtaposes the behaviour of two processes and combines their respective formulae by the use of a separation operator. The next rule, $\mathsf{P}_{\mathsf{action}}$ requires that the process P in the configuration σ can perform an action labelled ℓ, so we must search for a continuations of $(\sigma, \mathcal{C})$ after an action ℓ and find a configuration which satisfies the rest of the formula, i.e., ϕ. Analogously, $\mathsf{P}_{\mathsf{may}}$ looks for a continuation in the reachable configuration of $(\sigma, \mathcal{C})$ in oder to satisfy ϕ. The rule $\mathsf{P}_{\exists}$ says that in order to satisfy an $\exists t.\ \phi$, it is sufficient to find a value w for t in the free names used by the choreography $\mathcal{C}$ or in the free names used by the formula ϕ. Finally, the rule $\mathsf{P}_{\mathsf{exp}}$ denotes evaluation of expressions.

We now proceed to prove the soundness of the proof system with respect to the semantics of assertions presented before.

Lemma 6.4.4 (Structural congruence preserves satisfability). *If $\mathcal{C} \equiv \mathcal{C}'$ and $\mathcal{C} \models_\sigma \phi$, then $\mathcal{C}' \models_\sigma \phi$.*

Proof. (Sketch) It follows from structural induction over ϕ. □

Theorem 6.4.5 (Soundness). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, if $\mathcal{C} \vdash_\sigma \phi$ then $\mathcal{C} \models_\sigma \phi$.*

Proof. It follows by induction on the derivation of $\vdash_\sigma$. □

Lemma 6.4.6. *For every configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion free, and every formula $\exists t.\ \phi$, if $\{n_1, \ldots, n_k\} = fn(\mathcal{C}) \cup fn(\phi)$, then $\mathcal{C} \models_\sigma \exists t.\ \phi$ iff $\exists m \in \{n_1, \ldots, n_k\}$ such that $\mathcal{C} \models_\sigma \phi[m/t]$.*

Proof. (Sketch) By induction on the structure of ϕ. It is similar to the proof of [Cardelli & Gordon 2000, Lemma 5.3(3)]. □

Theorem 6.4.7 (Completeness). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, if $\mathcal{C} \models_\sigma \phi$ then $\mathcal{C} \vdash_\sigma \phi$.*

Proof. By rule induction on the derivation of $\models_\sigma$. □

Theorem 6.4.8 (Termination). *For any configuration $(\sigma, \mathcal{C})$, where $\mathcal{C}$ is recursion-free, and every formula ϕ, proof-checking algorithm terminates.*

Proof. First, notice that all the functions Norm, Next, and Reachable are total and computable. The proof is by induction over the structure of ϕ. □

6.5 End-Point Calculus

6.5.1 Syntax

The end-point calculus (EPC) [Carbone *et al.* 2007] is the π-calculus [Milner 1999] extended with sessions [Honda *et al.* 1998] as well as locations [Hennessy 2007] and

store [Carbone *et al.* 2004]. Below, $P, Q, \ldots$ denote *processes*, $M, N, \ldots$ *networks*.

$$
\begin{array}{lllll}
P ::= & ! a(\tilde{k}).\ P & \text{(initin)} & \mid\ \overline{a}\langle\tilde{k}\rangle.\ P & \text{(initout)} \\
\mid & k!\langle e\rangle.\ P & \text{(send)} & \mid\ k?(x)\,.\ P & \text{(receive)} \\
\mid & k \lhd \mathrm{l}.\ P & \text{(label selection)} & \mid\ k \rhd \{\sum_i \mathrm{l}_i.\ P_i\} & \text{(label branching)} \\
\mid & P_1 \oplus P_2 & \text{(plus)} & \mid\ P_1 \mid P_2 & \text{(par)} \\
\mid & \mu X.\ P & \text{(rec)} & \mid\ X & \text{(recvar)} \\
\mid & \textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2 & \text{(cond)} & \mid\ \mathbf{0} & \text{(inact)}
\end{array}
$$

$$
\begin{array}{lll}
N ::= & A[\,P\,]_\sigma & \text{(participant)} \\
\mid & N_1 \mid N_2 & \text{(parnet)} \\
\mid & \varepsilon & \text{(inactnet)}
\end{array}
$$

(initin) and (initout) are dual operations for describing session initiation: $! a(\tilde{k}).\ P$ denotes a process offering a replicated (available in many copies) service a with session channels $\tilde{k}$ while $\overline{a}\langle\tilde{k}\rangle.\ P$ denotes a process requesting a service a with session channels $\tilde{k}$. In both cases, P is the continuation. The next two processes denote standard in-session communications (where y_i in the first construct, the branching input, is not bound in P_i, and $\{l_i\}$ should be pairwise distinct). The term (PLUS) denotes internal choice. The rest is standard. Networks are parallel composition of participants, where a participant has the shape $A[\,P\,]_\sigma$, with A being the name of the participant, P its behaviour, and σ its local state, now interpreted as a local function from variables to values. We often omit σ when irrelevant. The free session channels, free term variables and service channels are defined as usual over processes and networks and, similarly to the global calculus, are denoted by $fsc(P/N), fv(P/N)$ and $channels(P/N)$ respectively. The syntax here presented differs from its original presentation in the absence of the local assignments and restriction of networks and processes.

6.5.2 Semantics

We give an operational semantics in terms of configurations $N \xrightarrow{\mathrm{m}} N'$, where N and N' are networks and m belongs to the sets of labels $\{\tau,\ s \uparrow k,\ s \downarrow k,\ k!\langle x\rangle,\ k?(x),\ k \rhd \mathrm{l},\ k \lhd \mathrm{l}\}$. Its labelled transition semantics follows the π-calculus and is defined by the rules given in Figure 6.7. Note that symmetric rules are omitted.

Rules in the transition semantics for EPC treat processes and networks differently. (E-S.INIT.O) and (E-S.INIT.I) describe session initiation from the point of view of the requester and provider, respectively. Here, $! a(k).P$ denotes a replicated service. Message passing communication over sessions are described by(E-M.OUT) and (E-M.IN), where $e \Downarrow v$ describes the evaluation of expression and $P[v/x]$ the substitution of variables v by x in P. Label selection/branching is given by (E-L.SEL) and (E-L.BRANCH). (E-PAR.P) and (E-SUM) are standard rules representing parallel composition and internal choice. Rules (E-PART) allows transition labels to travel out from processes to networks without modifying the store. (E-PART.IN) modifies the store of a participant after having exhibited an input behaviour. (E-COM) describes

$$\frac{}{\overline{a}\langle\tilde{k}\rangle.\ P \xrightarrow{a\uparrow\tilde{k}} P}\ \text{E}-\text{S.Init.O} \qquad \frac{}{!\,a(\tilde{k}).\ P \xrightarrow{a\downarrow\tilde{k}} P \mid !\,a(\tilde{k}).\ P}\ \text{E}-\text{S.Init.I}$$

$$\frac{e \Downarrow v}{\overline{k}!\langle e\rangle.\ P \xrightarrow{k!\langle v\rangle} P}\ \text{E}-\text{M.Out} \qquad \frac{}{k?(x).\ P \xrightarrow{k?(x)} P}\ \text{E}-\text{M.In}$$

$$\frac{}{k \lhd \mathsf{l}.\ P \xrightarrow{k\lhd\mathsf{l}} P}\ \text{E}-\text{L.Sel} \qquad \frac{1 \le j \le i}{k \rhd \{\sum_i \mathsf{l}_i.\ P_i\} \xrightarrow{k\rhd\mathsf{l}_j} P_j}\ \text{E}-\text{L.Branch}$$

$$\frac{i \in \{1,2\}}{P_1 \oplus P_2 \xrightarrow{\tau} P_i'}\ \text{E}-\text{Sum} \qquad \frac{P \xrightarrow{\mathsf{m}} P'}{P \mid Q \xrightarrow{\mathsf{m}} P' \mid Q}\ \text{E}-\text{Par.P}$$

Figure 6.6: End Point Calculus: LTS semantics for Processes

$$\frac{P \xrightarrow{\mathsf{m}} P' \quad \mathsf{m} \neq k?(x)}{A[\,P\,]_\sigma \xrightarrow{\mathsf{m}} A[\,P'\,]_\sigma}\ \text{E}-\text{Part} \qquad \frac{M \xrightarrow{\mathsf{m}} M'}{M \mid N \xrightarrow{\mathsf{m}} M' \mid N}\ \text{E}-\text{Par.N}$$

$$\frac{P \xrightarrow{k?(x)} P'}{A[\,P\,]_\sigma \xrightarrow{k?(v)} A[\,P'\,]_{\sigma[x\mapsto v]}}\ \text{E}-\text{Part.In}$$

$$\frac{N \xrightarrow{\mathsf{m}} N' \quad M \xrightarrow{\overline{\mathsf{m}}} M' \quad \mathsf{m}\ \text{not init}}{N \mid M \xrightarrow{\tau} N' \mid M'}\ \text{E}-\text{Com} \qquad \frac{N \xrightarrow{a\uparrow k} N' \quad M \xrightarrow{a\downarrow k} M' \quad k \notin fn(N') \cup fn(M')}{N \mid M \xrightarrow{\tau} (N' \mid M')[w/k]}\ \text{E}-\text{Init}$$

$$\frac{\sigma \vdash e \Downarrow \mathsf{tt} \qquad A[\,P_1\,]_\sigma \xrightarrow{\mathsf{m}} A[\,P_1'\,]_\sigma}{A[\,\textbf{if}\ e\ \textbf{then}\ P_1\ \textbf{else}\ P_2\,]_\sigma \xrightarrow{\mathsf{m}} A[\,P_1'\,]_\sigma}\ \text{E}-\text{IfT}$$

$$\frac{\sigma \vdash e \Downarrow \mathsf{ff} \qquad A[\,P_2\,]_\sigma \xrightarrow{\mathsf{m}} A[\,P_2'\,]_\sigma}{A[\,\textbf{if}\ e\ \textbf{then}\ P_1\ \textbf{else}\ P_2\,]_\sigma \xrightarrow{\mathsf{m}} A[\,P_2'\,]_\sigma}\ \text{E}-\text{IfF}$$

Figure 6.7: End Point Calculus: LTS semantics for Networks

synchronization of transition labels, used for message passing and label selection between networks. (E-Init) represents session initiation between different networks, here w acts as a "fresh" variable, used to reperesent the creation of a new session between N and M. Finally, (E-Par.N) describes the parallel composition between networks

Lemma 6.5.1 (Reduction and LTS semantics coincide in EPC). *Given N, N' networks, $\rightarrow$ the reduction relation between networks in [Carbone et al. 2007] (given for readability in Appendix 6.C) and $\xrightarrow{\mathsf{m}}$ the between processes in Figure . We can say that:*

1. *(Soundness): If $N \rightarrow N$, then $\exists \mathsf{m}$ s.t. $N \xrightarrow{\mathsf{m}} N'$.*
2. *(Completeness): For any m, if $N \xrightarrow{\mathsf{m}} N$ then $N \rightarrow N$.*

Proof. In both cases the proof proceeds by induction on the length of the derivations, being $\rightarrow$ for (1) and $\xrightarrow{\mathsf{m}}$ for (2). □

6.5.3 Session Types for the End-Point Calculus

Session types for the EPC builds from the syntax of session types in equation 6.12. Basically, the type discipline of the EPC stems from the Global Calculus, but assigns session types to every single participant instead of the whole choreography. In this way, the session typing in the EPC describes the end-point behaviour. An *end-point typing judgment* contains judgements for processes in the form $\Gamma \vdash_A P \triangleright \Delta$ (where P is typed as a behaviour for A) and judgements for networks $\Gamma \vdash N \triangleright \Delta$. In both, mappings Γ and Δ are *service* and *session typings* respectively. Here, Γ and Δ are defined as:

$$\begin{array}{rcl} \Gamma & ::= & \emptyset \mid \Gamma, a@A:(\tilde{k})\alpha \mid \Gamma, \overline{a}@A:(\tilde{k})\alpha \mid \Gamma, x@A:\theta \mid \Gamma, X:\Delta \\ \Delta & ::= & \emptyset \mid \Delta, \tilde{k}@A:\alpha \mid \Delta, \tilde{k}:\bot \end{array}$$

Above, $a@A:(\tilde{k})\alpha$ indicates the service located at A which is invoked with fresh session channels $\tilde{k}$ and offers service of the shape α, while $\overline{a}@A:(\tilde{k})\alpha$ indicates the type abstraction for the dual invocation, i.e. a client of an A's service which invokes with fresh channels $\tilde{k}$ and engages in interactions abstracted as α. Note $@A$ indicates the location of a service in both forms. As before, $\tilde{k}$ should be a vector of pairwise distinct session channels which should cover all session channels in α, and α does not contain free type variables. $(\tilde{k})$ binds occurrences of session channels in $(\tilde{k})$ in α, which induces the standard alpha-equality. A central concept in this type discipline is the notion of duality for session types, which is defined as:

$$\overline{(\tilde{k})\alpha}@A = ?(\tilde{k})\overline{\alpha}@A \qquad \overline{?(\tilde{k})\alpha@A} = (\tilde{k})\overline{\alpha}@A$$

where the notion of duality α of α remains the same.

The typing rules are almost identical as the ones from the original presentation of the EPC [Carbone *et al.* 2007], where the only difference lies on the separation between input-output types and selection-branching types as originally presented in [Honda *et al.* 1998]. Here we only comment some examples on the typing rules, and the full type system can be found in Appendix 6.D. Similarly as with the type system for the Global Calculus, we will focus the examples in session initiation and communication. The two rules (E-TInit.In),(E-TInit.Out) describe session initiation primitives:

$$\frac{\Gamma \vdash_A P \triangleright \tilde{k}@A : \alpha \quad a \notin dom(\Gamma) \quad client(\Gamma)}{\Gamma, !\, a(\tilde{k})\alpha@A \vdash_A !a(\tilde{k}).\ P \triangleright \emptyset}\ \mathrm{E-TInit.In} \qquad \frac{\Gamma, a : (\tilde{k})\alpha@B \vdash_A P \triangleright \Delta \cdot \tilde{k}@A : \alpha}{\Gamma, a : (\tilde{k})\alpha@B \vdash_A \overline{a}\langle\tilde{k}\rangle P \triangleright \Delta}\ \mathrm{E-TInit.Out}$$

In (E-TInit.In), the premise only allows for typings of session channels involved in the session initalisation of service a, that is, only the channels in $\tilde{k}$. This linearity condition blocks free session channels from ocurring during a replicated input. The condition $a \notin dom(\Gamma)$ prevents from self-calls and ensures that the type assignment

occurs at the side of the client. Requirements for the complementary typing rule (E-TInit.In) are analogous, although the linearity condition is removed. Communication rules are standard for session types, for instance, the rule (E.TOut) is used to type message outputs:

$$\frac{k \in \tilde{k} \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{k}@A : \alpha \quad \Gamma \vdash e : \theta}{\Gamma \vdash_A k!\langle e \rangle.\ P \triangleright \Delta \cdot \tilde{k}@A : k \uparrow \theta.\ \alpha} \text{E.TOut}$$

Here, process $k!\langle e \rangle.\ P$ types after evaluation that the typing of e corresponds to a correct value type and that the continuation P behaves as established by the session type in $\Delta \cdot \tilde{k}@A : \alpha$. Analogous requirements hold for typing the input process $k?(x)\,.\,P$.

6.5.4 End Point Projection

The relation between global and local views at the specification of communication protocols is given at the level of types. The central idea is that one can *project* the behaviour (type) of a global specification given in terms of choreography into a parallel composition of the behaviours of end-points. The mapping is far from trivial, and need to preserve causal relations between messages and threads, namely *connectedness*, *well-threadedness* and *coherence*. The next subsection presents a recap from the work at [Carbone *et al.* 2007]. We will use these definitions (specially Theorem 6.5.4 and Definition 6.5.5) in order to relate the work on end-point projections with their corresponding logical counterpart. In order to give the formal definition of end point projection, we first annotate global specifications with identifiers for threads.

An annotated interaction, is an annotation of a choreography with t's denoting each thread in play. Annotated interactions are written $\mathcal{A}, \mathcal{A}', \ldots$, and they are given by the following grammar:

$$\begin{array}{rlll}
\mathcal{A} ::= & A^{t_1} \to B^{t_2} : a(k).\ \mathcal{A} & \mid & \mathcal{A}_1 |^t \mathcal{A}_2 \\
\mid & A^{t_1} \to B^{t_2} : k\langle e, y \rangle.\ \mathcal{A} & \mid & \mu^t X^A.\ \mathcal{A} \\
\mid & A^{t_1} \to B^{t_2} : k[l_i : \mathcal{A}_i]_{i \in I} & \mid & X_t^A \\
\mid & \textbf{if}\ e@A^t\ \textbf{then}\ \mathcal{A}_1\ \textbf{else}\ \mathcal{A}_2 & \mid & \mathbf{0}
\end{array}$$

where each t is a natural number. We call $t, t', \cdots$ occurring in an annotated interaction, *threads*. Each $\mathcal{A}$ can be regarded as an abstract syntax built from a constructor in its root (either a prefix or a parallel product), if the tree is originated from a single thread, or a pair of threads if the intreaction involves an interaction (session initiation, message communication or selection/branching). The following is the consistent annotation of (OB).

$$\begin{array}{c} \mathrm{Cust}^1{\rightarrow}\mathrm{AC}^2{:}\mathrm{ob}(k_1, k_2).\ \mathrm{Cust}^1{\rightarrow}\mathrm{AC}^2 : k_1\langle \mathrm{booking}, x\rangle. \\ \mathrm{AC}^2{\rightarrow}\mathrm{Cust}^1 : k_2\langle \mathrm{offer}, y\rangle.\ \mathrm{Cust}^1{\rightarrow}\mathrm{AC}^2 : k_1\langle \mathrm{accept}, z\rangle.\ \mathbf{0} \end{array} \qquad (OB_{\mathcal{A}})$$

Which, although simple, could be more complicated in the case there are more than one session initiation involved in the choreography. Take for instance the case where Cust→AC:ob(k_1, k_2) is decomposed by the sequence of processes Cust→AC:ob(k_1). AC→Cust:ob(k_2) We can have different annotations for Cust and AC. The sequence: $\mathrm{Cust}^1{\rightarrow}\mathrm{AC}^2{:}\mathrm{ob}(k_1).\ \mathrm{AC}^2{\rightarrow}\mathrm{Cust}^3{:}\mathrm{ob}(k_2)$ generates a valid annotation as it places each session initation between the customer and the AC in different threads, any other annotation would be invalid.

A choreography $\mathcal{C}$ is *connected*, if the interactions within $\mathcal{C}$ describe strongly connected sequences of interactions where active/passive participants (the ones originating/receivers of an interaction). Informally, for each participant A in the set of participants of a choreography $\mathcal{C}$, a communication activity originated by A should have been immediately by a communication activity where A had acted as a receiver, or been preceeded by a self-contained action (evaluation of expressions, for instance).

Consistent annotations In order to provide meaningful projections between choreographies and its end-points, we need to define a notion of "consistent annotation", that is, an annotation $\mathcal{A}$ such that it respects causality conditions, and can be realised by a projection. Such conditions are: 1) Causal Consistency: if a participant annotated with t is passive in an interaction (a receiver), then the subsequent interaction will be marqued with t as well, or it will be a self-contained action, 2) Session Consistency: Two actions in $\mathcal{A}$ identified by the same session name are annotated with the same thread, and 3) Distinctness Condition: The input of session initiation is always given a fresh thread.

The *Well-threadedness* condition ensures global specifications are free from unrealisable dependencies among actions. We say $\mathcal{A}$ is well-threaded if it is connected and it has a consistent annotation.

Mergeability Annotations in a choreography allow for the extraction of threads directly from the global behaviour. As threads are sequences of actions to be executed at each end-point, we need to ensure that threads generated from choreographical annotations are meaningful, in the sense that they project only to the required end-points, and threads describing the behaviour of the same end point are encapsulated (*merged*) on a single service description. Mergeability, denoted by $\bowtie$, is the smallest equivalence over typed terms up to $\equiv$, closed under all typed contexts and

$$\frac{\forall i \in (I \cap H).(P_i \bowtie Q_i) \quad \forall j \in J \backslash H. \forall h \in H \backslash J. \mathrm{l}_j \neq \mathrm{l}_h}{k \rhd \{\Sigma_{j\in J} \mathrm{l}_j.\ P_j\} \bowtie k \rhd \{\Sigma_{h\in H} \mathrm{l}_h.\ P_h\}}\ \mathrm{M-Sel} \qquad \frac{fsc(P) = 0}{P \bowtie \mathbf{0}}\ \mathrm{M-Zero}$$

When $P \bowtie Q$, we say that P and Q are mergeable.

Above, a context is any end-point calculus process with some holes. (M-Sel) is for branching and says that we can allow differences in branches which do not overlap, but we do demand each pair of behaviours with the same operation to be identical.

The operation $P \sqcup Q$ allows for merging typed processes as long as they are mergeable according to the rules above. $P \sqcup Q$ is a partial commutative binary operator on typed processes which is well-defined iff $P \bowtie Q$. We see an example of the merging rules, and the full set can be consulted in Appendix 6.E. The merging of two branching processes $k \rhd \{\Sigma_{i\in I} \mathrm{l}_i.\ P_i\}$ and $k \rhd \{\Sigma_{i\in J}\}$ is given as:

$$k \rhd \{\Sigma_{i\in I} \mathrm{l}_i.\ P_i\} \sqcup k \rhd \{\Sigma_{i\in J} \mathrm{l}_i.\ Q_i\} \stackrel{\text{def}}{=} k \rhd \left\{ \begin{array}{l} \Sigma_{i\in I\cap J} \mathrm{l}_i.\ P_i \sqcup Q_i \\ +\Sigma_{i\in I\setminus J} \mathrm{l}_i.\ P_i \\ +\Sigma_{i\in J\setminus I} \mathrm{l}_i.\ Q_i \end{array} \right\}$$

That is, the resulting merge groups in a single session branching all the options coming from multiple branches that have the same session key.

Given a consistent annotation, we can project each of its threads onto an end-point process. The thread projection $\mathrm{TP}(\mathcal{A}, t)$ is a partial operation that uses the merge operator, some of the rules are given below (the full set are included in Appendix 6.F):

$$\mathrm{TP}(A^{t_1} \to B^{t_2} : b(\tilde{k}).\ \mathcal{A}, t) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \overline{b}\langle \tilde{k} \rangle.\ \mathrm{TP}(\mathcal{A}, t_1) & \text{if } t = t_1 \\ !\, b(\tilde{k}).\ \mathrm{TP}(\mathcal{A}, t_2) & \text{if } t = t_2 \\ \mathrm{TP}(\mathcal{A}, t) & \text{otherwise} \end{array} \right.$$

$$\mathrm{TP}(A^{t_1} \to B^{t_2} : k[l_i : \mathcal{A}_i]_{i\in I}, t) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} k \lhd l_i.\ \mathrm{TP}(\mathcal{A}_i, t) & \text{if } t = t_1 \\ k \rhd \{\sum_i l_i\}.\ \mathrm{TP}(\mathcal{A}_i, t) & \text{if } t = t_2 \\ \mathrm{TP}(\mathcal{A}, t) & \text{otherwise} \end{array} \right.$$

$$\mathrm{TP}(\textbf{if } e@A^{t'} \textbf{ then } \mathcal{A}_1 \textbf{ else } \mathcal{A}_2\ , t) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \textbf{if } e \textbf{ then } \mathrm{TP}(\mathcal{A}_1, t') \textbf{ else } \mathrm{TP}(\mathcal{A}_2, t') & \text{if } t = t' \\ \mathrm{TP}(\mathcal{A}_1, t) \sqcup \mathrm{TP}(\mathcal{A}_2, t) & \text{otherwise} \end{array} \right.$$

Definition 6.5.2 (Coherent Interactions). *Given a well-threaded, consistently annotated interaction $\mathcal{A}$, we say that $\mathcal{A}$ is coherent if the following two conditions hold:*

1. *For each thread t in $\mathcal{A}$, $TP(\mathcal{A}, t)$ is well-defined.*
2. *For each pair of threads t_1, t_2 in $\mathcal{A}$ with $t_1 \equiv_A t_2$, we have $TP(A, t_1) \bowtie TP(\mathcal{A}, t_2)$.*

Below, $part(\mathcal{C})$ denotes the set of participants names occurring in $\mathcal{C}$. Recall also being coherent entails being well-typed, connected and well-threaded.

Definition 6.5.3 (End-Point Projection). *Let $\mathcal{C}$ be a coherent interaction, and $\mathcal{A}$ be a consistent annotation of $\mathcal{C}$. Then the* end point projection of $\mathcal{A}$ under a state σ, *denoted* $\mathrm{EPP}(\mathcal{A}, \sigma)$, *is given as the following network.*

$$\mathrm{EPP}(\mathcal{A}, \sigma) \stackrel{def}{=} \Pi_{A\in part(\mathcal{C})}\ A[\, \Pi_{[t]} \bigsqcup_{t'\in[t]} \mathrm{TP}(\mathcal{A}, t')\,]_{\sigma@A}$$

The mapping given above is defined after choosing a specific annotation of an interaction. The following result shows the map in fact does not depend on a specific (consistent) annotation chosen, as far as a global description has no incomplete threads, i.e. it has no free session channels (which is what programmers/designers usually produce).

Theorem 6.5.4 (Soundness and Completeness of End-point Projections[Carbone *et al.* 2007]). *Assume $\mathcal{A}$ is well-typed, strongly connected, well-threaded and coherent. Assume further $\Gamma \vdash \mathcal{A} \triangleright \Delta$ and $\Gamma \vdash \sigma$. Then the following properties hold:*

- *(soundness) if* $\mathsf{EPP}(\mathcal{A}, \sigma) \longrightarrow N$ *then there exists* $\mathcal{A}'$ *such that* $(\sigma, \mathcal{A}) \longrightarrow (\sigma', \mathcal{A}')$ *such that* $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.
- *(completeness) If* $(\sigma, \mathcal{A}) \longrightarrow (\sigma', \mathcal{A}')$ *then there exist* N *such that* $\mathsf{EPP}(\mathcal{A}, \sigma) \longrightarrow N$ *and* $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N$.
- *(soundness with action labels) if* $\mathsf{EPP}(\mathcal{A}, \sigma) \xrightarrow{\mathrm{m}} N$ *then there exists* $\mathcal{A}'$ *such that* $(\sigma, \mathcal{A}) \xrightarrow{\ell} (\sigma', \mathcal{A}')$ *such that* $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.
- *(completeness with action labels) If* $(\sigma, \mathcal{A}) \xrightarrow{\ell} (\sigma', \mathcal{A}')$ *then there exist* N *such that* $\mathsf{EPP}(\mathcal{A}, \sigma) \xrightarrow{\mathrm{m}} N$ *and* $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N$.

Where $\equiv_{rec}$ denotes equality induced by the unfolding of process recursion. The assymetric relation $P \prec Q$ indicates that P is the result of cutting off "unnecessary branches" of Q, in the light of P's own typing, is formally defined as follows:

Definition 6.5.5 (Pruning). *Let $\Gamma \vdash_A P \triangleright \Delta$ for Γ and Δ minimal and $\Gamma, \Gamma' \vdash_A Q \triangleright \Delta$. If further we have $Q \equiv Q_0 \mid !R$ where $\Gamma \vdash Q_0 \triangleright \Delta$, $\Gamma' \vdash_A R$ and $P \sqcup Q_0$, then we can write: $\Gamma \vdash_A P \prec Q\,\Delta$ or $P \prec Q$ for short, and say P prunes Q under $\Gamma; \Delta$. $\prec$ is extended to networks accordingly.*

6.6 $\mathcal{LL}$: A logic for End Points

In this section, we introduce a simple logic for orchestrations. Having close resemblance with the global logic, $\mathcal{LL}$ expresses properties of end-points processes at the *local level*. This is possible by presenting a logic featuring assertions for modalities for locations [Cardelli & Gordon 2006], equality, value/name passing and spatial operators [Caires & Cardelli 2001] The grammar of assertions is given in Figure 6.8.

The logic consists of the standard FOL operators $\wedge$, $\neg$, and the existential quantifier $\exists$. In $\exists t.\ \psi$, the variable t is meant to range over service and session channels and participants. Accordingly, it works as a binder in ψ. In addition to the standard operators, we include an unspecified (decidable) equality on expressions ($e_1 = e_2$). Our operators depend on the labels of the labelled transition system of the end point calculus: $A[\mathrm{m}].\ \psi$ represents the execution of an action m located at participant A,

ψ, ω	::=	$A[\mathrm{m}].\ \psi$	(Located action)
		$\mid (e_1 = e_2)@A$	(equality)
		$\mid \psi \wedge \omega$	(conjunction)
		$\mid \neg\psi$	(neg)
		$\mid \exists t.\ \psi$	(exists)
		$\mid \Diamond\psi$	(may)
		$\mid \psi \mid \omega$	(parallel)
		$\mid \mathrm{end}$	(inaction)
m	::=	$a \downarrow k$	(Service Init Input)
		$\mid a \uparrow k$	(Service Init Output)
		$\mid k?(x)$	(Input)
		$\mid k!\langle x \rangle$	(Output)
		$\mid k \rhd \mathrm{l}$	(Branching)
		$\mid k \lhd \mathrm{l}$	(Selection)

Figure 6.8: Syntax of $\mathcal{LL}$

followed by the assertion ψ execution of a labelled action m followed by the assertion ψ; $\circ\psi$ and $\Diamond\psi$ denote the standard next and eventutally operators from Linear Temporal Logic. The spatial operator in $\psi|\chi$ denotes composition of formulae where ψ and χ do not share variables, as we can see in the definition of its semantics below.

Remark 6.6.1 (Derived Operators). We can get the full set of operators in $\mathcal{LL}$ by standard derivation from the above presented set of operations:

$$\mathrm{tt} = (0 = 0) \qquad \mathrm{ff} = (0 = 1) \qquad (e_1 \neq e_2) = \neg(e_1 = e_2)$$
$$\circ\psi = \exists \ell.\ \exists A.\ A[\mathrm{m}].\ \psi \qquad \psi \wedge \omega = \neg(\neg\psi \vee \neg\omega) \qquad \forall x.\ \psi = \neg\exists x.\ \neg\omega$$
$$\Box\psi = \neg\Diamond\neg\psi \qquad [A[\mathrm{m}]].\ \psi = \neg A[\mathrm{m}].\ \neg\psi \qquad \psi \Rightarrow \omega = \neg\psi \vee \omega$$

6.6.1 Examples of formulae in $\mathcal{LL}$

Request - Reply For a classical request-reply system in the End Point Calculus:

$$\begin{aligned} P &::= !\,p(k_1, k_2).\ k_1!\langle e_1 \rangle.\ k_2?(y)\,.\,\mathbf{0} \\ B &::= \overline{p}\langle k_1, k_2 \rangle.\ k_1?(x)\,.\,k_2!\langle e_2 \rangle.\ \mathbf{0} \\ System &::= A[P]_\sigma \mid B[Q]_\delta \end{aligned}$$

We can describe some of the formulae that $\mathcal{LL}$ can verify of regarding to this system. At the participant level, we can describe a formula enforcing an eventual response after a given message exchange:

$$
\begin{array}{lcl}
N \models A[\mathrm{m}].\ \psi & \stackrel{\text{def}}{=} & N \equiv A[\,P\,]_\sigma \mid M \wedge \exists Q, \sigma'. A[\,P\,]_\sigma \xrightarrow{\mathrm{m}} A[\,Q\,]_{\sigma'} \wedge A[\,Q\,]_{\sigma'} \mid M \models \psi \\
N \models (e_1 = e_2)@A & \stackrel{\text{def}}{=} & N \equiv A[\,P\,]_\sigma \ \wedge\ \sigma(e_1) = \sigma(e_2) \\
N \models \psi \wedge \rho & \stackrel{\text{def}}{=} & N \models \psi \text{ and } N \models \rho \\
N \models \neg\psi & \stackrel{\text{def}}{=} & N \not\models \psi \\
N \models \exists t.\ \psi & \stackrel{\text{def}}{=} & N \models \psi[w/t] \quad \text{(for some appropriate } w\text{)} \\
N \models \Diamond\psi & \stackrel{\text{def}}{=} & N \longrightarrow^* M \ \text{ and } M \models \psi \\
N \models \psi \mid \rho & \stackrel{\text{def}}{=} & N \equiv M \mid M' \text{ s.t. } M \models \psi \text{ and } M' \models \rho \\
N \models \text{end} & \stackrel{\text{def}}{=} & N \equiv \varepsilon
\end{array}
$$

Figure 6.9: Assertions of the Local logic

$$System \models A[a \uparrow (k_1, k_2)].\ A[k_1!\langle e_1\rangle].\ \Diamond A[k_2?(x)].\ \text{end}$$

An dually for B. On its composition, we can use both the parallel product or the magic wand operation to denote the fact that both participants should be present in the interaction providing complementary actions. A partial description leaving out session initiation constructions is given below:

$$System \models (A[k_1!\langle e_1\rangle].\ \Diamond A[k_2?(x)].\ \text{end}) \mid (B[k_1?(x)].\ \Diamond B[k_2!\langle e_2\rangle].\ \text{end})$$

6.6.2 Semantics of $\mathcal{LL}$

We now give a formal meaning to $\mathcal{LL}$ by providing a set of assertions describing the semantics of each of the operators of the logic with respect to the LTS semantics of the end point calculus described in the previous section. In particular we introduce the notion of satisfaction. We write $N \models \psi$ whenever a network N satisfies a formula ψ in $\mathcal{LL}$. The relation $\models$ is the maximum relation satisfying the rules given in Figure 6.9.

Definition 6.2.2 (Satisfiability, Validity and Logical Equivalence in $\mathcal{LL}$).

- *A formula ψ is* satisfiable *if there exists a network under which it is true, that is, $N \models \psi$ for some N.*
- *A formula ψ is* valid *if it is true in every network N, that is, $N \models \psi$ for every N.*
- *A formula ψ is a* logical consequence *of a formula ρ (or ψ* logically implies *ρ), denote with an abuse of notation as $\psi \models \rho$, if every network N that makes ψ true also makes ρ true up to alpha-renaming*

- *We say that a formula ψ is* logical equivalent *to a formula ρ, written $\psi \equiv_{\models} \rho$, if $\psi \models \rho$ iff $\rho \models \psi$.*
- *Given a set of formulae Ψ and $\equiv_{\models}$, the equivalence class of $\phi \in \Psi$ is the subset of all elements in Ψ such that are logically equivalent to ψ:*

$$[\psi] = \{x \in \Psi | x \equiv_{\models} \psi\}$$

We provide some auxiliary lemmas describing the relation of the logic with the behaviours evidenced in end-point processes. The main result here is the preservation of satisfiability in type-bisimilar processes. That is, if two processes are prunable (the type bisimilarity introduced in the End Point Projection), then they satisfy the same formula in $\mathcal{LL}$.

Lemma 6.6.3 (Structural congruence preserves satisfability). *If $M \equiv N$ and $M \models \psi$, then $N \models \psi$.*

Sketch. It follows from structural induction over ψ. □

Lemma 6.6.4. *if $N \models \psi$ and $\exists M.M \xrightarrow{\mathrm{m}} N$ then $M \models \rho \wedge \rho \Rightarrow \psi$*

Sketch. It follows by induction on the transitions leading to N in $M \xrightarrow{\mathrm{m}} N$ and the definition of $N \models \psi$. □

Definition 6.6.5 (Input-Output Correspondence). *We say that a network N is input-output correspondent if it contains no dangling inputs. That is for each $x \in channels(N)$, then $\bar{x} \in channels(N)$.*

Proposition 6.6.6 (Preservation of satisfiability in pruning). *Let M, N input-output correspondent networks, then $M \models \psi$ and $M \prec N$ then $N \models \psi$.*

Proof. From the definition of pruning, we have that $M \prec N$ iff $\Gamma \vdash M \triangleright \Delta$, $\Gamma, \Gamma' \vdash N \triangleright \Delta$, $N \equiv N_0 \mid !N'$ where $\Gamma \vdash N_0 \triangleright \Delta$, $\Gamma \vdash N' \triangleright$ and $M \sqcup N_0$. We need to prove that given $N \equiv N_0 \mid !N'$ then $N \models \psi$.

From Definition 6.6.5 and the definition of pruning, we know $M \sqcup N_0$ and $channels(M) = channels(N)$, so we are not filtering any formulae related to inputs that could be lost in the pruning.

From the definition of $\models$, we have that $N \models \psi \mid \rho$ iff $N \equiv N'' \mid N'''$. Substituting N_0 for N'' and $!N$ for N''', then $N \models \psi$ holds as a logical consequence of $N \models \psi \mid \rho$. □

6.6.3 Translation from $\mathcal{GL}$ to $\mathcal{LL}$

In the same way that we define that the end point projection relates operational views of choreographies and end-points, we need to have a projection between the declarative visions of choreographies and their corresponding visions in end points. We do so by providing a mapping between the Global Logic and the Local Logic, as expressed below:

Proposition 6.6.7 ($\mathcal{GL}$ Normal form). *For all $\mathcal{GL}$ formulae ϕ, there exists ϕ' such that $\phi \equiv_{\models} \exists\vec{t}.\ \exists\vec{B}.\ \phi'$, where ϕ' is $\exists$ free.*

Sketch. It follows from structural induction over ϕ □

Definition 6.6.8 (Logical Projection). *We define a translation operator $[\cdot]$ from the formulae of $\mathcal{GL}$ to the formulae of $\mathcal{LL}$. To this end we will generate a set of formulae for every process involved into the global formula. Let be ϕ a generic $\mathcal{GL}$ formula and let $\phi = \exists\vec{t}.\exists\vec{B}.\phi$ be its normal form in virtue of Proposition 6.6.7, then $[\cdot]$ is defined as follows.*

$$[\exists\vec{t}.\ \exists\vec{B}.\ \Phi] = \exists\vec{t}.\ \exists\vec{B}.\ (\prod_{A\in \mathit{parts}(\Phi)} [\Phi]_A)$$

$$\begin{aligned}
&[\mathrm{tt}]_A = \mathrm{tt}\\
&[\langle\ell\rangle.\ \Phi]_A = [l]_A.\ [\Phi]_A \quad \text{if } A \in \mathit{parts}(l)\\
&[\langle\ell\rangle.\ \Phi]_A = [\Phi]_A \qquad \text{if } A \notin \mathit{parts}(l)\\
&[\Diamond\Phi]_A = \Diamond[\Phi]_A \vee [\Phi]_A\\
&[\Phi \wedge \Psi]_A = [\Phi]_A \wedge [\Psi]_A\\
&[\Phi \mid \Psi]_A = [\Phi]_A \mid [\Psi]_A\\
&[\neg\Phi]_A = \neg[\Phi]_A\\
&[\mathrm{end}]_A = \mathrm{end}\\
&[e_1@A = e_2@B]_A = \ (e_1 = e_2)@A
\end{aligned}$$

Where $[l]_A$ is defined as:

$$\begin{aligned}
&[\mathit{init}\ A \to B\ \mathit{on}\ a(k)]_A = A[a \uparrow k]\\
&[\mathit{init}\ A \to B\ \mathit{on}\ a(k)]_B = B[a \downarrow k]\\
&[\mathit{com}\ A \to B\ \mathit{over}\ k(x)]_A = A[k!\langle x\rangle]\\
&[\mathit{com}\ A \to B\ \mathit{over}\ k(x)]_B = B[k?(x)]\\
&[\mathit{sel}\ A \to B\ \mathit{over}\ k : \mathit{op}]_A = A[k \rhd \mathrm{op}]\\
&[\mathit{sel}\ A \to B\ \mathit{over}\ k : \mathit{op}]_B = B[k \lhd \mathrm{op}]
\end{aligned}$$

The logical projection takes a formula ϕ in expressed $\mathcal{GL}$ and generates the corresponding $\mathcal{LL}$ formula for each of the participants contained in ϕ. The mapping is standard, therefore we focus our description in the case where ϕ corresponds to the action formula $\langle\ell\rangle\phi'$. The projection is defined for each participant, so there are corresponding parts for sender and receiver end-points in an action formula. For example, in the case $\phi = \langle \mathsf{init}\ A \to B\ \mathsf{on}\ a(k)\rangle.\ \phi'$, we have that ϕ projects to the parallel composition of formulae for participants A and B, which are defined as $A[a \uparrow k].\ [\phi']_A$ and $B[a \downarrow k].\ [\phi']_B$ respectively. Analogous are the cases for in-session communication and label selection.

We finish this section by providing proof of the soundness of the mapping between logics. Theorem 6.6.10 states the correspondence between end-point projections and logical projections between formulae.

Lemma 6.6.9 (End-point projections preserve $\equiv_{rec}$). *If $\mathcal{C} \equiv_{rec} \mathcal{C}'$ then* $\mathrm{EPP}(\mathcal{A}, \sigma) \equiv_{rec} \mathrm{EPP}(\mathcal{A}', \sigma)$ *where $\mathcal{A}, \mathcal{A}'$ are consistent annotated interactions of $\mathcal{C}, \mathcal{C}'$.*

Proof. By induction on the structure of $\mathcal{C}$, and Definition 6.5.3. □

Theorem 6.6.10 (Preservation of satisfaction over logical translation). *Let $\mathcal{C}$ a choreography and $\mathcal{A}$ a consistent annotation of $\mathcal{C}$. We can conclude that* $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\phi]$ *if $\mathcal{C} \models_\sigma \phi$.*

Proof. Follows by structural induction over $\mathcal{C} \models_\sigma \phi$, definitions 6.5.3 and 6.6.8.

Let $parts(\mathcal{C})$ a function returning the set of participants involved in a choreography. We have the following cases:

Case $\mathcal{C} \models_\sigma$ end: If $\mathcal{C} \models_\sigma$ end then $\mathcal{C} \equiv \mathbf{0}$, then $\mathcal{A} = \mathbf{0}$. Using Definition 6.5.3, $\mathrm{EPP}(\mathbf{0}, \sigma) = \prod_{A \in parts(\mathbf{0})} A[\, \prod_{[t]} \bigcup_{t' \in [t]} TP(\mathbf{0}, t')\,]_{\sigma@A} = \varepsilon$.

On the logical side, we have that [end] = end from definition 6.6.8. Then $\varepsilon \models$ end holds from the definition of $\models$.

Case $\mathcal{C} \models_\sigma \neg\phi$: If $\mathcal{C} \models_\sigma \phi$ then $\mathcal{C} \not\models_\sigma \phi$. We have to show $\mathrm{EPP}(\mathcal{A}, \sigma) \not\models [\sigma]$.

We proceed by contradiction: Take $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\sigma]$, then $\mathcal{C} \models_\sigma \phi$, which is a contradiction to $\mathcal{C} \models_\sigma \neg\phi$.

Case $\mathcal{C} \models_\sigma \phi \wedge \chi$: From the definition of $\models_\sigma$, $\mathcal{C} \models_\sigma \phi \wedge \chi$ iff $\mathcal{C} \models_\sigma \phi \wedge \mathcal{C} \models_\sigma \chi$.

From the induction hypothesis we have $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\phi]$ and $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\chi]$.

Then $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\phi] \wedge [\chi]$ from the definition of $\models$.

Case $\mathcal{C} \models \phi \mid \chi$: From the definition of $\models_\sigma$, $\mathcal{C} \models_\sigma \phi \mid \chi$ iff $\mathcal{C} \equiv \mathcal{C}_1 \mid \mathcal{C}_2$ such that $\mathcal{C}_1 \models_\sigma \phi$ and $\mathcal{C}_2 \models_\sigma \chi$.

From the induction hypothesis we know that, $\mathrm{EPP}(\mathcal{A}_1, \sigma) \models [\phi]$ and $\mathrm{EPP}(\mathcal{A}_2, \sigma) \models [\chi]$.

From Lemma 6.4.4 we know that $\mathcal{C}_1 \mid \mathcal{C}_2 \models_\sigma \phi$.

From the definition of the end-point projection and Lemma 6.6.9, we know that there exists N such that $N \equiv_{rec} \mathrm{EPP}(\mathcal{A}_1, \sigma) \mid \mathrm{EPP}(\mathcal{A}_2, \sigma)$.

Then $N \models [\phi] \mid [\chi]$ follows from the definition of $\models$.

Case $C \models_\sigma \langle\ell\rangle\phi$: From the definition of $\models_\sigma$, $\mathcal{C} \models_\sigma \langle\ell\rangle\phi$ iff $\langle\sigma, \mathcal{C}\rangle \xrightarrow{\ell} \langle\sigma', \mathcal{C}'\rangle \wedge \mathcal{C}' \models_{\sigma'} \phi$. We have to show that $\mathrm{EPP}(\mathcal{A}, \sigma) \models [\langle\ell\rangle\phi]$.

From induction hypothesis, we have that:

$$\mathrm{EPP}(\mathcal{A}', \sigma') \models [\phi] \wedge \mathrm{EPP}(\mathcal{A}', \sigma') \prec N \tag{6.13}$$

From (completeness with action labels) in Theorem 6.5.4 we have that, given $\langle \sigma, \mathcal{C} \rangle \xrightarrow{\ell} \langle \sigma', \mathcal{C}' \rangle$ then:

$$\mathrm{EPP}(\mathcal{A}, \sigma) \xrightarrow{\mathrm{m}} N \wedge \mathrm{EPP}(\mathcal{A}', \sigma') \prec N. \tag{6.14}$$

Now, we only have to show that $N \models [\phi]$, which holds after using Proposition 6.6.6 in $\mathrm{EPP}(\mathcal{A}', \sigma') \prec N$ from Equation 6.13.

Case $\mathcal{C} \models_\sigma \Diamond\phi$: From the definition of $\models_\sigma$, $\mathcal{C} \models_\sigma \Diamond\phi$ iff $(\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')$ and $\mathcal{C}' \models_\sigma \phi$. Assume $(\sigma, \mathcal{C}) \longrightarrow^* (\sigma', \mathcal{C}')$ as a finite sequence of transitions $(\sigma, \mathcal{C}) \longrightarrow^n (\sigma', \mathcal{C}')$. We proceed by induction on n.

Case $n = 1$: then $\mathcal{C} \models_\sigma \Diamond\phi$ iff $(\sigma, \mathcal{C}) \longrightarrow^1 (\sigma', \mathcal{C}') \wedge \mathcal{C}' \models_{\sigma'} \phi$, which is the same case as for $\mathcal{C} \models_\sigma \langle \ell \rangle \phi$ for any ℓ. As $\mathcal{C} \models_\sigma \langle \ell \rangle \phi \implies \mathrm{EPP}(\mathcal{A}, \sigma) \models [\langle \ell \rangle \phi]$ then we are done.

Case $n > 1$: then $\mathcal{C} \models_\sigma \Diamond\phi$ iff $(\sigma, \mathcal{C}) \longrightarrow^{n-1} (\sigma'', \mathcal{C}'') \longrightarrow^1 (\sigma', \mathcal{C}') \wedge \mathcal{C}' \models_{\sigma'} \phi$.

From the induction hypothesis we get $(\sigma, \mathcal{C}) \longrightarrow^{n-1} (\sigma'', \mathcal{C}'')$ and $\mathcal{C}'' \models_{\sigma''} \phi$, then $\mathcal{C} \models_\sigma \Diamond\phi$.

Moreover, $\mathrm{EPP}(\mathcal{A}'', \sigma'') = N$ and $N \models [\phi]$, with $\mathcal{A}''$ a consistent annotation of $\mathcal{C}''$, then $\mathrm{EPP}(\mathcal{A}, \sigma) \models \Diamond \circ [\phi]$.

Finally, using subsumption we know that $\Diamond[\phi] \implies \Diamond \circ [\phi]$. Then $\mathrm{EPP}(\mathcal{A}, \sigma) \models \Diamond[\phi]$, which is what we had to show.

Case $\mathcal{C} \models_\sigma e_1@A = e_2@B$: Then $\sigma(e_1@A) \Downarrow v \wedge \sigma(e_2@B) \Downarrow v$.

From $[\cdot]$, we know $[e_1@A = e_2@B] = (e_1 = e_2)@A \mid (e_1 = e_2)@B$.

We know that $\mathrm{EPP}(\mathcal{A}, \sigma) \models (e_1 = e_2)@A \mid (e_1 = e_2)@B$ iff $N \equiv A[\, P \,]_\sigma \mid B[\, Q \,]_{\sigma'} \mid M$ and $\sigma(e_1) = \sigma(e_2) \wedge \sigma'(e_1) = \sigma'(e_2)$, which follows directly from Definition 6.5.3.

□

6.6.4 $\mathcal{LL}$: Proof System

Similarly to $\mathcal{GL}$, $\mathcal{LL}$ is equipped with an inference system to deduct whether an end point N respects a given formula ψ. We write $N \vdash \psi$ for the provability judgement where N is a network and ψ is a formula in $\mathcal{LL}$.

Definition 6.6.11 (Exhibition - End Points). *We say that a network N* exhibits *a formula ψ, written $N \vdash \psi$, iff the assertion $N \vdash \psi$ has a proof in the proof system given in Figure 6.10.*

$$\frac{A[P]_\sigma \vdash \psi}{A[\overline{a}\langle k\rangle.\ P]_\sigma \vdash A[a \uparrow k].\ \psi}\ \mathsf{L.P}_{\mathsf{init-out}} \qquad \frac{A[P]_\sigma \vdash \psi}{A[!\ a(k).\ P]_\sigma \vdash A[a \downarrow k].\ \psi}\ \mathsf{L.P}_{\mathsf{init-in}} \qquad \frac{N \vdash \psi}{N \vdash \Diamond\psi}\ \mathsf{L.P}_{\mathsf{may1}} \qquad \frac{N \vdash \psi \vee \circ\Diamond\psi}{N \vdash \Diamond\psi}\ \mathsf{L.P}_{\mathsf{may2}}$$

$$\frac{\forall i \in I.\ A[P_i]_\sigma \vdash \psi_i}{A[k \rhd \{\sum_i \mathsf{l}_i.\ P_i\}] \vdash \bigwedge_{i\in I} A[k \rhd \mathsf{l}_i].\ \psi_i}\ \mathsf{L.P}_{\mathsf{bra}} \qquad \frac{A[P]_\sigma \vdash \psi}{A[k \lhd \mathsf{l}.\ P] \vdash A[k \lhd \mathsf{l}].\ \psi}\ \mathsf{L.P}_{\mathsf{sel}} \qquad \frac{}{\mathbf{0} \vdash \mathsf{end}}\ \mathsf{L.P}_{\mathsf{end}}$$

$$\frac{A[P]_\sigma \vdash \psi}{A[\overline{k}!\langle x\rangle.\ P]_\sigma \vdash A[k!\langle x\rangle].\ \psi}\ \mathsf{L.P}_{\mathsf{send}} \qquad \frac{A[P]_\sigma \vdash \psi}{A[k?(x).\ P]_\sigma \vdash A[k?(x)].\ \psi}\ \mathsf{L.P}_{\mathsf{rcv}} \qquad \frac{N \vdash \psi_1 \quad N \vdash \psi_2}{N \vdash \psi_1 \wedge \psi_2}\ \mathsf{L.P}_{\mathsf{and}}$$

$$\frac{N \equiv A[P]_\sigma \quad \sigma(e_1) \Downarrow v \quad \sigma(e_2) \Downarrow v}{N \vdash (e_1 = e_2)@A}\ \mathsf{L.P}_{\mathsf{exp}} \qquad \frac{N \vdash \psi \quad N \equiv N'}{N' \vdash \psi}\ \mathsf{L.P}_{\mathsf{struct}} \qquad \frac{N \not\vdash \psi}{N \vdash \neg\psi}\ \mathsf{L.P}_{\mathsf{neg}}$$

$$\frac{\sigma(e) \Downarrow \mathsf{tt} \quad A[P_1]_\sigma \vdash \psi}{A[\textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2\,]_\sigma \vdash \psi}\ \mathsf{L.P}_{\mathsf{ifT}} \qquad \frac{\sigma(e) \Downarrow \mathsf{ff} \quad A[P_2]_\sigma \vdash \psi}{A[\textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2\,]_\sigma \vdash \psi}\ \mathsf{L.P}_{\mathsf{ifF}} \qquad \frac{N_1 \vdash \psi_1 \quad N_2 \vdash \psi_2}{N_1 \mid N_2 \vdash \psi_1 \mid \psi_2}\ \mathsf{L.P}_{\mathsf{par}}$$

$$\frac{\exists w \in fn(N) \cup \{k\}.\ N \vdash \psi[w/t] \quad k \text{ fresh}}{N \vdash \exists t.\ \psi}\ \mathsf{L.P}_{\exists} \qquad \frac{A[P_1]_\sigma \vdash \psi}{A[P_1 \oplus P_2]_\sigma \vdash \psi}\ \mathsf{L.P}_{\mathsf{oplus1}} \qquad \frac{A[P_2]_\sigma \vdash \psi}{A[P_1 \oplus P_2]_\sigma \vdash \psi}\ \mathsf{L.P}_{\mathsf{oplus2}}$$

Figure 6.10: Proof system for the End Point Calculus.

Let us now describe some of the inference rules of the proof system. The rule $\mathsf{L.P}_{\mathsf{end}}$ relates the inaction terms with the termination formula. The rules $\mathsf{L.P}_{\mathsf{and}}$ and $\mathsf{L.P}_{\mathsf{neg}}$ denote rules for conjunction and negation in classical logic, respectively. The rule for parallel composition is represented in $\mathsf{L.P}_{\mathsf{par}}$; it does not indicate the behaviour of a given end-point, but relates interactiing end-points with their correspondend formulae: $\mathsf{L.P}_{\mathsf{par}}$ juxtaposes the behaviour of two processes and combines their respective formulae by the use of a separation operator. The correspondence between a network and a may formula is given by a formula realting each of the possible labels it can contain.: $\mathsf{L.P}_{\mathsf{bra}}$ can be explained as follows: suppose we are given a process $N = A[\,s \rhd \{\Sigma_i l_i\}\,.\ P_i\,]_\sigma$, a set of branch labels $\{l_i \mid i \in I\}$ (determined by typing) and we are given a proof that each $A[\,\sigma\,]_{P_i}$ satisfies ψ_i, then we certainly have a proof saying that every derivation of N should satisfy a guard l_i followed by a formula ψ_i. $\mathsf{L.P}_{\mathsf{init-out}}$ and $\mathsf{L.P}_{\mathsf{init-in}}$ describe the session initiation formulae, $\mathsf{L.P}_{\mathsf{bra}}$ and $\mathsf{L.P}_{\mathsf{sel}}$ describe label branching and selection, and $\mathsf{L.P}_{\mathsf{rcv}}$ and $\mathsf{L.P}_{\mathsf{send}}$ data communication. Analogously, The rule $\mathsf{P}_{\exists}$ says that in order to satisfy an $\exists t.\ \psi$, it is sufficient to find a value w for t in the free names used by the network N or in the free names used by the formula ψ. Rule $\mathsf{P}_{\mathsf{exp}}$ denotes evaluation of local expressions. As can be noted by $\mathsf{L.P}_{\mathsf{may1}}$ and $\mathsf{L.P}_{\mathsf{may2}}$, the proof system encodes the

eventual operator as the unfolding recursion of $\Diamond\psi$. $\mathrm{P}_{\mathrm{struct}}$ proves that processes that are structurally congruent bear correspondence with the same logical formula. Conditional and internal choice rules $\mathrm{L.P}_{\mathrm{ifT}}$, $\mathrm{L.P}_{\mathrm{ifF}}$, $\mathrm{L.P}_{\mathrm{oplus1}}$ and $\mathrm{L.P}_{\mathrm{oplus2}}$ are also standard.

Theorem 6.6.12 (Soundness). *For any recursion-free network N, and every formula ψ, if $N \vdash \psi$ then $N \models \psi$.*

Proof. It follows by induction on the derivation of $\vdash$.

Case $\mathrm{P}_{\mathrm{end}}$: trivial.

Case $\mathrm{L.P}_{\mathrm{init-out}}$: We have that $N = A[\,\overline{a}\langle k\rangle.\ P\,]_\sigma$ and $N \vdash A[a \uparrow k].\ \psi$. By $\mathrm{L.P}_{\mathrm{init-out}}$, we have that $A[\,P\,]_\sigma \vdash \psi$. From induction hypothesis we get that $A[\,P\,]_\sigma \models \psi$. We have to show that $A[\,\overline{a}\langle k\rangle.\ P\,]_\sigma \models A[a \uparrow k].\ \psi$. From the assertion semantics, we have that $N' \models A[a \uparrow k].\ \psi$ iff $N' \equiv A[\,Q\,]_\sigma \mid M \wedge \exists R, \sigma'. A[\,Q\,]_\sigma \xrightarrow{a\uparrow k} A[\,R\,]_{\sigma'} \wedge A[\,R\,]_{\sigma'} \mid M \models \psi$, which holds immediately from the selection of $Q = \overline{a}\langle k\rangle.\ P$ and the induction hypothesis.

Cases $\mathrm{L.P}_{\mathrm{init-in}}, \mathrm{L.P}_{\mathrm{bra}}, \mathrm{L.P}_{\mathrm{sel}}, \mathrm{L.P}_{\mathrm{send}}, \mathrm{L.P}_{\mathrm{rcv}}$: Analogous to case $\mathrm{L.P}_{\mathrm{init-out}}$.

Case $\mathrm{L.P}_{\mathrm{par}}$: We have that $N = N_1 \mid N_2$ and $N \vdash \psi_1 \mid \psi_2$ and by $\mathrm{L.P}_{\mathrm{par}}$ we know that $N_1 \vdash \psi_1$ and $N_2 \vdash \psi_2$. From the induction hypothesis we know that $N_1 \models \psi_1$ and $N_2 \models \psi_2$. We have to show $N \models \psi_1 \mid \psi_2$, which follows directly from the assertion semantics for $\psi_1 \mid \psi_2$ and the induction hypothesis.

Cases $\mathrm{L.P}_{\mathrm{and}}$, $\mathrm{L.P}_{\mathrm{exp}}$: Analogous to $\mathrm{P}_{\mathrm{par}}$.

Case $\mathrm{L.P}_{\mathrm{neg}}$: We have that $N \vdash \neg\psi$, so by $\mathrm{L.P}_{\mathrm{neg}}$ we get $N \nvdash \psi$. By induction hypothesis we have that $N \not\models \psi$, which is necessary condition to deduce $N \models \neg\psi$.

Case $\mathrm{L.P}_{\exists}$: We have that $N \vdash \exists t.\psi$ and by $\mathrm{L.P}_{\exists}$ we have that $\exists w \in fn(N) \cup \{k\}$ and $N \vdash \psi[w/t]$. By induction hypothesis we know that $N \models \psi[w/t]$. Take $w \in fn(N)$ or a fresh name, then we know that $N \models \psi[w/t]$ with appropriate w, and then $N \models \exists t.\psi$ follows from the definition of the assertion semantics.

Case $\mathrm{L.P}_{\mathrm{may1}}$: We have that $N \vdash \Diamond\psi$ and by $\mathrm{L.P}_{\mathrm{may1}}$ then $N \vdash \psi$. By induction hypothesis we have $N \models \psi$. We have to show that $N \models \Diamond\psi$, which follows immediately from $\psi \Rightarrow \Diamond\psi$, so $N \models \Diamond\psi$.

Case $\mathrm{L.P}_{\mathrm{may2}}$: We have that $N \vdash \Diamond\psi$ and by $\mathrm{L.P}_{\mathrm{may2}}$ then $N \vdash \psi \vee \circ\Diamond\psi$. By induction hypothesis we have that $N \models \psi \vee \circ\Diamond\psi$. We have to prove that $N \models \Diamond\psi$. By definition of the assertion semantics, $N \models \Diamond\psi$ iff $N \longrightarrow^* M$. $M \models \psi$. We can express $N \longrightarrow^* M$ as a finite sequence of transitions $N \longrightarrow^n M$. We have to show that there exists $0 \leq i \leq n$ such that $N \longrightarrow^i N' \longrightarrow^j M$ and $N' \models \Diamond\psi$. We proceed by second induction on $j - i$:

1. **($j - i = j$):** Then $N \longrightarrow^{0} N' \longrightarrow^{n} M$ and $M \models \psi$, which is the same case than the one for L.P$_{\text{may1}}$, hence $N \models \Diamond\psi$.
2. **($j - i = 1$):** Then $N \longrightarrow^{1} N' \longrightarrow^{n-1} M$ and $M \models \psi$, so $N \models \circ\Diamond\psi$, then $N \models \Diamond\psi$ holds true by direct application of the induction hypothesis.
3. **($j - i = k \wedge 1 < k \leq n$):** Then we have that $N \longrightarrow^{i} N' \longrightarrow^{j} M$ and $M \models \psi$. By second inductive hypothesis we have that $N' \models \Diamond\psi$. We can decompose $N \longrightarrow^{i} N'$ as:

$$N \longrightarrow^{1} N'' \longrightarrow^{i-1} N' \tag{6.15}$$

The combination of the second inductive hypothesis and the assertion semantics for equation 6.15 leads to $N \models \circ\Diamond\Diamond\psi$, which reduces to $N \models \circ\Diamond\psi$ using standard formula equivalences from LTL [Emerson 1991].

Case L.P$_{\text{struct}}$**:** It follows by direct application of lemma 6.4.4.

Case L.P$_{\text{ifT}}$, L.P$_{\text{ifF}}$**:** We take only the proof for L.P$_{\text{ifT}}$, the other works similarly.

We have that $N = A[\,\textbf{if}\ e\ \textbf{then}\ P_1\ \textbf{else}\ P_2\,]_\sigma$, by L.P$_{\text{ifT}}$ we have that $\sigma(e) \Downarrow \text{tt}$ and $A[\,P_1\,]_\sigma \vdash \psi$, and by induction hypothesis we have that $A[\,P_1\,]_\sigma \models \psi$. We have to show that $N \models \psi$. Assume a σ s.t. $\sigma(e) \Downarrow \text{tt}$ (The other case is symmetric), from the assertion semantics, we get that $N' \models (e_1 = e_2)@A$ iff $N' \equiv A[\,P\,]_\sigma$ and $\sigma(e_1) = \sigma(e_2)$, which holds true from Lemma 6.6.9 and the induction hypothesis, therefore $N \models \psi$.

Case L.P$_{\text{oplus1}}$**,** L.P$_{\text{oplus2}}$**:** It follows by direct application of lemma 6.6.4.

□

6.7 Conclusion and Related Work

This ongoing work aims at establishing the relations between imperative and declarative views of structured communications, and it constitutes just the first step towards a verification framework for communication-centred programs. Sumarising, this work argues that one can have more flexible specifications in a declarative (logical) of communication-centred programs than in an imperative one, and it presents ways of verifying the correspondence of imperative views with respect to their declarative ones, in terms of proof systems for each of the levels of abstraction here considered (choreographies and end-points). Similarly, we establish a connection between the methodology used for describing communication-centred programs imperatively (the end-point projection) and a logical projection between logics, and prove that the end-points generated from a global specification comply to the projections of global formulae in the local logic. Some further development of the ideas here exposed involve the proof about the completeness of $\mathcal{LL}$ in the same lines as the one in $\mathcal{GL}$, and exploring the termination of the proof checking algorithm. These results paves

the way towards the goal of veryfying structured communications, and one foresee further implementation of model checking techniques where the connections between declarative and imperative specifications can be exploited.

The development of a logical vision for structured communications have placed us with questions about the correct set of operators that we want to have in the logic. In this document we explored derivations of Hennessy-Milner Logics, where the main properties of interest involved action and may formulae both at the level of choreographies ond end-points. The may operator tell us important information about the existance of an evolution where a property is fulfilled, but someimes it can fail short by allowing other evolutions of the system that does not comply to the property. In [Carbone *et al.* 2011] we started studies on stronger versions of the may modality, where one is allowed to express that a property is fulfilled in all possible executions in an eventual state, and their implementation as part of the operators in $\mathcal{GL}$ is foresee. Other improvements to the logics proposed include the use of fixed points, essential for describing state-changing loops, and auxiliary axioms describing structural properties of a choreography.

Related Work The connections between logics and session types have been explored in different works. Here we comment on some of the most representative exponents, namely [Coppo & Dezani-Ciancaglini 2009, Caires & Pfenning 2010, Bocchi *et al.* 2010, Gordon & Fournet 2009, Berger *et al.* 2008]. In [Coppo & Dezani-Ciancaglini 2009], a calculus combining notions of concurrent constraint programming and name passing is proposed. The resulting calculus treats sessions as constraint formulae representing the requirements to be satisfied in a client-server communication, in similar approach as the CC-Pi calculus explained above. As communications are represented as constraints, the type discipline takes account on how processes and constraints are related, guaranteeing that communications follow an structured communications as in [Honda *et al.* 1998].

The relationship between session types and linear logics has been explored in [Caires & Pfenning 2010], where the authors establish a bidirectional correspondence between the session types and (dual) intuitionistic linear logic formulae. The correspondence is tight, and relates the existence of a simulation between reductions in session types and proof reductions in dual intuitionistic linear logic, and vice versa. In [Pérez *et al.* 2012], the authors make use of the linear logic interpretation of session types to describe a theory of logical relations for session types, allowing one to study properties like termination of well-typed interactions, and behavioural characterisations of session-typed isomorphisms as linear logic equivalences.

Type and effect systems have been used to study structured communications. In [Gordon & Jeffrey 2003], the π- calculus is extended with labelled assertions describing progress in their communication steps. Assertions have complementary operations, and one can ensure that the communication is safe if all specified assertions have their correspondent begin-end operations present in the run of a protocol. In [Bonelli *et al.* 2005], the theory of session types with corresponding assertions is

studied, providing stronger guarantees for session types, in the sense that correspondence assertions allows one to keep track of the changes on the data transmitted over sessions and the way data is propagated across multiple parties.

Relations between types and logics can also give more information about the nature of structured communications. In [Bocchi *et al.* 2010], authors proposed the integration of typed-based signatures with logical predicates as a method to guarantee finer grained properties about the information in transit in structured interactions. The proposed a methodology (*Design by contract*), constitutes an extension of multiparty session types [Honda *et al.* 2008] with global assertions, describing global constraints on processes' interactions in terms of predicate logic formulae. In this way, types not only describe causal relations between the inter-process communications, but they also fulfil constraints regarding the values in transit.

In [Berger *et al.* 2008], a proof systems characterising May/Must testing pre-orders and bisimilarities over typed π-calculus processes is presented. The connection between types and logics in such system comes in handy to restrict the shape of the processes one might be interested, allowing us to consider such work as a suitable proof system for calculi describing the communication of end points.

In the context of security, the work on F7 [Gordon & Fournet 2009] has explored the integration of dependent and refinement types in a suite of functional programming languages, with the aim of statically checking assertions about data and state, in order to enforce security policies.

Appendix 6.A Global Calculus: Reduction Semantics

The reduction semantics of the Global Calculus is defined by the rules in Figure 6.11

$$\frac{h \text{ is fresh}}{(\sigma, A{\to}B{:}a(k).\ \mathcal{C}) \longrightarrow (\sigma, \mathcal{C}[h/k])}\ \mathrm{G-RInit} \qquad \frac{\mathcal{C} \equiv \mathcal{C}'' \quad (\sigma,\mathcal{C}) \longrightarrow (\sigma',\mathcal{C}') \quad \mathcal{C}' \equiv \mathcal{C}'''}{(\sigma,\mathcal{C}'') \longrightarrow (\sigma',\mathcal{C}''')}\ \mathrm{G-RStruct}$$

$$\frac{(\sigma, \mathcal{C}[\mu X.\mathcal{C}/\mathcal{C}]) \longrightarrow (\sigma',\mathcal{C}')}{(\sigma,\mu X.\mathcal{C}) \longrightarrow (\sigma',\mathcal{C}')}\ \mathrm{G-RRec} \qquad \frac{(\sigma,\mathcal{C}_1) \longrightarrow (\sigma',\mathcal{C}_1')}{(\sigma,\mathcal{C}_1 \mid \mathcal{C}_2) \longrightarrow (\sigma',\mathcal{C}_1' \mid \mathcal{C}_2)}\ \mathrm{G-RPar}$$

$$\frac{\sigma(e@A) \Downarrow \mathrm{tt}}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2\ ) \longrightarrow (\sigma,\mathcal{C}_1)}\ \mathrm{G-RIfT} \qquad \frac{\sigma(e@A) \Downarrow \mathrm{ff}}{(\sigma, \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2\ ) \longrightarrow (\sigma,\mathcal{C}_2)}\ \mathrm{G-RIfF}$$

$$\frac{\sigma(e@A) \Downarrow v}{(\sigma, A{\to}B : k\langle \mathrm{op}, e, x\rangle.\ \mathcal{C}) \longrightarrow (\sigma[x@B \mapsto v], \mathcal{C})}\ \mathrm{G-RCom}$$

Figure 6.11: Reduction Semantics for the Global Calculus

Appendix 6.B Global Calculus: Typing Rules

The global typing judgments are triples $\Gamma \vdash \mathcal{C} : \Delta$ inductively defined by the typing rules in Figure 6.12.

$$\frac{\Gamma, a@B : (\vec{k})\alpha \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[B, A] : \alpha \quad A \neq B}{\Gamma, a@B : (\vec{k})\alpha \vdash A{\to}B{:}a(\vec{k}).\ \mathcal{C} \triangleright \Delta} \; \mathsf{G-TInit}$$

$$\frac{\Gamma \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : \alpha \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash x@B : \theta \quad k \in \vec{k} \quad A \neq B}{\Gamma \vdash A{\to}B : k\langle e, x\rangle.\ \mathcal{C} \triangleright \Delta \cdot \vec{k}[A, B] : k \uparrow \theta.\ \alpha} \; \mathsf{G-TCom}$$

$$\frac{\Gamma \vdash \mathcal{C} \triangleright \Delta \cdot \vec{k}[B, A] : \alpha \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash x@B : \theta \quad k \in \vec{k} \quad A \neq B}{\Gamma \vdash A{\to}B : k\langle e, x\rangle.\ \mathcal{C} \triangleright \Delta \cdot \vec{k}[B, A] : k \downarrow \theta.\ \alpha} \; \mathsf{G-TComInv}$$

$$\frac{\Gamma \vdash \mathcal{C}_j \triangleright \Delta \cdot \vec{k}[A, B] : \alpha_j \quad k \in \vec{k} \quad A \neq B \quad j \in I}{\Gamma \vdash A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I} \triangleright \Delta \cdot \vec{k}[A, B] : \&\{l_i : \alpha_i\}_{i\in I}} \; \mathsf{G-TChoice}$$

$$\frac{\Gamma \vdash \mathcal{C}_j \triangleright \Delta \cdot \vec{k}[B, A] : \alpha_j \quad k \in \vec{k} \quad A \neq B \quad j \in I}{\Gamma \vdash A{\to}B{:}k[l_i : \mathcal{C}_i]_{i\in I} \triangleright \Delta \cdot \vec{k}[B, A] : \oplus\{l_i : \alpha_i\}_{i\in I}} \; \mathsf{G-TChoiceInv}$$

$$\frac{\Gamma \vdash \mathcal{C}_1 \triangleright \Delta_1 \quad \Gamma \vdash \mathcal{C}_2 \triangleright \Delta_2}{\Gamma \vdash \mathcal{C}_1 \mid \mathcal{C}_2 \triangleright \Delta_1 \bullet \Delta_2} \; \mathsf{G-Tpar} \qquad \frac{\Gamma \vdash e@A : \mathsf{bool} \quad \Gamma \vdash \mathcal{C}_1 \triangleright \Delta \quad \Gamma \vdash \mathcal{C}_2 \triangleright \Delta}{\Gamma \vdash \mathbf{if}\ e@A\ \mathbf{then}\ \mathcal{C}_1\ \mathbf{else}\ \mathcal{C}_2 \triangleright \Delta} \; \mathsf{G-TIf}$$

$$\frac{\Gamma \cdot X : \Delta \vdash \mathcal{C} \triangleright \Delta}{\Gamma \vdash \mu X.\ \mathcal{C} \triangleright \Delta} \; \mathsf{G-TRec} \qquad \frac{\Gamma, X : \Delta \text{ well formed}}{\Gamma, X : \Delta \vdash X \triangleright \Delta} \; \mathsf{G-TVar} \qquad \frac{\Gamma \text{ well formed} \quad \forall i \neq j, \{k_i\} \cap \{k_j\} = \emptyset}{\Gamma \vdash \mathbf{0} \triangleright \bigcup_i \vec{k_i}[A_i, B_i]\mathsf{end}} \; \mathsf{G-TZero}$$

Figure 6.12: Global Calculus: Typing Rules

Appendix 6.C End-Point Calculus: Reduction Semantics

The reduction semantics for the end-point calculus follows the π-calculus and is defined by the rules in Figure 6.13.

Appendix 6.D End-Point Calculus: Typing rules

The typing rules for the End Point Calculus are given in Figure 6.D, were the compatibility operators $\asymp$ and $\odot$ are defined accordingly as:

1. Two service typings Γ_1 and Γ_2 are compatible (written $\Gamma_1 \asymp \Gamma_2$) if they satisfy the following conditions:

   (a) if $a@A \in \mathbf{dom}(\Gamma_i)$ then $a@B \notin \mathbf{dom}(\Gamma_j)$ for every B and for $i \neq j$;

$$\text{E}-\text{RInit}\quad \frac{k_i \notin \mathrm{fsc}(P') \cup \mathrm{fsc}(Q') \quad \tilde{h} \text{ is fresh}}{A[\,!\,a(\tilde{k}).\ P \mid P'\,]_\sigma \mid B[\,\overline{a}\langle\tilde{k}\rangle.\ Q \mid Q'\,]_{\sigma'} \ \rightarrow\ (A[\,!\,a(\tilde{k}).\ P \mid P \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'})[\tilde{h}/\tilde{k}]}$$

$$\text{E}-\text{RCom}\quad \frac{\sigma \vdash e \Downarrow v}{A[\,k?(x)\,.\ P \mid P'\,]_\sigma \mid B[\,k!\langle x\rangle.\ Q \mid Q'\,]_{\sigma'} \ \rightarrow\ A[\,P \mid P'\,]_{\sigma[x \mapsto v]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

$$\text{E}-\text{RSel}\quad \frac{j \in I}{A[\,k \rhd \{\textstyle\sum_i l_i.\ P_i\} \mid P'\,]_\sigma \mid B[\,k \lhd l_j.\,.\ Q \mid Q'\,]_{\sigma'} \ \rightarrow\ A[\,P_j \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

$$\text{E}-\text{RIfT}\quad \frac{\sigma \vdash e \Downarrow \mathrm{tt}}{A[\,\textbf{if}\ e\ \textbf{then}\ P_1\ \textbf{else}\ P_2 \mid P'\,]_\sigma \ \rightarrow\ A[\,P_1 \mid P'\,]_\sigma} \qquad \text{E}-\text{RPar}\quad \frac{M \ \rightarrow\ M'}{M|N \ \rightarrow\ M'|N}$$

$$\text{E}-\text{RIfF}\quad \frac{\sigma \vdash e \Downarrow \mathrm{ff}}{A[\,\textbf{if}\ e\ \textbf{then}\ P_1\ \textbf{else}\ P_2 \mid P'\,]_\sigma \ \rightarrow\ A[\,P_2 \mid P'\,]_\sigma} \qquad \text{E}-\text{RSum}\quad \frac{i \in \{1,2\}}{A[P_1 \oplus P_2|R]_\sigma \ \rightarrow\ A[P_i|R]_\sigma}$$

$$\text{E}-\text{RRec}\quad \frac{A[\,P[\mu X.P/X] \mid Q\,]_\sigma \mid N \ \rightarrow\ N'}{A[\,\mu X.P \mid Q\,]_\sigma \mid N \ \rightarrow\ N'} \qquad \text{E}-\text{RStruct}\quad \frac{M \equiv M' \quad M' \rightarrow N' \quad N' \equiv N}{M \rightarrow N}$$

Figure 6.13: Reduction Relation for the End-Point Calculus

(b) if $\overline{a}@A \in \mathbf{dom}(\Gamma_i)$ and $\overline{a}@B \in \mathbf{dom}(\Gamma_j)$ then $A = B$ and $\Gamma_i(\overline{a}@A) = \Gamma_j(\overline{a}@B)$ for $i \neq j$ (up to α-renaming of bound names);

(c) if $a@A \in \mathbf{dom}(\Gamma_i)$ and $\overline{a}@B \in \mathbf{dom}(\Gamma_j)$ then $A = B$ and $\Gamma_i(a@A) = \overline{\Gamma_j(\overline{a}@B)}$ for $i \neq j$ (up to α-renaming of bound names);

(d) $\Gamma_1(x) = \Gamma_2(x)$ for each x in $\Gamma_{1,2}$;

(e) $\Gamma_1(X) = \Gamma_2(X)$ for each X in $\Gamma_{1,2}$.

2. Two session typings Δ_1 and Δ_2 are compatible (written $\Delta_1 \asymp \Delta_2$) if they satisfy the following conditions:

(a) if $\tilde{k} \in \mathrm{dom}(\Delta_1)$, $\tilde{t} \in \mathrm{dom}(\Delta_2)$ and $\tilde{k} \cap \tilde{t} \neq \emptyset$ then $\tilde{k} = \tilde{t}$;

(b) if $\tilde{k} : \bot \in \Delta_i$ then $\tilde{k} \notin \mathrm{dom}(\Delta_j)$ for $i \neq j$;

(c) if $\tilde{k}@A : \alpha_1$ in Δ_1 and $\tilde{k}@A : \alpha_2$ in Δ_2 then $\mathrm{fsc}(\alpha_1) \cap \mathrm{fsc}(\alpha_2) = \emptyset$;

(d) if $\tilde{k}@A : \alpha_1$ in Δ_1 and $\tilde{k}@B : \alpha_2$ in Δ_2 then $\alpha_1 = \overline{\alpha_2}$ (for $A \neq B$).

$$\frac{\Gamma \vdash_A P \triangleright \vec{k}@A : \alpha \quad a \notin dom(\Gamma) \quad client(\Gamma)}{\Gamma, !\, a(\vec{k})\alpha@A \vdash_A !a(\vec{k}).\ P \triangleright \emptyset}\ \text{E}-\text{TInit.In} \qquad \frac{\Gamma, a : (\vec{k})\alpha@B \vdash_A P \triangleright \Delta, \tilde{k}@A : \alpha}{\Gamma, a : (\vec{k})\alpha@B \vdash_A \overline{a}\langle\tilde{k}\rangle P \triangleright \Delta}\ \text{E}-\text{TInit.Out}$$

$$\frac{j \in J \quad J \subseteq I \quad k \in \tilde{k} \quad \Gamma \vdash_A P_j \triangleright \Delta \cdot \tilde{k}@A : \alpha_j}{\Gamma \vdash_A k \rhd \{\sum_{i\in I} l_i.\ P_i\} \triangleright \Delta \cdot \tilde{k}@A : k \rhd \&\{l_i : \alpha_i\}}\ \text{E.TBranch} \qquad \frac{}{\Gamma, X : \Delta \vdash_A X \triangleright \Delta}\ \text{E.TVar}$$

$$\frac{i, j \in I \quad k \in \tilde{k} \quad \Gamma \vdash_A P_j \triangleright \Delta \cdot \tilde{k}@A : \alpha_j}{\Gamma \vdash_A \vec{k} \lhd l_i.\ P \triangleright \Delta \cdot \tilde{k}@A : k \rhd \oplus\{l_i : \alpha_i\}}\ \text{E.TSel} \qquad \frac{\Gamma, X : \Delta \vdash_A P \triangleright \Delta}{\Gamma \vdash_A \mu X.\ P \triangleright \Delta}\ \text{E.TRec} \qquad \frac{}{\Gamma \vdash_A \mathbf{0} \triangleright \emptyset}\ \text{E.TInact}$$

$$\frac{k \in \tilde{k} \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{k}@A : \alpha \quad \Gamma \vdash x : \theta}{\Gamma \vdash_A k?(x).\ P \triangleright \Delta \cdot \tilde{k}@A : k \downarrow \theta.\ \alpha}\ \text{E.TIn} \qquad \frac{k \in \tilde{k} \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{k}@A : \alpha \quad \Gamma \vdash e : \theta}{\Gamma \vdash_A k!\langle e\rangle.\ P \triangleright \Delta \cdot \tilde{k}@A : k \uparrow \theta.\ \alpha}\ \text{E.TOut}$$

$$\frac{\Gamma \vdash e@A : \text{bool} \quad \Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash_A Q \triangleright \Delta}{\Gamma \vdash \textbf{if}\ e@A\ \textbf{then}\ P\ \textbf{else}\ Q \triangleright \Delta}\ \text{E.TIf} \qquad \frac{\Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash_A Q \triangleright \Delta}{\Gamma \vdash_A P \oplus Q \triangleright \Delta}\ \text{E.TSum}$$

$$\frac{\Gamma \vdash_A P \triangleright \Delta \quad \{\tilde{k}\} \cap \text{fsc}(\Delta) = \emptyset}{\Gamma \vdash_A P \triangleright \Delta, \tilde{k}@A : \text{end}}\ \text{E.TEnd} \qquad \frac{\Gamma_1 \vdash_A P \triangleright \Delta_1 \quad \Gamma_2 \vdash_A Q \triangleright \Delta_2 \quad \Gamma_1 \asymp \Gamma_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \odot \Gamma_2 \vdash_A P \mid Q \triangleright \Delta_1 \odot \Delta_2}\ \text{E.TPar}$$

$$\frac{\Gamma \vdash_A P \triangleright \Delta \quad \{\tilde{k}\} \cap \text{fsc}(\Delta) = \emptyset}{\Gamma \vdash_A P \triangleright \Delta, \tilde{k} : \bot}\ \text{E.TBot} \qquad \frac{\Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash_A A[P]_\sigma \triangleright \Delta}\ \text{E.TPart} \qquad \frac{}{\Gamma \vdash_A \varepsilon \triangleright \emptyset}\ \text{E.TInactNW}$$

$$\frac{\Gamma \vdash N \triangleright \Delta \quad \tilde{k} \cap fsc(\Delta) = \emptyset}{\Gamma \vdash N \triangleright \Delta \cdot \tilde{k} : \bot}\ \text{E.TBotN} \qquad \frac{\Gamma \vdash N \triangleright \Delta \quad \{\tilde{k}\} \cap \text{fsc}(\Delta) = \emptyset}{\Gamma \vdash_A N \triangleright \Delta, \tilde{k}@A : \text{end}}\ \text{E.TEndN}$$

$$\frac{\Gamma_2 \vdash N_1 \triangleright \Delta_1 \quad \Gamma_1 \vdash N_2 \triangleright \Delta_2 \quad \Gamma_1 \asymp \Gamma_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \odot \Gamma_2 \vdash N_1 \mid N_2 \triangleright \Delta_1 \odot \Delta_2}\ \text{E.TParN}$$

Figure 6.14: End Point Calculus: Typing rules

3. $\Gamma_1 \odot \Gamma_2$, defined whenever $\Gamma_1 \asymp \Gamma_2$, is the minimum service typing such that:
   (a) if $a@A : \alpha \in \Gamma_i$ then $a@A : \alpha \in \Gamma_1 \odot \Gamma_2$;
   (b) if $\overline{a}@A : \alpha \in \Gamma_i$ and $a@A : \alpha \notin \Gamma_j$ then for $i \neq j$, $\overline{a}@A : \alpha \in \Gamma_1 \odot \Gamma_2$;
   (c) if $x@A : \theta \in \Gamma_i$ ($X : \Delta \in \Gamma_i$) then $x@A : \theta \in \Gamma_1 \odot \Gamma_2$ ($X : \Delta \in \Gamma_1 \odot \Gamma_2$).
4. $\Delta_1 \odot \Delta_2$, defined whenever $\Delta_1 \asymp \Delta_2$, is the minimum session typing such that:
   (a) if $\tilde{k}@A \in \text{dom}(\Delta_i) \backslash \text{dom}(\Delta_j)$ for $i \neq j$, then $\tilde{k}@A : \Delta(\tilde{k}@A) \in \Delta_1 \odot \Delta_2$;
   (b) if $\tilde{k} \in \text{dom}(\Delta_i) \backslash \text{dom}(\Delta_j)$ for $i \neq j$, then $\tilde{k} : \bot \in \Delta_1 \odot \Delta_2$;
   (c) if $\tilde{k}@A : \alpha \in \Delta_i$ and $\tilde{k}@A : \beta \in \Delta_j$ for $i \neq j$, then $\tilde{k}@A : \alpha \mid \beta \in \Delta_1 \odot \Delta_2$;
   (d) if $\tilde{k}@A \in \text{dom}(\Delta_i)$ and $\tilde{k}@B \in \text{dom}(\Delta_j)$ for $i \neq j$, then $\tilde{k} : \bot \in \Delta_1 \odot \Delta_2$.

$$! a(k).\ P \sqcup ! a(k).\ Q \stackrel{\text{def}}{=} ! a(k).\ (P \sqcup Q)$$

$$\overline{a}\langle k\rangle.\ P \sqcup \overline{a}\langle k\rangle.\ Q \stackrel{\text{def}}{=} \overline{a}\langle k\rangle.\ (P \sqcup Q)$$

$$k!\langle x\rangle.\ P \sqcup k!\langle x\rangle.\ Q \stackrel{\text{def}}{=} k!\langle x\rangle.\ (P \sqcup Q)$$

$$k?(y)\,.\ P \sqcup k?(y)\,.\ Q \stackrel{\text{def}}{=} k?(y)\,.\ (P \sqcup Q)$$

$$k \rhd \{\Sigma_{i\in I} \mathsf{l}_i.\ P_i\} \sqcup k \rhd \{\Sigma_{i\in J} \mathsf{l}_i.\ Q_i\} \stackrel{\text{def}}{=} k \rhd \left\{ \begin{array}{l} \Sigma_{i\in I\cap J} \mathsf{l}_i.\ P_i \sqcup Q_i \\ +\Sigma_{i\in I\setminus J} \mathsf{l}_i.\ P_i \\ +\Sigma_{i\in J\setminus I} \mathsf{l}_i.\ Q_i \end{array} \right\}$$

$$k \lhd l.\ P \sqcup k \lhd l.\ Q \stackrel{\text{def}}{=} k \lhd l.\ (P \sqcup Q)$$

$$\textbf{if } e \textbf{ then } P_1 \textbf{ else } P_2 \ \sqcup \textbf{if } e \textbf{ then } Q_1 \textbf{ else } Q_2 \stackrel{\text{def}}{=} \textbf{if } e \textbf{ then } (P_1 \sqcup Q_1) \textbf{ else } (P_2 \sqcup Q_2)$$

$$(P_1 \mid P_2) \sqcup (P_3 \mid P_4) \stackrel{\text{def}}{=} (P_1 \sqcup P_3) \mid (P_2 \sqcup P_4)$$

$$(P_1 \oplus P_2) \sqcup (Q_1 \oplus Q_2) \stackrel{\text{def}}{=} (P_1 \sqcup Q_1) \oplus (P_2 \sqcup Q_2)$$

$$\mu X.\ P \sqcup \mu X.\ Q \stackrel{\text{def}}{=} \mu X.\ (P \sqcup Q)$$

$$X \sqcup X \stackrel{\text{def}}{=} X$$

$$P \sqcup \mathbf{0} \stackrel{\text{def}}{=} P$$

$$P \sqcup Q \stackrel{\text{def}}{=} P' \sqcup Q' \qquad (P \equiv P', Q \equiv Q')$$

Figure 6.15: End-Point Projection: Merging Rules

Appendix 6.E End Point Projection: Merging

Definition 6.E.1 (Merge Operator). *$P \sqcup Q$ is a partial commutative binary operator on typed processes which is well-defined iff $P \bowtie Q$ and satisfies the rules in Figure 6.15, where, in the right-hand side of each rule, we assume that each application of the operator to, say, P and Q, is such that $P \bowtie Q$.*

Appendix 6.F End Point Projection: Thread Projection

Definition 6.F.1 (Thread Projection). Given a consistently annotation $\mathcal{A}$, the partial operation $\mathrm{TP}(\mathcal{A}, t)$ is defined as:

$$\mathrm{TP}(A^{t_1} \to B^{t_2} : b(\tilde{k}).\ \mathcal{A}, t) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \overline{b}\langle\tilde{k}\rangle.\ \mathrm{TP}(\mathcal{A}, t_1) & \text{if } t = t_1 \\ ! b(\tilde{k}).\ \mathrm{TP}(\mathcal{A}, t_2) & \text{if } t = t_2 \\ \mathrm{TP}(\mathcal{A}, t) & \text{otherwise} \end{array} \right.$$

$$\mathrm{TP}(A^{t_1}\!\to\!B^{t_2} : k\langle e, x\rangle.\ \mathcal{A}, t) \stackrel{\text{def}}{=} \begin{cases} \overline{k}!\langle e\rangle.\ \mathrm{TP}(\mathcal{A}, t) & \text{if } t = t_1 \\ k?(x).\ \mathrm{TP}(\mathcal{A}, t) & \text{if } t = t_2 \\ \mathrm{TP}(\mathcal{A}, t) & \text{otherwise} \end{cases}$$

$$\mathrm{TP}(A^{t_1}\!\to\!B^{t_2}\!:k[l_i : \mathcal{A}_i]_{i\in I}, t) \stackrel{\text{def}}{=} \begin{cases} k \lhd l_i.\ \mathrm{TP}(\mathcal{A}_i, t) & \text{if } t = t_1 \\ k \rhd \{\textstyle\sum_i l_i\}\ .\ \mathrm{TP}(\mathcal{A}_i, t) & \text{if } t = t_2 \\ \mathrm{TP}(\mathcal{A}, t) & \text{otherwise} \end{cases}$$

$$\mathrm{TP}(\mathbf{if}\ e@A^{t'}\ \mathbf{then}\ \mathcal{A}_1\ \mathbf{else}\ \mathcal{A}_2\ , t) \stackrel{\text{def}}{=} \begin{cases} \mathbf{if}\ e\ \mathbf{then}\ \mathrm{TP}(\mathcal{A}_1, t')\ \mathbf{else}\ \mathrm{TP}(\mathcal{A}_2, t') & \text{if } t = t' \\ \mathrm{TP}(\mathcal{A}_1, t) \sqcup \mathrm{TP}(\mathcal{A}_2, t) & \text{otherwise} \end{cases}$$

$$\mathrm{TP}(\mathcal{A}_1 |^{t'} \mathcal{A}_2, t) \stackrel{\text{def}}{=} \mathrm{TP}(A_1, t') \mid \mathrm{TP}(A_2, t')$$

$$\mathrm{TP}(\mu^{t':\{\tilde{t}_i\}} X^A.\ \mathcal{A}, t) \stackrel{\text{def}}{=} \mu X.\ \mathrm{TP}(\mathcal{A}, t) \text{ if } t \in \{\tilde{t}_i\},\ \mathrm{TP}(\mathcal{A}, t) \text{ otherwise.}$$

$$\mathrm{TP}(X^A_{t:\{\tilde{t}_i\}}, t) \stackrel{\text{def}}{=} X \text{ if } t \in \{\tilde{t}_i\},\ \mathbf{0} \text{ otherwise.}$$

$$\mathrm{TP}(\mathbf{0}, t) \stackrel{\text{def}}{=} \mathbf{0}.$$

If $\mathrm{TP}(\mathcal{A}, t)$ is undefined then we set $\mathrm{TP}(\mathcal{A}, t) = \bot$.

Above, we augment consistent annotations with a further annotation for recursions $\mu^{t:\{t_i\}} X$ and recursion variables $X^A_{t:\{t_i\}}$, with $\{t_i\}$ be the set of threads occurring in, but not initiated in, $\mathcal{A}$ (a thread is initiated in $\mathcal{A}$ whenever it occurs passive in a session initiation).

Chapter 7

Time and Exceptional Behaviour in Multiparty Structured Interactions

Abstract: The Conversation Calculus (CC) is an extension of the π-calculus, intended as a model of multiparty interactions. The CC is built upon the notion of *conversation*—a possibly distributed medium in which participants may communicate. We study the interplay of time and exceptional behavior for models of structured communications based on conversations. We propose C3, a *timed* variant of the CC in which conversations feature both standard and exceptional behavior. The exceptional behavior may be triggered either by the passing of time (a timeout) or by an explicit signal for conversation abortion. We present a compelling example from a healthcare scenario, and argue that the combination of time and exceptional behavior greatly enhances the significance and level of detail of specifications of structured communications.

Contents

7.1 Introduction

This paper is an initial step in understanding how *time* and forms of *exceptional behavior* can be jointly captured in models of multiparty structured communications.

Time usually plays a crucial rôle in practical scenarios of structured communication. Consider, for instance, a web banking application: interaction between a user and her bank generally takes place by means of a secure session, which is meant to expire after a certain period of inactivity of the user. When that occurs, she must exhibit again her authentication credentials, so as to initiate another session or to continue with the expired session. In some cases, the session (or parts of it) has a predetermined duration and so interactions may also be bounded in time. The user may need to reinitiate the session if, for instance, her network connection is too slow. Crucially, the different incarnations of time in interactions (session durations, timeouts, delays) can be seen to be closely related to the behavior of the system in exceptional circumstances. A specification of the web banking application above would appear incomplete unless one specifies how the system should behave when, e.g., the session has expired and the user attempts to reinitiate it, or when interaction is taking longer than anticipated.

In real scenarios of structured communications, time then appears to go hand in hand with *exceptional behavior*. This observation is particularly evident in *healthcare scenarios* [Lyng *et al.* 2009]—a central source of motivation for our work. In healthcare scenarios, structured communications often involve *strict time bounds*, as in, e.g., "monitor the patient every two hours, for the next 48 hours". They may also include interaction patterns defined as both a *default behavior* and an *alternative behavior* to be executed in case of unexpected conditions: "contact a substitute doctor if the designated doctor cannot be reached within 15 minutes". Also, scenarios involve tasks that may be *suspended* or *aborted*, as in, e.g.,"stop administering the medicine if the patient reacts badly to it".

Unfortunately, expressing appropriately the interplay of time and exceptional behavior turns out to be hard in known formalisms for structured communications. In fact, although some of such formalisms have been extended with constructs for exceptional behavior (see, e.g., [Carbone *et al.* 2008, Caires *et al.* 2008, Capecchi *et al.* 2010]), to the best of our knowledge none of these works considers constructs for timed behavior. To overcome this lack, here we introduce C3, a model of structured communications that integrates time and exceptional behavior in the context of multiparty interactions. C3 arises as an extension of the Conversation Calculus (CC) [Vieira *et al.* 2008, Vieira 2010] in which conversations have durations and are sensible to compensations. Below, we first present the CC by means of a running example; then, we introduce C3 by enhancing the example with time and exceptional behavior.

The CC is an interesting base language for our study. First, it is a *simple* model: it corresponds to a π-calculus [Milner *et al.* 1992] extended with *conversation contexts* (see below). Hence, the definition of C3 can take advantage of previous works on extensions of the π-calculus with time and forms of exceptional behavior (see, e.g., [Berger & Honda 2000, Ferreira *et al.* 2010]). Second, the CC counts with a number of

$$
\begin{aligned}
&\text{Buyer} \blacktriangleleft [\textbf{new}\ \text{Seller} \cdot \text{BuyService} \Leftarrow \text{buy}^{\downarrow}!(prod).\text{price}^{\downarrow}?(p).\text{details}^{\downarrow}?(d)] \\
&\mid \text{Seller} \blacktriangleleft [\text{PriceDB} \mid\ \textbf{def}\ \text{BuyService} \Rightarrow \text{buy}^{\downarrow}?(prod).\text{askPrice}^{\uparrow}!(prod). \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{priceVal}^{\uparrow}?(p).\text{price}^{\downarrow}!(p). \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{join}\ \text{Shipper} \cdot \text{DelivService} \Leftarrow \text{product}^{\uparrow}!(prod)] \\
&\mid \text{Shipper} \blacktriangleleft [\textbf{def}\ \text{DelivService} \Rightarrow \text{product}^{\downarrow}?(p).\text{details}^{\downarrow}!(data)]
\end{aligned}
$$

Figure 7.1: The purchasing scenario in CC.

reasoning techniques to build upon, in particular so-called *conversation types* [Caires & Vieira 2010]. Third, and most importantly, the CC allows for the specification of *multiparty interactions*, which are ubiquitous in many practical settings.

Fig. 7.1 gives a CC specification of the well-known *purchasing scenario* [Carbone *et al.* 2007, Vieira 2010]. This scenario describes the interaction of a buyer and a seller for buying a given product; the seller later involves a shipper who is in charge of delivering the product. In the CC, a *conversation context* represents a distributed communication medium where two or more partners may interact. Process $n \blacktriangleleft [P]$ is the conversation context with behavior P and identity n; process P may seamlessly interact with processes contained in any other conversation context named n. The model in Fig. 7.1 thus involves three participants: Buyer, Seller, and Shipper. Buyer invokes a new instance of the BuyService service, defined by Seller. As a result, a *conversation* on a fresh name is established between them; this name can then be used to exchange information on the product and its price (the latter is retrieved by Seller from the database PriceDB). When the transaction has been agreed, Shipper joins in the conversation, and receives product information from Seller and delivery details from Buyer. The model in Fig. 7.1 relies on the following *service idioms* which, interestingly, can be derived from the basic syntax of the CC:

$\textbf{def}\ s \Rightarrow P$	$\stackrel{\texttt{def}}{=}$	$\text{s}^{\downarrow}?(x).x \blacktriangleleft [P]$	Define a service s with behavior P
$\textbf{new}\ n \cdot s \Leftarrow Q$	$\stackrel{\texttt{def}}{=}$	$(\nu c)(n \blacktriangleleft [\text{s}^{\downarrow}!(c)] \mid c \blacktriangleleft [Q])$	Create instance of a service s located at n
$\textbf{join}\ n \cdot s \Leftarrow Q$	$\stackrel{\texttt{def}}{=}$	$\textbf{this}(x).(n \blacktriangleleft [s^{\downarrow}!(x)] \mid Q)$	Join instance of service s located at n

The main design decision in defining C3 is considering time and exceptional behavior *directly* into conversation contexts: C3 features *timed*, *compensable* conversation contexts, denoted as $n \blacktriangleleft [P\,;\, Q]^{t}_{k}$. As before, n is the identity of the conversation context. Process P describes the *default* behavior for n, which is performed while the *duration* t is greater than 0. Observable actions from P witness the time passage in n;

as soon as $t = 0$, the default behavior is dynamically replaced by Q, the *compensating* behavior. Name κ represents an explicit *abort* mechanism: the interaction of $n \blacktriangleleft [P\,;\, Q]^t_\kappa$ with a *kill prefix* $\kappa^\dagger$ immediately sets t to 0.

An immediate and pleasant consequence of our extended conversation contexts is that the signature of the service idioms (given above) can be extended too. Hence, C3 specifications can express richer information on timeouts and exceptional behavior. It suffices to extend the idioms representing *timed* service definition and instantiation:

$$\textbf{def}\, s\, \textbf{with}\, (\kappa, t) \Rightarrow \{P; Q\} \quad \stackrel{\texttt{def}}{=} \quad \mathrm{s}^{\downarrow}?(y).\ y \blacktriangleleft [P\,;\, Q]^t_\kappa \qquad \text{Timed service definition}$$

$$\textbf{new}\, n \cdot s\, \textbf{with}\, (\kappa, t) \Leftarrow \{P; Q\} \quad \stackrel{\texttt{def}}{=} \quad (\nu c)\ (n \blacktriangleleft [\mathrm{s}^{\downarrow}!(c)] \mid c \blacktriangleleft [P\,;\, Q]^t_\kappa) \qquad \text{Timed service instantiation}$$

In the former we assume y and c are fresh in P and Q, and different from κ, t, n, while in the latter $n \blacktriangleleft [\mathrm{s}^{\downarrow}!(c)]$ stands for $n \blacktriangleleft [\mathrm{s}^{\downarrow}!(c)\,;\, \mathbf{0}]^\infty_\emptyset$. This way, we are able to define timed, compensable extensions for service definition and instantiation idioms; they rely on a compensation signal κ, a timeout value t, and a compensating protocol definition Q. As a simple example, the C3 processes

$$\begin{array}{l} \mathrm{Client} \blacktriangleleft [\textbf{new}\, \mathrm{Provider} \cdot \mathrm{Service}\, \textbf{with}\, (\kappa_c, t_c) \Leftarrow \{P; Q\}] \\ \mathrm{Provider} \blacktriangleleft [\textbf{def}\, \mathrm{Service}\, \textbf{with}\, (\kappa_p, t_p) \Rightarrow \{R; T\}] \end{array}$$

may interact and evolve into $(\nu s)\,(\mathrm{Client} \blacktriangleleft [s \blacktriangleleft [P\,;\, Q]^{t_c}_{\kappa_c}] \mid \mathrm{Provider} \blacktriangleleft [s \blacktriangleleft [R\,;\, T]^{t_p}_{\kappa_p}])$.

Some related approaches (e.g. [Carbone *et al.* 2008]) distinguish the behavior originated in the standard definition of a service from the behavior associated to related compensating activities. In those works, the objective is to return to the standard control flow by orderly escaping from compensating activities; handling nested compensations thus becomes a delicate issue. In contrast, we do not enforce such a distinction: we believe that in many realistic scenarios the main goal is timely availability of services; hence, the actual origin of the offered services should be transparent to the users. This way, e.g., for the users of a web banking application, interacting with the main server or with one of its backups is irrelevant as long as they receive the required services.

We illustrate these ideas by considering an extended version of the purchase scenario in C3; see Fig. 7.2. Suppose a buyer who is willing to interact with a specific provider only for a finite amount of time. She first engages in conversations with several providers at the same time; then, she picks the provider with the best offer, abandoning the conversations with the other providers. In the model, $\star P$ denotes the replicated version of process P, with the usual semantics. We consider one buyer and three sellers. NewBuyer creates three instances of the BuyService service, one from each seller. The part of each such instances residing at NewBuyer can be aborted by suitable messages on c_i. The part of the protocol for BuyService that resides at NewBuyer is similar as before, and is extended with an output signal com_i which

$$
\begin{array}{l}
\mathsf{NewBuyer} \blacktriangleleft [\prod_{i\in[1..3]} \ \textbf{new}\ \mathsf{Seller}_i \cdot \mathsf{BuyService} \\
\qquad\qquad\qquad\qquad \textbf{with}\,(c_i, v_i) \Leftarrow \{ \\
\qquad\qquad\qquad\qquad\qquad\qquad P_i \\
\qquad\qquad\qquad\qquad\qquad\qquad ; Q_i\} \\
\qquad\qquad\qquad\quad \mid \mathsf{Control} \\
\qquad\qquad\quad ; \mathsf{CancelOrder}]_x^{tmax} \\
\mid \prod_{i\in[1..3]} \ \mathsf{Seller}_i \blacktriangleleft [\mathsf{PriceDB} \\
\qquad\qquad\qquad\quad \mid\ \textbf{def}\ \mathsf{BuyService} \\
\qquad\qquad\qquad\qquad \textbf{with}\,(b_i, w_i) \Rightarrow \{ \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{offer}^{\downarrow}?(prod). \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{askPrice}^{\uparrow}!(prod). \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{priceVal}^{\uparrow}!(p). \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{price}^{\downarrow}!(p). \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{join}\ \mathsf{Shipper} \cdot \mathsf{DelivService} \Leftarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{product}^{\uparrow}!(prod) \\
\qquad\qquad\qquad\qquad\qquad\qquad ; R_i\} \\
\qquad\qquad\quad ;\ \mathsf{CancelSell}_i]_{x_i}^{t_i} \\
\mid \mathsf{Shipper} \blacktriangleleft [\textbf{def}\ \mathsf{DelivService} \\
\qquad\qquad\quad \textbf{with}\,(d, t) \Rightarrow \{ \\
\qquad\qquad\qquad\qquad\quad \mathsf{product}^{\downarrow}?(p). \\
\qquad\qquad\qquad\qquad\quad \mathsf{details}^{\downarrow}!(data) \\
\qquad\qquad\qquad\qquad\quad ; T\} \\
\qquad\qquad\quad ; \mathbf{0}]_z^{t_3}
\end{array}
$$

$$
\text{where} \quad P_i \stackrel{\textbf{def}}{=} \mathsf{offer}^{\downarrow}!(prod).\ \mathsf{price}^{\downarrow}?(p).\ \mathsf{com_i}^{\uparrow}!(p).\ \mathsf{details}^{\downarrow}?(d)
$$

$$
\begin{array}{rl}
\mathsf{Control} \stackrel{\textbf{def}}{=} & \star\,(\mathsf{com}_1{}^{\uparrow}?(p).\ (c_2^{\dagger} \mid c_3^{\dagger}) \\
& \quad +\mathsf{com}_2{}^{\uparrow}?(p).\ (c_1^{\dagger} \mid c_3^{\dagger}) \\
& \quad +\mathsf{com}_3{}^{\uparrow}?(p).\ (c_1^{\dagger} \mid c_2^{\dagger}))
\end{array}
$$

Figure 7.2: The purchasing scenario in C3.

allows to commit the selection of seller i. The commitment to one particular seller (and the discard of the rest) is implemented by process Control. The duration of NewBuyer is given by $tmax$; its compensation activity (CancelOrder) is left unspecified. Seller_i follows the lines of the basic scenario, extended with compensation signals y_i which trigger the compensation process $\mathsf{CancelSell}_i$. Notice that while Q_i controls the failure of the i-th service invoked by NewBuyer, CancelOrder is meant to control the failure of NewBuyer as a whole.

This extended example illustrates two of the features of C3: explicit conversation abortion and conversations bounded in time. The first one can be appreciated in the

selection implemented by Control, which ensures that only one provider will be able to interact with NewBuyer, by explicitly aborting the conversations at NewBuyer with the other two providers. However, Control only takes care of the interactions at the buyer side; there are also conversation pieces at each Seller_i, which are not under the influence of Control (we assume $c_i \neq w_i$). The "garbage-collection" of such pieces is captured by the second feature: since such conversations are explicitly defined with the time bound w_i, they will be automatically collected (i.e. aborted) after w_i time units. That is, the passing of time avoids "dangling" conversation pieces. This example reveals the complementarity between the explicit conversation abortion (achieved via abortion signals) and the more implicit conversation abortion associated to the passing of time.

In Section 7.2 we summarize the main definitions of the CC. Section 7.3 introduces the syntax and semantics of C3. In Section 7.4 we discuss its expressiveness, by comparing it to some other related languages. Then, in Section 7.5, we present a compelling example from the healthcare domain, that is analysed on the light of refinement relations for C3 in Section 7.6. We review related work in Section 7.7; some concluding remarks are given in Section 7.8.

7.2 The Conversation Calculus

Here we briefly introduce the Conversation Calculus (CC, in the following). Further details can be found at [Vieira *et al.* 2008, Vieira 2010].

The CC corresponds to a π-calculus with *labeled* communication and extended with *conversation contexts*. A conversation context can be seen as a medium in which interactions take place. It is similar to sessions in service-oriented calculi (see [Honda *et al.* 1998]) in the sense that every conversation context has an unique identifier (e.g.: an URI). Interactions in CC may be intuitively seen as communications in a pool of messages, where the pool is divided in areas identified by conversation contexts. Multiple participants can access many conversation contexts concurrently, provided they can get hold of the name identifying the context. Moreover, conversations can be nested multiple times (for instance, a private chat room within a multi-user chat application).

Definition 7.2.1 (CC Syntax). *Let $\mathcal{N}$ be an infinite set of names. Also, let $\mathcal{L}$, $\mathcal{V}$, and χ be infinite sets of labels, variables, and recursion variables, respectively. Using d to range over $\uparrow$ and $\downarrow$, the set of actions α and processes P is given below:*

$$\begin{aligned}\alpha ::= & \; \mathrm{l}^d!(\vec{n}) \\ & \mid \mathrm{l}^d?(\vec{x}) \\ & \mid \mathbf{this}(x)\end{aligned}$$

$$
\begin{aligned}
P, Q ::= & \; n \blacktriangleleft [P] \\
& \mid \sum_{i \in I} \alpha_i . \; P_i \\
& \mid P \mid Q \\
& \mid (\nu n) \; P \\
& \mid \mu X . \; P \\
& \mid X
\end{aligned}
$$

Above, $\vec{n}$ and $\vec{x}$ denote tuples of names and variables in $\mathcal{N}$ and $\mathcal{V}$, respectively. Actions can be an output $\mathsf{l}^d!(\vec{n})$ or an input $\mathsf{l}^d?(\vec{x})$, as in the π-calculus, with $\mathsf{l} \in \mathcal{L}$ in both cases. The *message direction* $\downarrow$ (read "here") decrees that the action it is associated to should take place in the *current* conversation context, while $\uparrow$ (read "up") decrees that the action should take place in the *enclosing* one. We often omit the "here" direction, and write $\mathsf{l}?(y).P$ and $\mathsf{l}!(\vec{n}).P$ rather than $\mathsf{l}^{\downarrow}?(y).P$ and $\mathsf{l}^{\downarrow}!(\vec{n}).P$. The context-aware prefix $\mathbf{this}(x)$ binds the name of the enclosing conversation context to x. The syntax of processes includes the conversation context $n \blacktriangleleft [P]$, where $n \in \mathcal{N}$. We follow the standard π-calculus interpretation for guarded choice, parallelism, restriction, and recursion (for which we assume $X \in \chi$). As usual, given $\sum_{i \in I} \alpha_i . \; P_i$, we write $\mathbf{0}$ when $|I| = 0$, and $\alpha_1 . \; P_1 + \alpha_2 . \; P_2$ when $|I| = 2$. We assume the usual definitions of free/bound variables and free/bound names for a process P, noted $fv(P), bv(P)$ and $fn(P), bn(P)$, respectively. The set of names of a process is defined as $n(P) = fn(P) \cup bn(P)$. Finally, notice that labels in $\mathcal{L}$ are not subject to restriction or binding.

The semantics of the CC is given as a labeled transition system (LTS). As customary, a transition $P \xrightarrow{\lambda} P'$ represents the evolution from P to P' through action λ. We write $P \xrightarrow{\lambda}$ if $P \xrightarrow{\lambda} P'$, for some P'. We define $P \longrightarrow P'$ as $P \xrightarrow{\tau} P'$. We use $P \longrightarrow^* P'$ to denote the transitive closure of $P \longrightarrow P'$, and write $P \stackrel{\lambda}{\Rightarrow} P'$ when $P \longrightarrow^* \xrightarrow{\lambda} \longrightarrow^* P'$.

Definition 7.2.2. *Transition* labels λ *are defined in terms of* actions σ*, as defined by the following grammar:*

$$
\sigma ::= \tau \;\mid\; \mathsf{l}^d?(\vec{x}) \;\mid\; \mathsf{l}^d!(\vec{n}) \;\mid\; \mathsf{this} \qquad \lambda ::= \sigma \;\mid\; c\,\sigma \;\mid\; (\nu n)\,\lambda
$$

Action τ denotes internal communication, while $\mathsf{l}^d?(\vec{x})$ and $\mathsf{l}^d!(\vec{n})$ represent an input and output to the environment, respectively. Action this represents a conversation identity access. A transition label λ can be either the (unlocated) action σ, an action σ *located at* conversation c (written $c\,\sigma$), or a transition label in which n is bound with scope λ. This is the case of bounded output actions. $out(\lambda)$ denotes the names produced by a transition, so $out(\lambda) = a$ if $\lambda = \mathsf{l}^d!(a)$ or $\lambda = c\mathsf{l}^d!(a)$ and $c \neq a$. A transition label λ denoting communication, such as $\mathsf{l}^d?(\vec{x})$ or $\mathsf{l}^d!(\vec{n})$ is subject to *duality* $\overline{\lambda}$. We write $\overline{\mathsf{l}^d?(\vec{x})} = \mathsf{l}^d!(\vec{n})$ and $\overline{\mathsf{l}^d!(\vec{n})} = \{\mathsf{l}^d?(\vec{x}) \mid \vec{x} \in \mathcal{V}\}$.

Fig. 7.3 presents the LTS. There, $\equiv$ stands for a structural congruence relation on processes; see [Vieira 2010] for details. The rules in the upper part of Fig. 7.3

$$\frac{}{\lfloor^d?(\vec{x}).\ P \xrightarrow{\lfloor^d?(\vec{n})} P[\vec{n}/\vec{x}]}\ (\text{CC-In}) \qquad \frac{}{\lfloor^d!(\vec{n}).\ P \xrightarrow{\lfloor^d!(\vec{n})} P}\ (\text{CC-Out}) \qquad \frac{}{\mathbf{this}(x).\ P \xrightarrow{c\ \mathtt{this}} P[c/x]}\ (\text{CC-This})$$

$$\frac{P \xrightarrow{\lambda} Q \quad n \in out(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} Q}\ (\text{CC-Open}) \qquad \frac{P \xrightarrow{\lambda} Q \quad n \notin n(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} (\nu n)\ Q}\ (\text{CC-Res}) \qquad \frac{\alpha_j.\ P_j \xrightarrow{\lambda} P'_j \quad j \in I}{\sum_{i\in I} \alpha_i.\ P_i \xrightarrow{\lambda} P'_j}\ (\text{CC-Sum})$$

$$\frac{P \xrightarrow{\lambda} P' \quad bn(\lambda)\#fn(Q)}{P \mid Q \xrightarrow{\lambda} P' \mid Q}\ (\text{CC-Par1}) \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ (\text{CC-Comm1}) \qquad \frac{P[X/\mu X.\ P] \xrightarrow{\lambda} Q}{\mu X.\ P \xrightarrow{\lambda} Q}\ (\text{CC-Rec})$$

$$\frac{P \xrightarrow{(\nu \vec{n})\ \overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad \vec{n}\#fn(Q)}{P \mid Q \xrightarrow{\tau} (\nu \vec{n})\ (P' \mid Q')}\ (\text{CC-Close1}) \qquad \frac{P \xrightarrow{\lambda^{\downarrow}} P'}{c \blacktriangleleft [P] \xrightarrow{c\ \lambda^{\downarrow}} c \blacktriangleleft [P']}\ (\text{CC-LocL}) \qquad \frac{P \xrightarrow{\lambda^{\uparrow}} P'}{c \blacktriangleleft [P] \xrightarrow{\lambda^{\downarrow}} c \blacktriangleleft [P']}\ (\text{CC-HereL})$$

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu n)\ c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} (\nu n)\ (P' \mid Q')}\ (\text{CC-ThisClose1}) \qquad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} P' \mid Q'}\ (\text{CC-ThisComm1}) \qquad \frac{P \xrightarrow{a\ \lambda^{\downarrow}} P'}{c \blacktriangleleft [P] \xrightarrow{a\ \lambda^{\downarrow}} c \blacktriangleleft [P']}\ (\text{CC-ThruL})$$

$$\frac{P \xrightarrow{\tau} P'}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [P']}\ (\text{CC-TauL}) \qquad \frac{P \xrightarrow{c\ \mathtt{this}} P'}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [P']}\ (\text{CC-ThisLocL})$$

Figure 7.3: An LTS for CC. Rules with labels ending with "1" have a symmetric counterpart (with label ending with "2") which is elided.

follow the transition rules for a π-calculus with recursion. For instance, rule (CC-Open) corresponds to the usual scope extrusion rule in the π-calculus. The rest of the rules are specific to the CC. Rule (CC-This) captures the name of an enclosing conversation context. Rule (CC-LocL) locates an action to a particular conversation context, and rule (CC-HereL) changes the direction of an action occurring inside a context. Rules (CC-ThisClose1) and (CC-ThisComm1) are located versions of (CC-Close) and (CC-Comm), respectively. Rule (CC-ThisLocL) hides an action occurring inside a conversation context. Rules (CC-ThruL) and (CC-TauL) formalize how actions change when they "cross" a conversation context.

7.3 C3: CC + Time + Compensations

As anticipated in the Introduction, the syntax of C3 extends that of the CC with timed, compensable conversation contexts and a process for aborting running conversations:

Definition 7.3.1 (C3 Syntax). *The syntax of* C3 *processes is obtained from that given in Definition 7.2.1 by replacing the conversation contexts* $n \blacktriangleleft [P]$ *with* $n \blacktriangleleft [P\,;\, Q]^t_\kappa$ *(with* $n, \kappa \in \mathcal{N}$ *and* $t \in \mathbb{N}_0 \cup \{\infty\}$*) and by adding* $\kappa^\dagger$ *to the grammar of processes.*

Every notational convention introduced for CC processes carries over to C3 processes. In particular, as in the CC, notice that labels in $\mathcal{L}$ are not subject to restriction or binding. Unlike the LTS of CC, however, we assume a relation of structural congruence with the usual axioms for the π-calculus only (i.e., axioms for α-conversion, parallel composition, restriction, and the inactive process). In particular, because of the timed nature of conversation contexts in C3 (on which we comment below), we refrain from adopting the axioms for manipulation of conversation contexts given in [Vieira 2010].

The notion of time in C3 is relative to each conversation context: it serves as a bound on the duration of the interactions *contained* in it. The time signature $t+1$ in a conversation context $c \blacktriangleleft [P\,;\, Q]^{t+1}_\kappa$ can evolve into t if the enclosing process P executes a "standard" action (i.e.: any action except a compensation), or to 0 in case P fires a compensation. Hence, time in C3 is inherently *local* to each conversation context, rather than *global* to the whole system. We find this rather fine account of time in accordance with the intention of conversation contexts—distributed pieces of behavior in which the whole communication is organized. Put differently, since conversation contexts are essentially distributed abstractions of the participants of the multiparty interaction, considering a time signature local to each of them is a way of enforcing distribution. Also, as shown by our examples, this notion of time is convenient for the interplay with exceptional behavior.

The LTS for C3 is defined by the rules in Fig. 7.4; transition labels are obtained by extending the set of *actions* σ of the LTS of CC with a new action $\kappa^\dagger$. The convention on rule names for symmetric counterparts given in the LTS of the CC carries over to the LTS of C3. Moreover, for each of the left rules in Fig. 7.4—which describe evolution in the default behavior and have rule names ending in "L"—, there is an elided right rule characterizing evolution in the compensation behavior.

The passage of time in C3 is governed by the time elapsing function below. Intuitively, one time unit passes by as a consequence of the action. (Actions with durations different from one can be easily accommodated.)

Definition 7.3.2 (Time-elapsing function). *Given a* C3 *process* P*, we use* $\phi(P)$ *to denote the function that decreases the time bounds in* P*, inductively defined as:*

$$\phi(n \blacktriangleleft [Q\,;\, R]^{t+1}_\kappa) = n \blacktriangleleft [\phi(Q)\,;\, R]^t_\kappa \quad \phi(P \mid Q) = \phi(P) \mid \phi(Q) \quad \phi((\nu n)\, P) = (\nu n)\, \phi(P)$$
$$\phi(n \blacktriangleleft [Q\,;\, R]^0_\kappa) = n \blacktriangleleft [Q\,;\, \phi(R)]^0_\kappa \qquad \phi(P) = P \quad \textit{Otherwise.}$$

Given $k > 0$*, we define* $\phi^k(P) = \phi(P)$ *if* $k = 1$ *and* $\phi^k(P) = \phi(\phi^{k-1}(P))$*, otherwise.*

$$\frac{}{\lfloor^d?(\vec{x}).\ P \xrightarrow{\lfloor^d?(\vec{n})} P[\vec{n}/\vec{x}]}\ (\textsc{In}) \qquad \frac{}{\lfloor^d!(\vec{n}).\ P \xrightarrow{\lfloor^d!(\vec{n})} P}\ (\textsc{Out}) \qquad \frac{}{\mathbf{this}(x).\ P \xrightarrow{c\ \mathtt{this}} P[c/x]}\ (\textsc{This})$$

$$\frac{P \xrightarrow{\lambda} Q \quad n \in out(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} Q}\ (\textsc{Open}) \qquad \frac{P \xrightarrow{\lambda} Q \quad n \notin n(\lambda)}{(\nu n)\ P \xrightarrow{(\nu n)\ \lambda} (\nu n)\ Q}\ (\textsc{Res}) \qquad \frac{\alpha_j.\ P_j \xrightarrow{\lambda} P'_j \quad j \in I}{\sum_{i\in I} \alpha_i.\ P_i \xrightarrow{\lambda} P'_j}\ (\textsc{Sum})$$

$$\frac{P \xrightarrow{(\nu \vec{n})\ \overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad \vec{n}\#fn(Q)}{P \mid Q \xrightarrow{\tau} (\nu \vec{n})\ (P' \mid Q')}\ (\textsc{Close1}) \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ (\textsc{Comm1}) \qquad \frac{P[X/\mu X.\ P] \xrightarrow{\lambda} Q}{\mu X.\ P \xrightarrow{\lambda} Q}\ (\textsc{Rec})$$

$$\frac{P \xrightarrow{\lambda} P' \quad bn(\lambda)\#fn(Q)}{P \mid Q \xrightarrow{\lambda} P' \mid \phi(Q)}\ (\textsc{Par1}) \qquad \frac{P \xrightarrow{\sigma} P' \ Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} P' \mid Q'}\ (\textsc{ThisComm1}) \qquad \frac{P \xrightarrow{\sigma} P' \ Q \xrightarrow{(\nu n)\ c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} (\nu n)\ (P' \mid Q')}\ (\textsc{ThisClose1})$$

$$\frac{P \xrightarrow{\tau} P'}{c \blacktriangleleft [P\,;\ Q]^t_\kappa \xrightarrow{\tau} c \blacktriangleleft [P'\,;\ Q]^t_\kappa}\ (\textsc{TauL}) \qquad \frac{P \xrightarrow{c\ \mathtt{this}} P'}{c \blacktriangleleft [P\,;\ Q]^{t+1}_\kappa \xrightarrow{\tau} c \blacktriangleleft [P'\,;\ Q]^t_\kappa}\ (\textsc{ThisLocL})$$

$$\frac{P \xrightarrow{a\ \lambda\downarrow} P'}{c \blacktriangleleft [P\,;\ Q]^{t+1}_\kappa \xrightarrow{a\ \lambda\downarrow} c \blacktriangleleft [P'\,;\ Q]^t_\kappa}\ (\textsc{ThruL})$$

$$\frac{P \xrightarrow{\lambda\downarrow} P'}{c \blacktriangleleft [P\,;\ Q]^{t+1}_\kappa \xrightarrow{c\ \lambda\downarrow} c \blacktriangleleft [P'\,;\ Q]^t_\kappa}\ (\textsc{LocL}) \qquad \frac{P \xrightarrow{\lambda\uparrow} P'}{c \blacktriangleleft [P\,;\ Q]^{t+1}_\kappa \xrightarrow{\lambda\downarrow} c \blacktriangleleft [P'\,;\ Q]^t_\kappa}\ (\textsc{HereL})$$

$$\frac{}{\kappa^\dagger \xrightarrow{\kappa^\dagger} \mathbf{0}}\ (\textsc{Abort}) \qquad \frac{P \xrightarrow{\kappa^\dagger} P'}{P \mid c \blacktriangleleft [Q\,;\ R]^t_\kappa \xrightarrow{\tau} P' \mid c \blacktriangleleft [Q\,;\ R]^0_\kappa}\ (\textsc{FailPar1}) \qquad \frac{Q \xrightarrow{\lambda} Q'}{c \blacktriangleleft [P\,;\ Q]^0_\kappa \xrightarrow{\lambda} c \blacktriangleleft [P\,;\ Q']^0_\kappa}\ (\textsc{Comp})$$

$$\frac{P \xrightarrow{\kappa^\dagger} P' \quad \kappa \neq \gamma}{c \blacktriangleleft [P\,;\ Q]^t_\gamma \xrightarrow{\kappa^\dagger} c \blacktriangleleft [P'\,;\ Q]^t_\gamma}\ (\textsc{FailThruL}) \qquad \frac{P \xrightarrow{\kappa^\dagger} P'}{c \blacktriangleleft [P\,;\ Q]^t_\kappa \xrightarrow{\tau} c \blacktriangleleft [P'\,;\ Q]^0_\kappa}\ (\textsc{FailInt})$$

Figure 7.4: Rules for the LTS of C3.

We now describe some representative rules of the LTS of C3. Rule (Par1) decrees that executing an action in P decreases in one the time signatures of the conversation contexts in Q. This is the only rule that appeals to the time-elapsing function. For example, given an evolution $P \xrightarrow{\lambda} P'$, an environment $(\nu n)\ (c \blacktriangleleft [P\,;\ Q]^{t_1}_\kappa) \mid c \blacktriangleleft [R\,;\ S]^{t_2}_\chi$ evolves into $(\nu n)\ (c \blacktriangleleft [P'\,;\ Q]^{t_1 - 1}_\kappa) \mid \phi(c \blacktriangleleft [R\,;\ S]^{t_2}_\chi)$ using (LocL) (Res) and

(Par1). Rules formalizing communication and closing of scope extrusion do not affect the passage of time. In $e \blacktriangleleft [d \blacktriangleleft [(\nu n)\ (c \blacktriangleleft [P\,;\, Q]^{t_1}_{\kappa})\ |\ c \blacktriangleleft [R\,;\, S]^{t_2}_{\chi}\,;\, U]^{t_3}_{\psi}\,;\, V]^{t_4}_{\omega}$, timer t_4 will be updated after further applications of (ThruL) unless $c \blacktriangleleft [R\,;\, S]^{t_2}_{\chi} \xrightarrow{\bar{\lambda}}$, in which case rules (ThisClose1) and (ThisLocL) are applied and only timers t_1, t_2, t_3 are updated. This means that only the actions leading to a synchronization—but not the synchronization itself—contribute to the passage of time. Visible actions are privileged in the sense that they affect the time bound of the enclosing conversation context—compare rules (ThruL) and (TauL).

Rules (Abort), (FailPar1), and (Comp) formalize the essence of the handling of abortion signals and exceptional behavior in C3: the first formalize such signals, the second represents the abortion of a conversation context, and the third formalizes the behavior of an aborted conversation context. Intuitevely, compensation signals travel vertically upwards over levels of nested conversations: in $c \blacktriangleleft [d \blacktriangleleft [\kappa_c^{\dagger}\,;\, Q]^{t_d}_{\kappa_d}\,;\, R]^{t_c}_{\kappa_c}$, the compensation signal $\kappa_c^{\dagger}$ will cross its own conversation context and affect the outermost conversation following rule (FailThruL). Compensation signals can also affect conversations located in surrounding contexts: $c \blacktriangleleft [\kappa_d^{\dagger}\,;\, Q]^{t_c}_{\kappa_c}\ |\ d \blacktriangleleft [R\,;\, S]^{t_d}_{\kappa_d}$ will make conversation d entering into its compensation mode using (FailPar1), provided $\kappa_c \neq \kappa_d$. Finally, compensation signals become unobservable if they do lead to abortion (cf. rule (FailInt).

7.4 Expressiveness

Here we comment on the expressiveness of C3 by relating its constructs to those present in similar languages. Below, we consider the fragments of C3 without compensation signals and without time (noted C3^{-k} and C3^{-t}, respectively). While in C3^{-k} there are no explicit abortion signals but conversation contexts have time bounds, in C3^{-t} there are abortion signals, but all conversation contexts have ∞ as time bound. Considering these fragments of C3 can be helpful to illustrate its two main features. Our treatment is largely informal, as our objective is to shed light on the nature of C3. More formal comparisons of relative expressiveness are left for future work.

7.4.0.1 C3 and constructs for exception handling.

We consider the extension of the CC given in Section 7.2 with a try-catch operator. We first extend Definition 7.2.1 with a new action throw and a new process construct try P catch Q. Then, we extend the LTS of CC with rules TC1–TC5 in Figure 7.5. This is the semantics of exception handling considered in works such as, e.g., [Bravetti & Zavattaro 2009]. Let us refer to this extension as CC^{tc1}.

In C3^{-t} we can model a try-catch construct with such a semantics, as follows:

$$[\text{try } P \text{ catch } Q]^{tc} = t \blacktriangleleft [[P]^{tc}\,;\, [Q]^{tc}]^{\infty}_{\kappa_t} \qquad [\text{throw}]^{tc} = \kappa_t^{\dagger}$$

$$\frac{}{\text{throw}.\, P \xrightarrow{\text{throw}} P}\ (\text{TC}_1) \qquad \frac{P \xrightarrow{\text{throw}} R}{P \mid Q \xrightarrow{\text{throw}} R}\ (\text{TC}_2) \qquad \frac{P \xrightarrow{\text{throw}} R}{n \blacktriangleleft [P] \xrightarrow{\text{throw}} R}\ (\text{TC}_3)$$

$$\frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \text{throw}}{\text{try } P \text{ catch } Q \xrightarrow{\lambda} \text{try } P' \text{ catch } Q}\ (\text{TC}_4) \qquad \frac{P \xrightarrow{\text{throw}} R}{\text{try } P \text{ catch } Q \xrightarrow{\tau} Q}\ (\text{TC}_5)$$

$$\frac{P \xrightarrow{\text{throw}} R}{\text{try } P \text{ catch } Q \xrightarrow{\tau} Q \mid R}\ (\text{TC6})$$

Figure 7.5: LTS rules for an extension of the CC with try-catch

with $[\cdot]^{tc}$ being an homomorphism for the other operators, and κ_t and t being distinguished fresh names. The encoding captures the semantics of the try-catch operator thanks to the fact that a compensation signal can have effect on the conversation context enclosing it (cf. rule FailInt in Fig. 7.4).

A compositional encoding in the opposite direction, i.e., an encoding of C3^{-t} into CC^{tc1}, in which try-catch blocks are used to model conversation contexts, appears difficult. This is because of the nature of compensation in C3: extended conversation contexts can be aborted by both *internal* and *external* signals (cf. rules (FailInt) and (FailPar1) in Fig. 7.4), and not only by signals inside the try block. A possibility is to define an encoding that, using the number of conversation contexts and abortion signals in the process, creates a try-catch for every possible combination. This is a rather unsatisfactory solution, as interactions may give rise to new conversation contexts, and so the number of "global" try-catch constructs required for the encoding may not be predictable in advance. Based on these observations, we conjecture that C3^{-t} is strictly more expressive than CC^{tc1}.

Notice that an encoding such as the above would not work with a CC with try-catch with a different semantics. For instance, semantics enforcing advanced treatment of nested try blocks [Xu *et al.* 1998] may be difficult to handle. Let us consider CC^{tc2}, the extension of CC with the semantics given by rules TC_1-TC_4 and TC6 in Fig. 7.5. This is the semantics used in, e.g., [Caires *et al.* 2008]. The crucial difference between rules TC_5 and TC6 is that in the latter the state of the try-block just after the exception has been raised (i.e., R in both rules) is preserved when the exception block (i.e., Q) is called for execution, while in the former such a state is discarded. It is not obvious how to represent such a preservation of state in C3, as the standard part of the conversation context is completely discarded when the context is aborted, and there is no way of accessing it afterwards. We thus conjecture the non existence of an encoding of CC^{tc2} into C3^{-t}.

7.4.0.2 Time and Interruptions in C3.

From the point of view of exceptional behavior, time in C3 can be assimilated to an interruption mechanism over the standard part of a conversation context. We now argue that this character of timed behavior provides a gain in expressiveness from the basic CC to $C3^{-k}$.

Let $P = \mu X.\ \mathsf{l}_1!(n).X$ be the persistent process which is always ready to offer an output on l_1. When P is placed in a CC conversation context, we can infer an infinite sequence of transitions $c \blacktriangleleft [P] \xrightarrow{c\,\mathsf{l}_1!(n)} c \blacktriangleleft [P] \xrightarrow{c\,\mathsf{l}_1!(n)} \cdots$. Now consider the $C3^{-k}$ process $S = c \blacktriangleleft [P\,;\, \mathsf{l}_2!(n)]^t$, with P as before and some finite $t > 0$. Using the LTS of $C3^{-k}$, we infer that P evolves until $t = 0$, when it gets interrupted permanently and the execution control passes to the compensation part of the context. Hence, the observable behavior of S consists of t output signals on l_1, followed by a final output signal on l_2. This allows us to conjecture that an encoding of $C3^{-k}$ into CC, which preserves persistent behavior, does not exist.

It is useful to consider the situation for persistent behavior when CC is extended with try-catch constructs as the described above. Let us first consider CC^{tc1}. An encoding of $C3^{-k}$ into CC^{tc1} could exploit the fact that the semantics of try-catch allows to interrupt the behavior of a persistent process placed in the try block. However, it is not obvious at all where to place the throw prefixes so as to properly model the passage of time. That is, process interruption could be encoded but not at the *right time*: hence, it is not possible to guarantee that the behavior of the $C3^{-k}$ process is faithfully captured. Therefore, we conjecture that there is no encoding of $C3^{-k}$ into CC^{tc1}, up to some notion of operational correspondence sensible to timed behavior. Interestingly, when considering CC^{tc2}, the conjecture appears somewhat more certain since, as discussed before, the semantics of CC^{tc2} does preserve the last state of the try block before the exception is raised. That is, such a semantics does not implement interruption of the try block. This way, using P defined as above, process $S' = \mathsf{try}\, P \,||\, \mathsf{throw}.\mathbf{0}\, \mathsf{catch}\, \mathsf{l}_2!(n)$ would exhibit persistent behavior, even after the exception has been raised. Hence, outputs on l_1 and the output l_2 would be observable (interleaved) at the same time.

7.5 A Healthcare Compelling Example

Here we present a compelling example for C3: a medicine delivery scenario, adapted from [López *et al.* 2009, Campadello *et al.* 2006], which features time and exceptional behavior. After presenting the scenario, we present a series of models that gradually capture its main aspects. We begin with a basic model in CC, then we present two enhanced models: one capturing compensating behavior (using $C3^{-t}$) and another capturing time (using $C3^{-k}$). We then show how C3 allows a more comprehensive specification of the scenario.

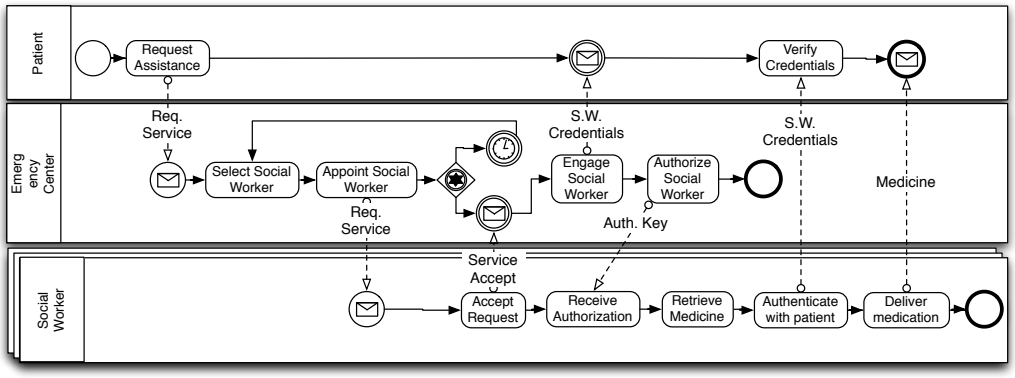


Figure 7.6: BPMN diagram for the medicine delivery scenario.

7.5.1 The Medicine Delivery Scenario.

We consider a simplified version of a medicine delivery scenario [López *et al.* 2009, Campadello *et al.* 2006]. Alice is a patient recently discharged from a hospital after a cardiac arrest. Sensors attached to the patient are monitoring her health conditions 24 hours a day, and data is controlled and processed in an Emergency Response Center (EC). The medicine delivery scenario takes place when Alice feels weak and, instead of driving to the pharmacy to get the medicine, asks to be supported by the EC. To this end, the EC requests a Social Worker (SW) to bring the medicine to Alice. There are both *mobile* SWs (in charge of requests outside the EC) and *in-house* SWs (the rest). If none of the mobile SWs can attend the request then an in-house SW is contacted. The selected SW gets appointed by the EC by sending him authorization keys for receiving the medicine and communicating with Alice. The SW can now acknowledge the request and go to the pharmacy. After a successful message exchange between Alice's terminal and the EC, the SW is authenticated and entitled to receive the medicine. Finally, the SW must authenticate to Alice in order to deliver the medicine. Figure 7.6 presents a specification of the above scenario in the Business Process Modeling Notation (BPMN 2.0), the de-facto notation for executable business processes.

7.5.1.1 Some Considerations: Time and Exceptions

Apart from the functional requirements exhibited on the previous case study, non-functional requirements about the *availability* of the system are very important in this scenario. For instance, it is necessary to ensure that in the exceptional circumstance in which there are no available social workers nearby the patient, there is always someone who can complete the process and provide the medication. These are local policies (in that they rule the execution of an agent in the system) but have effects at a global level (as they may refer to other agents apart from the ones currently

$$T \stackrel{\text{def}}{=} Patient \mid EC \mid SW \text{ where:}$$

$$Patient \stackrel{\text{def}}{=} A\blacktriangleleft[\textbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ \mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a]$$

$$EC \stackrel{\text{def}}{=} B\blacktriangleleft[\star\textbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \textbf{join}\, C \cdot BC \Leftarrow \mathrm{c}!(y).\ \mathrm{acc}?(v).\ (\nu pk)\ \mathrm{a}!(v,pk).\ \mathrm{c}!(pk).\ \checkmark b]$$

$$SW \stackrel{\text{def}}{=} C\blacktriangleleft[\textbf{def}\, BC \Rightarrow \mathrm{c}?(z).\ ((\nu v)\ \mathrm{acc}!(v).\ \mathrm{c}?(m_n).\ \mathrm{a2}!(n,m_n).\ \mathrm{a3}!(p)).\ \checkmark c]$$

Figure 7.7: Medicine Delivery Scenario: Basic Model in CC

executing in the current conversation). Timed aspects intrinsically related to the scenario are also inherently global. For instance, studies in [Rittenberger *et al.* 2006] show that patients with out-of-hospital cardiac arrests who are not provided with medicine within 17 minutes have higher chance of death. This kind of policies do not pertain the behavior of a single agent, but they involve global conditions applied to the whole interaction of roles involved in the process. That is, they are global policies which have specific consequences in the involved principals.

7.6 Timed and Compensating Models.

We proceed to incrementally define models for the medicine delivery scenario. We shall exploit variants of CC and C3 enhanced with special observable actions, or *tests*:

Definition 7.6.1. *Let $\mathcal{O}$ be a set of names, with $\mathcal{O} \cap \mathcal{N} = \emptyset$. The set of* testing processes *is obtained by extending* CC *and* C3 *as follows: the grammar of prefixes α is extended with a testing prefix $\checkmark o$, with $o \in \mathcal{O}$. Also, the associated LTS is extended with the rule* (Test) $\checkmark o.P \xrightarrow{o} P$. *Given a process P, we define $tn(P) = \{a \in \mathcal{O} \mid \checkmark a \text{ occurs in } P\}$*

We sometimes call $\mathcal{O}$ the set of testing names; observe that via the above definition we also obtain extensions for $\mathrm{C3}^{-t}$ and $\mathrm{C3}^{-k}$. Our first model is a CC specification giving only the basic interacting behavior in the medicine delivery scenario. We call this process T, and is presented in Figure 7.7. There are three conversation contexts, one for each agent involved in the business process: A (the patient), B (the EC), and C (the SW). Observe how we have added test prefixes at the end of each conversation context; this will be useful to observe the completion of the behavior of each agent. Interactions are meant to occur between A and B first, then between B and C, and finally between C and A. More precisely, A starts the protocol by invoking service AB, located at B. The body of $\textbf{new}\, B \cdot AB \Leftarrow \cdots$ first receives a request from A, and then extends the established conversation so as to include C, using the idiom $\textbf{join}\, C \cdot BC \Leftarrow \cdots$. Once A, B, and C share the conversation, they are able to interact between each other. This is evident in interactions on c and acc between B and C and interactions on a between A and B. Once the prefix $\checkmark b$ is emitted, A and C can interact on a2 and a3, and tests $\checkmark a, \checkmark c$ are observed.

$$
\begin{aligned}
S^F \stackrel{\text{def}}{=}\ & Patient \mid EC \mid SW \mid Nurse \text{ where:}\\
Patient \stackrel{\text{def}}{=}\ & A\blacktriangleleft[\mathbf{new}\, B \cdot AB \Leftarrow \\
& \qquad \mathrm{b}!(l).\ (\mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a + \kappa_A^{\dagger})\,; \\
& \quad \mathbf{new}\, B \cdot AB \Leftarrow \\
& \qquad \mathrm{b}!(l).\ (\mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a]_{\kappa_A}^{\infty} \\
EC \stackrel{\text{def}}{=}\ & B\blacktriangleleft[\star\mathbf{def}\, AB \Rightarrow\ \mathrm{b}?(y).\ \mathbf{join}\, C \cdot BC \Leftarrow\ \mathrm{c}!(y).\ (\mathrm{dny}?(v).\ \kappa_B^{\dagger} \\
& \qquad\qquad + \\
& \qquad\qquad \mathrm{acc}?(v).\ (\nu pk)\ \mathrm{a}!(v,pk).\ \mathrm{c}!(pk).\ \checkmark b)\,; \\
& \kappa_A^{\dagger} \mid \\
& \star\,\mathbf{def}\, AB \Rightarrow\ \mathrm{b}?(y). \\
& \qquad \mathbf{join}\, D \cdot BD \Leftarrow \\
& \qquad\quad \mathrm{d}!(y).\ \mathrm{acc}?(v).\ (\nu pk)\ \mathrm{a}!(v,pk).\ \mathrm{c}!(pk).\ \checkmark b]_{\kappa_B}^{\infty} \\
SW \stackrel{\text{def}}{=}\ & C\blacktriangleleft[\mathbf{def}\, BC \Rightarrow \\
& \qquad \mathrm{c}?(z).\ ((\nu v)\ (\mathrm{dny}!(v) \\
& \qquad\qquad + \\
& \qquad\qquad \mathrm{acc}!(v)).\ \mathrm{c}?(m_n).\ \mathrm{a2}!(n,m_n).\ \mathrm{a3}!(p)).\ \checkmark c\,;\, \mathbf{0}]_{\kappa_C}^{\infty} \\
Nurse \stackrel{\text{def}}{=}\ & D\blacktriangleleft[\mathbf{def}\, BD \Rightarrow\ \mathrm{d}?(z).\ ((\nu v)\ \mathrm{acc}!(v).\ \mathrm{d}?(m).\ \mathrm{a2}!(k,m).\ \mathrm{a3}!(p)).\ \checkmark c\,;\, \mathbf{0}]_{\kappa_D}^{\infty}
\end{aligned}
$$

Figure 7.8: Medicine Delivery Scenario: Exception-only model in $\mathrm{C3}^{-t}$

7.6.1 Exceptional Behavior.

We now consider two different variants of model T. The first one, denoted S^F, extends T with compensation activities only. The second variant, denoted S^T, extends T with explicit timed behavior and is described later on. The model for S^F is given in $\mathrm{C3}^{-t}$, and is presented in Figure 7.8. Essentially, S^F extends T by giving a patient the capability of contacting twice the EC, either at patients' discretion or fired by the sensors attached to him. Moreover, the SW can either accept or ignore the request; in the former case the EC will compensate by contacting its in-house SW.

The model of the medicine deliver scenario in $\mathrm{C3}^{-t}$ differs from that in CC in three ways. First, it allows the SW to refuse the engagement in the interaction; this is modeled as a non-deterministic output on dny. Second, it includes the specification of a nurse that will take care of delivering of the medicine in case the SW cannot be engaged in the interaction. Third, it allows for all conversation contexts to have compensating activities. In case of the patient, it abstracts the fact that he can call for attention more than once in case the service is not provided. As for the EC, it can use compensating activities to restart the requests refused by the SW and appoint a nurse instead. Here, we model explicit compensations (e.g., the emission of $\kappa_B^{\dagger}$ in EC) and chains of compensations (i.e., the compensating part of A follows after the

$$S^T \stackrel{\text{def}}{=} Patient \mid EC \mid SW \mid Nurse \text{ where:}$$

$$Patient \stackrel{\text{def}}{=} A\blacktriangleleft[\mathbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ \mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a\,;\ \mathbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ \mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a]^{t_A}$$

$$EC \stackrel{\text{def}}{=} B\blacktriangleleft[\star\mathbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \mathbf{join}\, C \cdot BC \Leftarrow \mathrm{c}!(y).\ (\mathbf{0} + \mathrm{acc}?(v)(\nu pk)\ \mathrm{a}!(v,pk).\ \mathrm{c}!(pk).\ \checkmark b)\,;\ \star\mathbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \mathbf{join}\, D \cdot BD \Leftarrow \mathrm{d}!(y).\ \mathrm{acc}?(v)(\nu pk)\ \mathrm{a}!(v,pk).\ \mathrm{c}!(pk).\ \checkmark b]^{t_B}$$

$$SW \stackrel{\text{def}}{=} C\blacktriangleleft[\mathbf{def}\, BC \Rightarrow \mathrm{c}?(z).\ ((\nu v)\ \mathrm{acc}!(v).\ \mathrm{c}?(m_n).\ \mathrm{a2}!(n,m_n).\ \mathrm{a3}!(p)).\ \checkmark c\,;\ \mathbf{0}]^{t_C}$$

$$Nurse \stackrel{\text{def}}{=} D\blacktriangleleft[\mathbf{def}\, BD \Rightarrow \mathrm{d}?(z).\ ((\nu v)\ \mathrm{acc}!(v).\ \mathrm{d}?(m).\ \mathrm{a2}!(k,m).\ \mathrm{a3}!(p)).\ \checkmark c\,;\ \mathbf{0}]^{t_D}$$

Figure 7.9: Medicine Delivery Scenario: Model in $\mathrm{C3}^{-k}$

compensating part of B has been activated). A model such as S^F improves T in that it is able to express exceptional behavior that is represented explicitly, but falls short to express more implicit forms of exceptional behavior that have to deal with time bounds. Despite its enhanced expressiveness for process compensations, a model in $\mathrm{C3}^{-t}$ is yet not able to deal with constraints involving the evolutions of the whole system, as required in these kind of scenarios. In fact, although there is not a notion of "global time" in this model, the (local) evolution of time in a process does affect the time signatures of surrounding processes (i.e., located within and in parallel to it). Hence, one may argue that via time signatures we are able to express a general notion of evolution that goes beyond local change.

7.6.2 A timed model.

We consider now a model where compensations are fired by implicit time constraints only. Process S^T in Figure 7.9 formalizes in $\mathrm{C3}^{-k}$ the case where A needs to get the medicine within t_A time units. S^T is an extension of T; the main improvements are the compensation parts in conversation contexts, which are associated to time bounds. Reaching the time bound in t_A means that the manual request for attention fired by the patient has expired, so the exceptional behavior will automatically restart the request. Similarly for the EC: once time bound t_B is reached, it will migrate the request for service from the SW (cf. conversation context C) to the Nurse (cf. conversation context D). This is in line with the treatment of timed aspects as outlined by clinical guidelines, such as the described in [Terenziani *et al.* 2000].

As opposed to S^F, in S^T we lack explicit signals to trigger a compensation activity. This entails some limitations in our model. For instance, in the case that the mobile SW responds negatively to a request from a EC, then the EC should be able to *immediately* enable a compensation activity and send its on-site SW. In general,

triggering exceptional behaviors through time passing only is far from ideal in the case of compensation activities which need to be triggered as soon as some certain event occurs.

7.6.3 Putting all together.

Exploiting the best features of S^F and S^T, a model of the medicine delivery scenario in C3 is presented in Figure 7.10. Process S extends T with differences in the interaction between conversations. In particular, the body of the service AB located at B contains calls to DB, a database containing the contacts of available SWs.[1] The direction of the messages between DB and the service is $\uparrow$, as communications have to cross the boundary defined by the service definition. AB describes the process that first selects one of the SWs and then forwards the request for attention started by the patient. Process C_n represents the behavior of the n-th instance of a mobile SW. As in S^F and S^T, in S the SW can either accept or ignore the request; in the latter case the process iterates until some SW is appointed. Upon acceptance, B will generate new credentials identifying the SW (represented in the model with a fresh name m); in turn, these will be transmitted to the patient for further checking.

Two important exceptional behaviors can be observed in this example. The first one concerns the prompt response required by the patient. In case the request for attention is not delivered on due time and the patient starts feeling dizzy, sensors attached to patient's body can detect a decrease in his health conditions (say, blood pressure) and restart automatically the medicine delivery process in order to ensure the request call is answered. The behavior of the sensors here is abstracted by a non-deterministic choice. This behavior is present in the compensated part of A, and it will be fired either when the expected time t_A has been exhausted, or when the EC reports unavailability (represented by signal $\kappa_A^{\dagger}$). The second timeout refers to internal process requirements from B, which stipulate that each request for attention has to be attended within t_B time units. This is irrespective from any further communication done elsewhere. The EC is fault tolerant, and on unavailability of a mobile SW, it will rely on the nurse.

7.6.4 The Semantics At Work.

Let us illustrate the LTS of C3 by revisiting the health care scenario discussed in the introduction. We describe the evolution of a modified S where the EC provides fault-handling with respect to its local database. This example is useful to appreciate the way time in C3 is defined. On failure to find available mobile SWs, the system will emit a signal $\mathrm{NA}^{\uparrow}!()$ that later will be used to spawn the compensation mechanisms of the EC and further contact its in-house SW. The modified system is as in Fig. 7.10

[1] We could have well invoked a database both in S^F and S^T, but we refrained to do so in order to keep those models simple.

$S \stackrel{\texttt{def}}{=} Patient \mid EC \mid SW_n \mid Nurse$ where

$Patient \stackrel{\texttt{def}}{=} A\blacktriangleleft[\mathbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ (\mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a + \kappa_A^{\dagger})\,;$
$\mathbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ \mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p).\ \checkmark a]_{\kappa_A}^{t_A}$

$SW_n \stackrel{\texttt{def}}{=} C_n\blacktriangleleft[\mathbf{def}\, BC \Rightarrow \mathrm{c}?(z).\ ((\nu v)\ (\mathrm{dny}!(v) + \mathrm{acc}!(v)).\ \mathrm{c}?(m_n).\ \mathrm{a2}!(n,m_n).\ \mathrm{a3}!(p)).\ \checkmark c\,;\ \mathbf{0}]_{\kappa_C}^{t_C}$

$Nurse \stackrel{\texttt{def}}{=} D\blacktriangleleft[\mathbf{def}\, BD \Rightarrow \mathrm{d}?(z).\ ((\nu v)\ \mathrm{acc}!(v).\ \mathrm{d}?(m).\ \mathrm{a2}!(k,m).\ \mathrm{a3}!(p)).\ \checkmark c\,;\ \mathbf{0}]_{\kappa_D}^{\infty}$

$EC \stackrel{\texttt{def}}{=} B\blacktriangleleft[DB \mid \star\, \mathbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \mu X.\ \mathrm{req}^{\uparrow}!(y).\ \mathrm{rep}^{\uparrow}?(n).$
$\mathbf{join}\, C_n \cdot BC \Leftarrow \mathrm{c}!(y).\ (\mathrm{dny}?(v).\ X + \mathrm{acc}?(v).\ (\nu m)\ \mathrm{a}!(v,m).\ \mathrm{c}!(m))\,;$
$\kappa_A^{\dagger} \mid DB \mid \star\, \mathbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \mathbf{join}\, D \cdot BD \Leftarrow \mathrm{d}!(y).\ \mathrm{acc}?(v).\ (\nu m)\ \mathrm{a}!(v,m).\ \mathrm{d}!(m)]_{\kappa_B}^{t_B}$

Figure 7.10: The medicine delivery scenario in C3

with the following definitions for A and B:

$$\begin{aligned} A\blacktriangleleft[&\mathbf{new}\, B \cdot AB \Leftarrow \mathrm{b}!(l).\ \mathrm{a}?(z,m).\ \mathrm{a2}?(z,m).\ \mathrm{a3}?(p)\,;\ CA]_{\kappa_A}^{t} \\ B\blacktriangleleft[&DB \mid \mathbf{def}\, AB \Rightarrow \mathrm{b}?(y).\ \mu X.\ \mathrm{req}^{\uparrow}!(y).\ (\mathrm{NA}^{\uparrow}?().\ \kappa_B^{\dagger} + \mathrm{rep}^{\uparrow}?(n). \\ &\mathbf{join}\, C_n \cdot BC \Leftarrow \mathrm{c}!(y).\ (\mathrm{dny}?(v).\ X + \mathrm{acc}?(v). \\ &(\nu m)\ \mathrm{a}!(v,m).\ \mathrm{c}!(m)))\,;\ \kappa_A^{\dagger}]_{\kappa_B}^{t_B} \end{aligned}$$

where in A, CA corresponds to the compensating behavior of A in Fig. 7.10. By expanding the definition of **def** and **new**, we have:

$$\begin{aligned} &A\blacktriangleleft[(\nu c)\ (B\blacktriangleleft[\mathrm{AB}!(c)\,;\ \mathbf{0}]_{\emptyset}^{\infty} \mid c\blacktriangleleft[P_A\,;\ \mathbf{0}]_{\emptyset}^{\infty})\,;\ CA]_{\kappa_A}^{t} \\ &\mid B\blacktriangleleft[DB \mid \mathrm{AB}?(x).\ x\blacktriangleleft[P_B\,;\ \mathbf{0}]_{\emptyset}^{\infty}\,;\ \kappa_A^{\dagger}]_{\kappa_B}^{t_B} \mid C_n \mid D \end{aligned}$$

where P_A, P_B are abbreviations of the behaviors at patient and EC sides, respectively. Let us focus on the interaction betwen A and B. First, we infer the following output transition from A, using rules (Out), (LocL), (Open), (Par1) and (ThruL):

$$\begin{aligned} A\blacktriangleleft[(\nu c)\ (B\blacktriangleleft[\mathrm{AB}!(\downarrow)c\,;\ \mathbf{0}]_{\emptyset}^{\infty} \mid c\blacktriangleleft[P_A\,;\ \mathbf{0}]_{\emptyset}^{\infty})\,;\ CA]_{\kappa_A}^{t} &\xrightarrow{(\nu c)\ B\ \mathrm{AB}!(c)} \\ A\blacktriangleleft[(\nu c)\ (B\blacktriangleleft[\mathbf{0}\,;\ \mathbf{0}]_{\emptyset}^{\infty} \mid c\blacktriangleleft[\phi(P_A)\,;\ \mathbf{0}]_{\emptyset}^{\infty})\,;\ CA]_{\kappa_A}^{t-1} & \end{aligned}$$

We can also infer an input transition from B, using rules (In), (Par2), and (LocL):

$$B\blacktriangleleft[DB \mid \mathrm{AB}?(x).\ x\blacktriangleleft[P_B\,;\ \mathbf{0}]_{\emptyset}^{\infty}\,;\ \kappa_P^{\dagger}]_{\kappa_B}^{t_B} \xrightarrow{B\ \mathrm{AB}?(c)} B\blacktriangleleft[\phi(DB) \mid c\blacktriangleleft[P_B\,;\ \mathbf{0}]_{\emptyset}^{\infty}\,;\ \kappa_P^{\dagger}]_{\kappa_B}^{t_B-1}$$

Given these transitions, a synchronization can be inferred using rule (Close1), taking c as shared name for A and B to communicate. When considering the system as a

whole, this synchronization will affect the time signatures in agents C_n and D; this is formalizes using rule (Par1). We then have:

$$(\nu c)\,(A \blacktriangleleft [B \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [\phi(P_A)\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, CA]^{t-1}_{\kappa_A} \mid B \blacktriangleleft [\phi(DB) \mid c \blacktriangleleft [P_B\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, \kappa_P^{\dagger}]^{t_B-1}_{\kappa_B}) \mid \phi(C_n) \mid \phi(D)$$

At this point, the default behavior of the system allows B to communicate locally with DB and ask for a SW. After receiving $\mathrm{NA}^{\uparrow}!(c)$ from DB, the system evolves into:

$$(\nu c)\,(A \blacktriangleleft [B \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [\phi^3(P_A)\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, CA]^{t-3}_{\kappa_A} \mid B \blacktriangleleft [\phi^3(DB) \mid c \blacktriangleleft [\kappa_B^{\dagger}\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, \kappa_P^{\dagger}]^{t_B-3}_{\kappa_B}) \mid \phi^3(C_n) \mid \phi^3(D) = S'$$

Here the abortion mechanisms come into play: by the emision of $\kappa_B^{\dagger}$ it is possible to switch to the compensating part of the conversation despite there is still some time before reaching a timeout. Compensation signals will travel upwards among conversations. Applying (Abort), (FailThruL), (FailInt) and (Par1), we obtain:

$$S' \xrightarrow{\tau} (\nu c)\,(A \blacktriangleleft [B \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [\phi^4(P_A)\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, CA]^{t-4}_{\kappa_A} \mid B \blacktriangleleft [\phi^3(DB) \mid c \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, \kappa_P^{\dagger}]^{0}_{\kappa_B}) \mid \phi^4(C_n) \mid \phi^4(D) = S''$$

Finally, compensating signals can travel across conversation contexts in parallel. After the execution of (Abort), (comp), (FailPar1) and (Par1), below we can see how the exceptional behavior of B makes A search for an alternative solution (i.e.: restart the request and increase the emergency level):

$$S'' \xrightarrow{\tau} (\nu c)\,(A \blacktriangleleft [B \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [\phi^4(P_A)\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, CA]^{0}_{\kappa_A} \mid B \blacktriangleleft [\phi^4(DB) \mid c \blacktriangleleft [\mathbf{0}\,;\,\mathbf{0}]^{\infty}_{\emptyset}\,;\, \mathbf{0}]^{0}_{\kappa_B}) \mid \phi^5(C_n) \mid \phi^5(D)$$

7.6.5 Refining the Initial Model.

We now explore a very basic notion of simulation for comparing models featuring timed and exceptional behavior. Roughly speaking, we say that a model (e.g., an implementation) *refines* another model (e.g., a specification) if it passes the same tests. One would expect the implementation to contain more behavior (for instance, compensation activities and timeouts) than the specification, therefore we aim for refinement to be a preorder (i.e.: a reflexive, transitive but not symmetric) relation. The following definitions are specialized for our refinement setting, coming from classical notions of simulation [Milner 1999] and testing [De Nicola & Hennessy 1984]:

Definition 7.6.2 (Refinement). *Let $\mathcal{O}$ be a non-empty set of testing names. A binary relation over testing processes $\mathcal{R}$ is a* refinement up to $\mathcal{O}$ *if $P\,\mathcal{R}\,Q$ implies:*
(1) if $P \longrightarrow P'$ then $\exists Q'.Q \longrightarrow^* Q'$ and $P'\,\mathcal{R}\,Q'$ and
(2) $\forall o \in \mathcal{O}$, $P \xrightarrow{o} P'$ implies $\exists Q'.Q \stackrel{o}{\Rightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$.
We say that Q refines *P* up to *$\mathcal{O}$ (written $P \sqsubseteq_{\mathcal{O}} Q$) if there exists a refinement up to $\mathcal{O}$ $\mathcal{R}$ such that $P\mathcal{R}Q$.*

Proposition 7.6.3 (Refinement is a preorder). *Let $\mathcal{O}$ be a fixed set of testing names. Then $\sqsubseteq_{\mathcal{O}}$ is a preorder on processes (i.e., a reflexive and transitive order relation).*

Proof. It is immediate to see that relation $\{(P, P) \mid P \text{ is a testing process}\}$ is a refinement up to $\mathcal{O}$, thus $\sqsubseteq$ is reflexive. For transitivity, we have to show that if $\mathcal{R}_1, \mathcal{R}_2$ are refinements up to $\mathcal{O}$, then their composition, defined as $\mathcal{R}_1\mathcal{R}_2 = \{(P, S) \mid \exists Q, P\mathcal{R}_1 Q \wedge Q\mathcal{R}_2 S\}$, is also a refinement up to $\mathcal{O}$. We thus verify the two conditions in Definition 7.6.2.

For the first condition, let $(P, S) \in \mathcal{R}_1\mathcal{R}_2$ and suppose $P \longrightarrow P'$. We have to exhibit a process S' such that $S \longrightarrow^* S'$ and $P'\mathcal{R}_1\mathcal{R}_2 S'$. By definition of $\mathcal{R}_1\mathcal{R}_2$ there exists a Q s.t. $P\mathcal{R}_1 Q$ and $Q\mathcal{R}_2 S$. Hence, if $P \longrightarrow P'$ then there exists a Q' s.t. $Q \longrightarrow^* Q'$ and $P'\mathcal{R}_1 Q'$. In turn, this ensures the existence of the desired S', as $S \longrightarrow^* S'$ and $Q'\mathcal{R}_2 S'$. Hence, $P'\mathcal{R}_1\mathcal{R}_2 S'$.

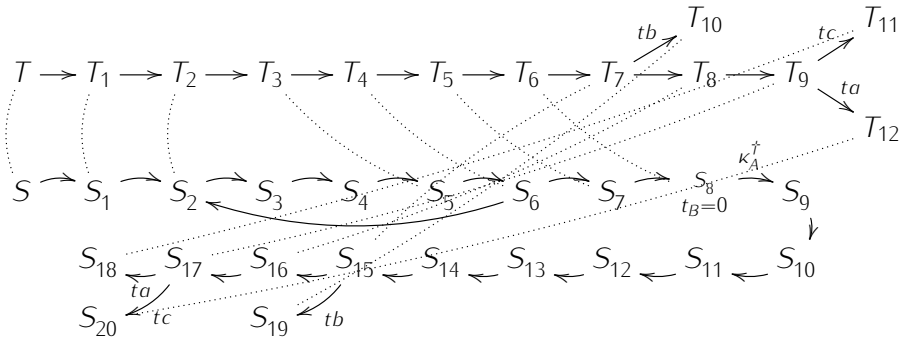
The reasoning for the second condition is similar and holds because both $\mathcal{R}_1$, $\mathcal{R}_2$ are refinements up to the same set of testing names, $\mathcal{O}$. Suppose $P \xrightarrow{o} P'$. We have to exhibit a process S' such that $S \stackrel{o}{\Rightarrow} S'$ and $P'\mathcal{R}_1\mathcal{R}_2 S'$. By definition of $\mathcal{R}_1\mathcal{R}_2$ there exists a Q s.t. $P\mathcal{R}_1 Q$ and $Q\mathcal{R}_2 S$. Hence, if $P \xrightarrow{o} P'$ then there exists a Q' s.t. $Q \stackrel{o}{\Rightarrow} Q'$ and $P'\mathcal{R}_1 Q'$. Since $Q\mathcal{R}_2 S$, S can match all these actions, and the existence of an S' such that $S \stackrel{o}{\Rightarrow} S'$ and $Q'\mathcal{R}_2 S'$ is ensured. Hence, $P'\mathcal{R}_1\mathcal{R}_2 S'$. □

We are now ready to establish the relation between the models introduced before.

Proposition 7.6.4. *Let T, S^F, S^T, S be the processes defined above. Then we have: $T \sqsubseteq S^F$, $T \sqsubseteq S^T$, and $T \sqsubseteq S$, $S^T \not\sqsubseteq S^F$ and $S^F \not\sqsubseteq S^T$.*

Sketch. Following Def. 7.6.2, it is sufficient to show that there exist relations $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ which are simulation relations under $tn(T)$ such that $T\,\mathcal{R}_1\,S^F$, $T\,\mathcal{R}_2\,S^T$, and $T\,\mathcal{R}_3\,S$. The last two cases can be proven by counterexamples.

We discuss the case of $\mathcal{R}_3$ below. Consider the relation of pairs of process states between T and S[2], so $\mathcal{R}_3 = \{(T, S), (T_1, S_1), (T_2, S_2), (T_3, S_5), (T_4, S_6), (T_5, S_7), (T_6, S_8), (T_7, S_{15}), (T_8, S_{16}), (T_9, S_{17}), (T_{10}, S_{19}), (T_{11}, S_{18}), (T_{12}, S_{20})\}$ denoted by the dotted lines in the following diagram:



[2]Note that there is a set of possible process transitions for each instantiation of the time signature t in a conversation $n \blacktriangleleft [P\,;\,Q]^t_\kappa$.

Above S_i, T_i represent the i-th transitions of processes S and T, arrows without labels represent unobservable transitions from synchronizations between different conversations, and ti denotes the execution of a test for process i. Similarly, $\kappa^{\dagger}$ represent the firing of a compensation signal κ. Intuitively, the diagram above describes how the simple tests and the complete specification are related to each other: They behave equally up-to pair (T_2, S_2), where S include extra evolutions describing its consultation on the social-workers database, meeting again in pair (T_3, S_5). The simulation goes onwards until having reached the communication of social worker credentials between B and A (pairs (T_6, S_8)), but timeout t_B is reached and compensation processes for A and B are fired. When reaching pair (T_7, S_{15}) we get back to a consistent state, and S can perform the tests given in T. □ □

7.7 Related Work

Although there is a long history of timed extensions for (mobile) process calculi (see, e.g., [Berger & Honda 2000]) and the study of constructs for exceptional behavior has received significant attention (see [Ferreira *et al.* 2010] for a recent overview), time and its interplay with forms of exceptional behavior do not seem to have been jointly studied in the context of models for structured communication. In our previous work [López *et al.* 2010] we have studied an LTL interpretation of the session language in [Honda *et al.* 1998] and proposed a extension of it with time, declarative information, and a construct for session abortion. The language in [Honda *et al.* 1998] is however limited for our purposes, as it does not support multiparty interactions. The differences in expressiveness between C3 and a previous variant of the CC featuring try-catch constructs [Vieira *et al.* 2008] have been already discussed in Section 7.4.

In the past, time and exceptional behavior have been considered only separately in orchestrations and choreographies. With respect to time, Timed Orc [Wehrman *et al.* 2008] introduced real-time observations for orchestrations by introducing a delay operator. Timed COWS [Lapadula *et al.* 2007b] extends COWS (the Calculus for Orchestration of Web Services [Lapadula *et al.* 2007a]) with operators of delimitation, abortion, and delays for orchestrations; we are not aware of reasoning techniques for Timed COWS. With respect to exceptional behavior, [Carbone *et al.* 2008, Capecchi *et al.* 2010] propose languages for *interactional exceptions*, in which exceptions in a protocol generate coordinated actions between all peers involved. Associated type systems ensure communication safety and termination among protocols with normal and compensating executions. In [Capecchi *et al.* 2010], the language is enriched further with multiparty session and global escape primitives, allowing nested exceptions to occur at any point in an orchestration. As for choreographies, [Carbone 2008] introduced an extension of a language of choreographies with try/catch blocks, guaranteeing that embedded compensating parts in a choreography are not arbitrarily killed as a result of an abortion signal.

On a similar track, the work in [Hongli *et al.* 2007] presents a denotational semantics based on traces for a simple language for choreographies with exception

handling and finalization constructs, allowing a projection from exceptional behavior of a choreography to its endpoints. The main differences from our approach are the language constructs and the semantics for compensable behavior used: First, the language for choreographies used assume that there is a principal taking the decisions about which branches to execute, whereas the semantics of choice in C3 assume a fully distributed system where any choice is equally possible. Second, the semantics of the compensating blocks act pretty much like exceptions in sequential languages, where exceptions are evaluations of expressions, and there is no treatment for nesting contexts.

Our work has been influenced by extensions to the (asynchronous) π-calculus, notably [Laneve & Zavattaro 2005, Berger & Honda 2000]. In particular, the role of the time-elapsing behavior for conversation contexts used in C3 draws inspiration from the behavior of long transactions in webπ, and from the π-calculus with timers in [Berger & Honda 2000]. Notice that the nature of these languages and C3 is very different. First, the communication model is different: C3 is synchronous, while the calculi in [Laneve & Zavattaro 2005, Berger & Honda 2000] are asynchronous. Second, webπ is a language tailored to study long-running transactions, and therefore exceptions in webπ and compensations in C3 have a completely different meaning, even if both constructs look similar.

7.8 Concluding Remarks

We have presented C3, a variant of the CC in which conversation contexts have an explicit duration, a compensation activity, and can be explicitly aborted. We have informally discussed the expressiveness and relevance of its two main features: explicit abortion signals and timed behavior. We have illustrated these features in a heathcare scenario of structured communications.

Ongoing Work. There are a number of directions which are worth pursuing based on the developments presented here. The most pressing issue concerns analysis techniques for C3 specifications. We would like to develop type disciplines for ensuring communication correctness in models featuring time and exceptional behavior. For this purpose, conversation types [Caires & Vieira 2010] and the linear/affine type system proposed in [Berger & Yoshida 2007] might provide a reasonable starting point. We are also interested in a notion of *refinement* between a model in CC and an associated model in C3. In this chapter we report some preliminary ideas in this direction: intuitively, the objective is to decree that a C3 model is a refinement of a related CC model if they pass a set of *tests* present in both models; the challenge is to obtain suitable characterizations for such tests, considering time and the execution of compensating behavior.

A different research direction concerns obtaining formal separation results for the expressiveness conjectures stated in Section 7.4. Different models for exception handling induce different semantics for treating exceptional behavior; it would be

interesting to understand their precise relation in terms of expressiveness. While some previous work has addressed similar questions [Lanese *et al.* 2010], we think it would be interesting to study such question in the context of concrete models for structured communications, such as CC and C3.

Finally, we are interested in equipping C3 with behavioral equivalences, and associated properties in the lines of that proposed for the CC. In particular, we would like to obtain a set *behavioral equations* (such as those defined for the CC in [Vieira *et al.* 2008, Prop. 4.3]) as a reasoning technique for C3. Such behavioural equivalences have to take into account the different nature of compensations here presented, and they are related to the work in communicating transactions [de Vries *et al.* 2010], with the difference that C3 does not assume any coordination checkpoint.

Acknowledgements. This research has been supported by the Danish Research Agency through the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, `www.TrustCare.eu`) and by the Portuguese Foundation for Science and Technology (FCT/MCTES) through the Carnegie Mellon Portugal Program, grant INTERFACES NGN-44 / 2009. We thank the anonymous reviewers for their useful comments.

Appendix 7.A Further Examples: Running the Buyer-Seller example

Let us illustrate the LTS of C3 by revisiting the extended purchase scenario discussed in the introduction. We describe the evolution of the system where the buyer invokes $Seller_1$ with an expected response time of two time units. We recall the definition of the complete system:

$$
\begin{array}{l}
NewBuyer \blacktriangleleft [\prod_{i\in\{1,2,3\}} \textbf{new}\ Seller_i \cdot \mathrm{BuyService}\,\textbf{with}\,(e_i, v_i) \Leftarrow \{P_i; Q_i\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid Control\,;\, CancelOrder]_x^{tmax} \\
\mid \prod_{i\in\{1,2,3\}} Seller_i \blacktriangleleft [DB \mid \textbf{def}\ \mathrm{BuyService}\,\textbf{with}\,(b_i, w_i) \Rightarrow \{\mathrm{offer}?(prod). \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{askPrice}^{\uparrow}!(prod).\ \mathrm{priceVal}^{\uparrow}?(p).\ \mathrm{price}!(p). \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{join}\ Shipper \cdot \mathrm{DeliveryService} \Leftarrow \mathrm{product}^{\uparrow}!(prod); \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad R_i\}\,;\, CancelSell_i]_{x_i}^{t_i} \\
\mid Shipper \blacktriangleleft [\textbf{def}\ DeliveryService\,\textbf{with}\,(d, t) \Rightarrow \{\mathrm{product}?(p).\ \mathrm{details}!(data); T\}\,;\, \mathbf{0}]_z^{t_3}
\end{array}
$$

where $P_i \stackrel{\textbf{def}}{=} \mathrm{offer}!(prod).\ \mathrm{price}?(p).\ \mathrm{com}_i{}^{\uparrow}!(p).\ \mathrm{details}?(d)$. By expanding the definition of **def** and **new**, we have:

$$
\begin{array}{l}
NewBuyer \blacktriangleleft [(\nu c)\ (Seller_1 \blacktriangleleft [\mathrm{BuyService}!(c)\,;\, \mathbf{0}]_{\emptyset}^{\infty} \mid c \blacktriangleleft [P_1\,;\, Q_1]_{v_1}^{2}) \mid S_b \mid Control\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad CancelOrder]_x^{tmax} \\
\mid Seller_1 \blacktriangleleft [DB \mid \mathrm{BuyService}?(y).\ y \blacktriangleleft [\mathrm{offer}?(prod).(\cdots)\,;\, R_1]_{w_1}^{b_1}\,;\, CancelSell_1]_{x_1}^{t_1} \\
\mid S_c \mid Shipper
\end{array}
$$

where S_b abbreviates the definitions of $Seller_2$ and $Seller_3$ at the buyer side, and S_c is the analogous process at the shippers side. Focusing on $NewBuyer$, we can

infer the following transition, using rules (Out), (Res), (LocL), and (Par1):

$$\begin{array}{l} \mathit{NewBuyer} \blacktriangleleft [(\nu c)\ (\mathit{Seller}_1 \blacktriangleleft [\mathsf{BuyService!}(c)\,;\ \mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [P_1\,;\ Q_1]^{2}_{v_1}) \mid S_b\,; \\ \qquad\qquad \mathit{CancelOrder}]^{t_{max}}_{x} \\ \qquad\qquad \xrightarrow{(\nu c)\ \mathit{Seller}_1\ \mathsf{BuyService!}(c)} \mathit{NewBuyer} \blacktriangleleft [\mathit{Seller}_1 \blacktriangleleft [\mathbf{0}\,;\ \mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [P_1\,;\ Q_1]^{1}_{v_1} \mid \phi(S_b)\,; \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{CancelOrder}]^{t_{max}}_{x} \end{array}$$

which decreases the time bound for conversation context c to 1. The behavior of Seller_1 is completely complementary to the above output, as inferred by using (In), (LocL), and (Par1):

$$\begin{array}{l} \mathit{Seller}_1 \blacktriangleleft [\mathit{DB} \mid \mathsf{BuyService?}(y).\ y \blacktriangleleft [\mathsf{offer!}(\mathit{prod}).(\cdots)\,;\ R_1]^{b_1}_{w_1}\,; \\ \qquad\qquad \mathit{CancelSell}_1]^{t_1}_{x_1} \\ \qquad\qquad \xrightarrow{\mathit{Seller}_1\ \mathsf{BuyService?}(c)} \mathit{Seller}_i \blacktriangleleft [\mathit{DB} \mid c \blacktriangleleft [\mathsf{offer?}(\mathit{prod}).(\cdots)\,;\ R_1]^{b_1}_{w_1}\,; \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{CancelSell}_1]^{t_1-1}_{x_1} \end{array}$$

Given these two transitions, a synchronization can be inferred using rules (CloseL) and (Par1), using c as shared name for $\mathit{NewBuyer}$ and Seller_1 to communicate. The state of the system is then:

$$\begin{array}{l} (\nu c)\ (\mathit{NewBuyer} \blacktriangleleft [\mathit{Seller}_1 \blacktriangleleft [\mathbf{0}\,;\ \mathbf{0}]^{\infty}_{\emptyset} \mid c \blacktriangleleft [\mathsf{offer!}(\mathit{prod}).\ (\cdots)\,;\ Q_1]^{1}_{v_1} \mid \phi(S_b)\,; \\ \qquad\qquad \mathit{CancelOrder}]^{t_{max}}_{x} \\ \mid \mathit{Seller}_1 \blacktriangleleft [\mathit{DB} \mid c \blacktriangleleft [\mathsf{offer?}(\mathit{prod}).(\cdots)\,;\ R_1]^{b_1}_{w_1}\,;\ \mathit{CancelS}_i]^{t_1-1}_{x_1}) \mid \phi(W) \end{array}$$

where W represents the rest of the system. At this point, a communication on offer between $\mathit{NewBuyer}$ and Seller_1 becomes possible. Omitting process $\mathit{Seller}_1 \blacktriangleleft [\mathbf{0}\,;\ \mathbf{0}]^{\infty}_{\emptyset}$, at the buyer side we have:

$$\begin{array}{l} (\nu c)\ (\mathit{NewBuyer} \blacktriangleleft [c \blacktriangleleft [\mathsf{offer!}(\mathit{prod}).\ \mathsf{price?}(p).\ (\cdots)\,;\ Q_1]^{1}_{v_1} \mid \phi(S_b)\,; \\ \qquad\qquad \mathit{CancelOrder}]^{t_{max}}_{x}) \\ \qquad\qquad \xrightarrow{c\ \mathsf{offer!}(\mathit{prod})} (\nu c)\ (\mathit{NewBuyer} \blacktriangleleft [c \blacktriangleleft [\mathsf{price?}(p).\ (\cdots)\,;\ Q_1]^{0}_{v_1} \mid \phi^2(S_b)\,; \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{CancelOrder}]^{t_{max}}_{x}) \end{array}$$

while Seller_1 makes an input transition located at c, inferred using (Out) and (LocL):

$$\begin{array}{l} (\nu c)\ (\mathit{Seller}_1 \blacktriangleleft [\mathit{DB} \mid c \blacktriangleleft [\mathsf{offer?}(\mathit{prod}).\ \mathsf{askPrice}^{\uparrow}!(\mathit{prod}).\ (\cdots)\,;\ R_1]^{b_1}_{w_1}\,; \\ \qquad\qquad \mathit{CancelS}_i]^{t_1-1}_{x_1}) \mid \phi(W) \\ \qquad\qquad \xrightarrow{c\ \mathsf{offer?}(\mathit{prod})} (\nu c)\ (\mathit{Seller}_1 \blacktriangleleft [\mathit{DB} \mid c \blacktriangleleft [\mathsf{askPrice}^{\uparrow}!(\mathit{prod}).\ (\cdots)\,;\ R_1]^{b_1-1}_{w_1}\,; \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{CancelSell}_1]^{t_1-1}_{x_1}) \mid \phi^2(W) \end{array}$$

Again, these complementary transitions can synchronize, thus firing an unobservable transition inferred using rule (Comm). The system then evolves to

$$\begin{array}{l} (\nu c)\ (\mathit{NewBuyer} \blacktriangleleft [c \blacktriangleleft [\mathsf{price?}(p).\ (\cdots)\,;\ Q_1]^{0}_{v_1} \mid \phi^2(S_b) \mid\ ;\ \mathit{CancelOrder}]^{t_{max}}_{x} \\ \mid \mathit{Seller}_1 \blacktriangleleft [\mathit{DB} \mid c \blacktriangleleft [\mathsf{askPrice}^{\uparrow}!(\mathit{prod}).\ (\cdots)\,;\ R_1]^{b_1-1}_{w_1}\,;\ \mathit{CancelSell}_1]^{t_1-1}_{x_1}) \mid \phi^2(W) \end{array}$$

At this point, the default behavior of the system establishes that $Seller_1$ contacts DB in order to communicate the price of the selected product to $NewBuyer$. However, we notice that conversation context c inside $NewBuyer$ has reached a timeout. As a consequence, the only possible way for progressing is by engaging into the compensating behavior, represented by process Q_1. Assuming $Q_1 \xrightarrow{\lambda} Q_1'$, the evolution of the compensating behavior can be inferred using rule (Comp). We then have:

$$(vc)\,(NewBuyer \blacktriangleleft [c \blacktriangleleft [\text{price?}(p).\,(\cdots);\, Q_1']_{v_1}^{0} \mid \phi^3(S_b);\, CancelOrder]_x^{tmax}$$
$$Seller_1 \blacktriangleleft [DB \mid c \blacktriangleleft [\text{askPrice}^{\uparrow}!(prod).\,(\cdots);\, R_1]_{w_1}^{b_1-1};\, CancelSell_1]_{x_1}^{t_1-1})) \mid \phi^3(W) \quad .$$

Appendix 7.B Proofs of Proposition 7.6.4

For $\mathcal{R}_1$, consider the relation of pairs of process states between T and S^F denoted by the dotted lines in Figure 7.11:

For $\mathcal{R}_2$, consider the relation of pairs of process states between T and S^T denoted by the dotted lines in Figure 7.12[3].

Finally, the counterexamples are built from the evolutions of S^F and S^T presented in Figure 7.11 and Figure 7.12, respectively. For $\mathcal{R}_4$, the relation breaks when S^T reach state S_8^T, that enables conversation A to start its compensating part after reaching a timeout. At this point of the evolution, the matching state at $S^F(S_8^F)$ cannot restart the process, and therefore will be blocked. Similar case happens when constructing $\mathcal{R}_5$, as it is impossible to map the states derived from the explicit compensation procedure generated by S^F in state S_2^F,

[3]In this example, we consider $t_B > t_A$.

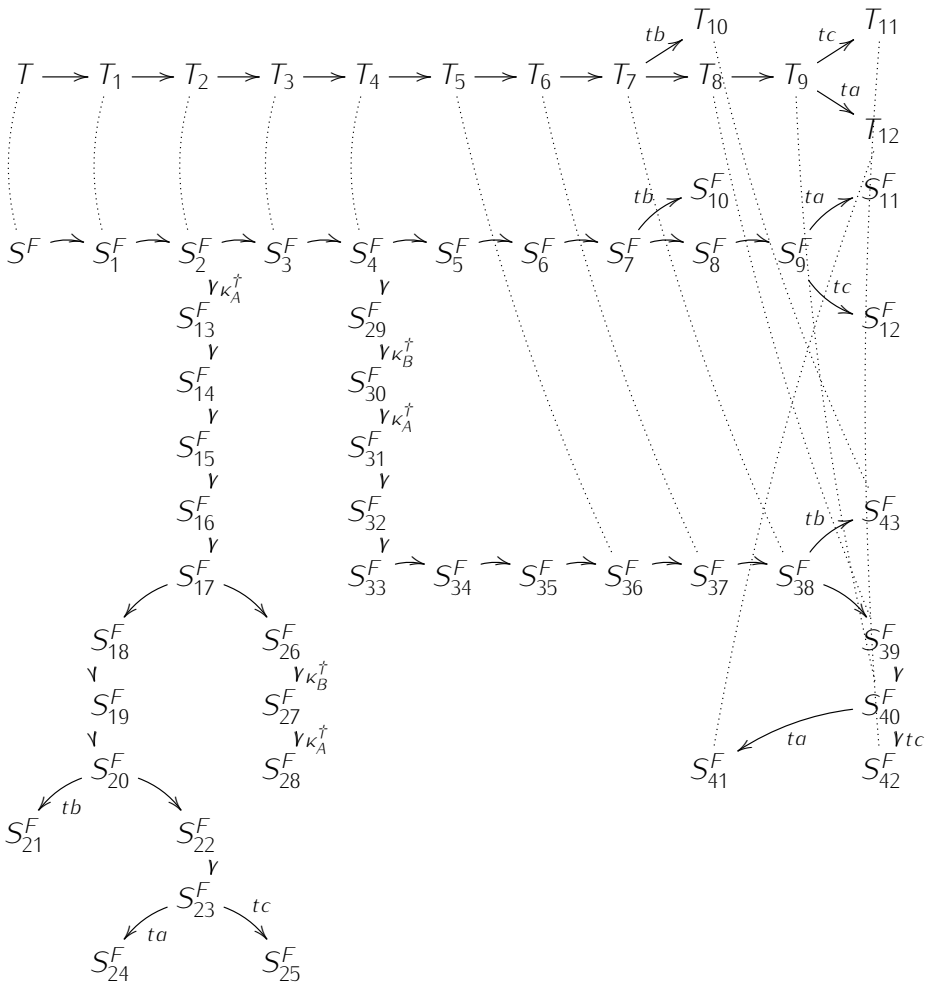


Figure 7.11: Relations between T and S^F.

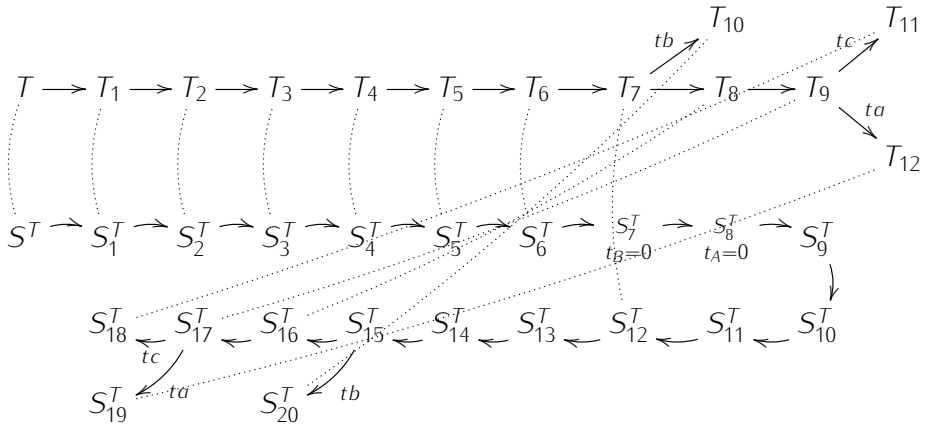


Figure 7.12: Relations between T and S^T.

Chapter 8

Towards Refinement Relations in Open Specifications

In Chapter 5 and Chapter 6 we discussed how declarative visions of communication-centred programs can provide more flexibility to specifications. The connection between program specifications and a logical framework can provide such flexibility, and allows for automated verification of program specifications with respect to a logical formulae. In this chapter we explore how to introduce such flexibility *directly* in the specifications. In this chapter we present initial ideas towards *Open Specifications*. An open specification has two components: a system description that presents the sequence of activities that must be performed, and "open" activities: tasks that a system may do and still conform to the specification. Here we present short notes on two initial, independent ideas towards the definition of refinement relations for open specifications. First, in Section 8.1 we propose a new denotational behavioural model called open mixed trees which generalises standard model of labelled trees (where labels are marked as negative, positive or both) by annotating each state with a set of so-called open actions and a flag indicating if termination is allowed in the state or not. The definition of refinement is then a generalisation of covariant-contravariant simulation that also takes account of termination and allows intermediate open parts of the specification. Second, in Section 8.2 we explore transition systems with responses for the specification of open specifications. A transition system with responses is a new generalisation of modal transition systems that allows for natural of deadlock freedom and liveness for infinite computations. Here we present a definition of refinement that fits transition systems with responses.

Contents

[0]This chapter collects the ideas presented in [Carbone *et al.* 2011, Carbone *et al.* 2012]

8.1 Refinement for Open Mixed Trees

Motivation The most common way of specifying concurrent systems is to take a set of communicating processes, and establish their interactions using input-output primitives. Think for a second on healthcare workflow process in which you have a patient Alice, a doctor Bob, and a Social Worker Charlie. The workflow describes the case were a patient feels dizzy and comes to the doctor to get diagnosed, prescribed and controlled along his illness. The following set of activities are included in the first specification S:

1. Alice comes to Bob for a medical appointment.
2. Bob receives Alice and gathers her symptomatology.
3. After consultation, Bob formulates a medicine treatment for Alice.
4. Bob sends the medicine formulation to Charlie, so he can deliver it to Alice.
5. Alice gets the medicine from Charlie and starts taking her treatment regularly as specified by Bob.
6. After some days, Alice comes back to Bob for a control, and the symptoms have disapeared.

Many details have been hindered from this example. First of all, it only details the interaction between three of the main actors involved. We may have a private health care institution that has to fulfill the auditing processes, where between activity 2 and 3. other actors will come into play. Our specification S could be extended accordingly to a new model S' including the two actions:

1. Alice comes to Bob for a medical appointment.
2. Bob receives Alice and gathers her symptomatology.
3. - On insuficiency of information, Bob will take blood samples and supplementary tests from Alice.
   - On cases with high variability, Bob will consult a pool of specialists on Alice's case.
4. Bob sends the medicine formulation to Charlie, so he can deliver it to Alice.
5. Alice gets the medicine from Charlie and starts taking her treatment regularly as specified by Bob.
6. After some days, Alice comes back to Bob for a control. and the symptoms have dissapeared.

It is to note, that even when S' has more behavior than S, it is still constrained to a set of activities that can be performed. The extra set of activities can be repeated many times and with different execution orders, but activities outside this set have to be ruled out. For instance, Bob cannot start operating Alice just after having gathered her symptomatology.

How is S related to S'? It is clear that the notion that we are looking for has a lot to do with the notion of *refinement*. Basically, refinement tells us that a specification S and an implementation S' are related if the set of behaviors in S is a subset of the set of behaviors exhibited by S'. There have been a myriad of papers during more than thirty years exploring different notions of partial transition systems and refinements capturing the relationship between abstract and concrete specifications, with views coming from branches as diverse as simulation and testing relations, modal transition systems, and abstract interpretation [Cousot & Cousot 1977, Larsen *et al.* 2007, Antonik *et al.* 2008a], and often applied to specific realms, like control theory [Baeten *et al.* 2010] and communication-centred programming [Bravetti & Zavattaro 2007].

In many cases we will specify systems by adding up more and more roles (and their respective behaviors) over the time. This, will lead us to start with a specification like S, knowing that each of the actions can be further refined with more and more behavior. We propose a new controlled way, called *open refinement*, to specify where and which actions can be inserted. The idea is in addition to standard transitions $P \xrightarrow{a} P'$ where a process P exhibits an immediate action a before evolving into P' to also specify *open* states $A \circlearrowright P \xrightarrow{a} P'$, where the process P can exhibit a finite series of actions in A before evolving with a into P'. The open state allows us to describe explicit stages in a process in which a process can be refined with any of the actions in a constrained set A. Here, transitions become weaker, as they might need more than one step for moving from P to P', but they also become broader than the standard weak transitions, as the set A can involve several (and possibly visible) actions and not just a dedicated internal action.

These changes lead us to proposing a new notion of refinement we call *open mixed refinement*. Starting from the covariant-contravariant simulations (that allow mixed, externally and internally controlled, activities and captures the necessary difference between such) we add the new notion of open states and also the ability to specify explicitly if a system may terminate in a state from which additional internally controlled activities are possible.

We believe the proposed model has both good uses in practice and good properties, i.e. can be given a clean categorical representation. We start in this brief abstract by giving the definition and the first result that open mixed refinement specializes to covariant-contravariant simulation if one allows no open states and always allows termination.

8.1.1 Open Mixed Trees and Refinement

Definition 8.1.1 (Open Mixed Trees). *An* open mixed tree *is a tuple*

$$T = \langle S, s_0, \mathrm{Act}^-, \mathrm{Act}^+, \sigma, \rightarrow \rangle$$

where

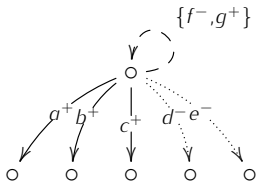
- S *is a set of* states,
- $s_0 \in S$ *is the* initial state,
- $\mathrm{Act} = \mathrm{Act}^- \cup \mathrm{Act}^+$ *is a set of actions characterized as externally controlled actions in* Act^- *(denoted by* a^-*) and internally controlled actions in* Act^+ *(denoted by* a^+*).*
- $\rightarrow \subseteq S \times \mathrm{Act} \times S$ *is a labelled transition relation between states*
- $\sigma : S \rightarrow \mathcal{P}(\mathrm{Act} \cup \{\checkmark\})$ *defines for each state the* open actions *and the possibility of terminating*
- $\checkmark \notin \sigma(s) \implies \exists s \xrightarrow{b^+}$*, i.e. an internally controlled action must be possible from every non-terminating state*
- $\forall s \in S$*, there exists a unique path* $S_0 \longrightarrow^* s$ *(i.e. the transition relation forms a tree)*

An open mixed tree where $\sigma(s) = \{\checkmark\}$ *for all* $s \in S$*, i.e. an open mixed tree with no open actions and which allow termination in every state, is referred to as just a mixed tree. A mixed tree is equivalent to a normal tree labelled with positive and negative labels.*

Intuitively, an open mixed tree represents the specification of a reactive, nondeterministic system with both internally controlled actions (e.g. output) and externally controlled actions (e.g. input). Note that there may be actions in $\mathrm{Act}^- \cap \mathrm{Act}^+$ that are both externally and internally controlled.

In any state with at least one internally controlled action any implementation must be able to do at least one of the internally controlled actions, or terminate if termination is also allowed by the specification. The states in which it is allowed to terminate is defined by the set T. Note that in order to not have any contradictions a state which is not in T (i.e. termination without further internally controlled actions is not allowed) must have at least one internally controlled action out of it.

Finally, the function σ pairs each state with a set of *open* (or underspecified) behavior, which allows an implementation to perform any action within the set a finite number of times before progressing (or terminating if the state is in T. We can depict open trees easily:



Below we write $s_1 \xrightarrow{\ell^d} s_2$ when $\{s_1, \ell, s_2\} \in \rightarrow$ and $\ell \in \mathrm{Act}^d$. Similarly, we write $\sigma^+(s_1)$ for the set of transitions such that $s_1 \xrightarrow{a^+} s_1' \in \sigma$

Definition 8.1.2 (Refinement of Open Mixed Trees). *A binary relation $\mathcal{R} \subseteq S_1 \times S_2$ between the state sets of two open mixed trees $P_j = \langle S_j, i_j, \mathrm{Act}^-, \mathrm{Act}^+, \sigma_j, \rightarrow_j \rangle$ for $j \in \{1, 2\}$ is a refinement if $i_1 \mathcal{R} i_2$ and $s_1 \mathcal{R} s_2$ implies*

1. $\forall s_1 \xrightarrow{a^-}_1 s_1'$, *implies* $\exists s_2 \xrightarrow{a_1}_2 s_{2,1} \xrightarrow{a_2}_2 \cdots \xrightarrow{a_n}_2 s_{2,n} \xrightarrow{a^-}_2 s_{2,n+1}'$, $a_i \in \sigma_1(s_1)$, *and* $s_1 \mathcal{R} s_{2,i}$

2. $\forall s_2 \xrightarrow{a^+}_2 s_2'$ *implies (i)* $\exists s_1 \xrightarrow{a^+}_1 s_1'$, *and* $s_1' \mathcal{R} s_2'$ *or (ii)* $a \in \sigma_1^+(s_1)$ *and* $s_1 \mathcal{R} s_2'$

3. $\sigma_2(s_2) \subseteq \sigma_1(s_1)$,

4. $\checkmark \in \sigma(s_1) \implies s_2 \xrightarrow{a_1}_2 s_{2,1} \xrightarrow{a_2}_2 \cdots \xrightarrow{a_n}_2 s_{2,n}$ *and* $a_i \in \sigma_1(s_1)$, $s_1 \mathcal{R} s_{2,i}$ *and* $\checkmark \in \sigma_s(s_{2,n})$.

5. *if* $s_2 = s_{2,0} \xrightarrow{a_0}_2 s_{2,1} \xrightarrow{a_1}_2 s_{2,2} \xrightarrow{a_2} \cdots$, $s_1 = s_{1,0}$ *and* $(s_{1,i} \xrightarrow{a_i} s_{1,i+1}$ *or* $(s_{1,i} = s_{1,1+1}$ *and* $a_i \in \sigma_1(s_{1,i}))$ *and* $s_{1,i} \mathcal{R} s_{2,i}$ *for* $i \in \omega$, *then* $|\{s_{1,i}\}_{i \in \omega}| = \omega$.

We say that Q is a refinement *of P, written $P \sqsubseteq Q$, whenever there exists a relation $\mathcal{R}$ such that $P \mathcal{R} Q$.*

Proposition 8.1.3. *The refinement relation $\sqsubseteq$ between open mixed trees as defined above*

1. *is reflexive and transitive, and*

2. *contains the identity relation*

As stated in the proposition below, refinement specializes for mixed trees (i.e. open mixed trees with no open actions and which allow termination in every state) to the notion of covariant-contravariant simulation defined in [Fábregas *et al.* 2010, Aceto *et al.* 2011].

Definition 8.1.4 (covariant-contravariant simulation [Fábregas *et al.* 2010]). *Given $P = (P, B, \rightarrow_P)$ and $Q = (Q, B, \rightarrow_Q)$, two LTS for the alphabet B, and $\{B^r, B^l, B^{bi}\}$ a partition of this alphabet. A (B^r, B^l)−simulation (or just a covariant-contravariant simulation) between them is a relation $S \subseteq P \times Q$ such that for every pSq we have that:*

- *for all* $a \in B^r \cup B^{bi}$ *and all* $p \xrightarrow{a} p'$ *there exists* $q \xrightarrow{a} q'$ *with* $p' S q'$.
- *for all* $a \in B^l \cup B^{bi}$ *and all* $q \xrightarrow{a} q'$ *there exists* $p \xrightarrow{a} p'$ *with* $p' S q'$.

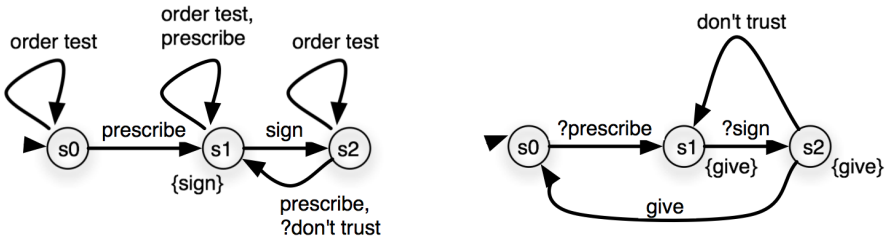
Proposition 8.1.5. *Refinement for open mixed trees coincides with covariant-contravariant simulations, taking* $B^r = \mathrm{Act}^+ \backslash \mathrm{Act}^-$, $B^l = \mathrm{Act}^- \backslash \mathrm{Act}^+$ *and,* $B^{bi} = \mathrm{Act}^- \cap \mathrm{Act}^-$.

8.2 Refinement for Transition Systems with Responses

Motivation Modal transition systems (MTS) were introduced originally in the seminal work of Larsen and Thomsen [Larsen & Thomsen 1988] (see also [Antonik *et al.* 2008b]) as a basic transition system model supporting stepwise specification and refinement of parallel processes. A MTS can be regarded as a labeled transition system (LTS) in which a subset of the transitions are identified as being required (must), while the others are allowed (may). In a MTS every required transition is also allowed, to avoid inconsistencies. A MTS describes simultaneously an over-approximation and an under-approximation of a process in an intertwined manner. In a stepwise refinement scenario this approximation interval is narrowed down to a single process, an LTS.

Subsequent work has lifted the assumption that required transitions need also be allowed, leading to the model of mixed transition systems [Dams 1996]. This means, that mixed transition systems allow states to have requirements that are not possible to fulfill, which we will refer to as *conflicting* requirements. However, the notion of a must transition that is not also a may transition appears quite intricate; it calls for interpreting the specifications at the targets of the must transitions which must *all* be satisfied in conjunction with some choice of may transition. We propose to take a step back and sketch a generalization of MTSs with a restricted kind of must transitions that allow for a simpler semantics. We simplify the exposition by restricting our attention to *action-deterministic* transition systems, where for each action a there is at most one a-transition from each state. We propose to replace the must transitions by a set of must *actions* assigned to every state. For readers familiar with mixed transition systems, this resembles a must transition to a "top" state from which every action is possible as a may transition. We refer to this set of actions as the response (or must) set, and we name the resulting model *Transition Systems with Responses* (TSR). We believe the mixed transition systems represented by TSRs are much simpler to understand and work with, and yet they still capture a rich set of specifications. Indeed, TRSs arise as the natural transition system underlying Dynamic Condition Response (DCR) Graphs (e.g. [Hildebrandt & Mukkamala 2010, Hildebrandt *et al.* 2011]), which generalize event structures to allow finite, executable specifications of ω-regular languages and are particularly useful for specification of flexible workflows where many actions are optional and liveness properties are needed.

Consider the example of Figure 8.1, illustrating two parts of a medical workflow described as TSRs. The TSR given in Figure 8.1(a) shows that the doctor may



(a) Medication, doctor (b) Medication, nurse

Figure 8.1: Medication workflow as two interacting transition systems with responses

first order any number of tests and then prescribe some medicine. Having prescribed medicine, it becomes a requirement to sign the prescription, so the response set of state $s1$ now contains the action **sign**. The TSR for a nurse given in Figure 8.1(b) may be interpreted similarly: If the nurse receives a prescription then the TSR moves to state $s1$ in which **give** is included in the response set, meaning that the medication must be given. However, this requirement cannot be satisfied in the present state, since there is no outgoing transition labelled with **give**. This reflects the rule in the workflow that the nurse is not allowed to give medicine before the prescription is signed. If a signature is received, then the nurse still has the requirement to do a **give** transition, and so can finally perform it and return to the initial state. However, the nurse can also choose to do a **don't trust** action, which signals to the doctor that signing must occur again. In the doctor's TSR the **?don't trust** action takes control back to the state where *sign* is required as response.

8.2.1 Transition Systems with Responses and Refinement

Definition 8.2.1 (Mixed Transition Systems). *A* Mixed Transition System *is a tuple* $T = \langle S, s_0, \text{Act}, \rightarrow_\Box, \rightarrow_\Diamond \rangle$ *where* S *is a set of* states, $s_0 \in S$ *is the* initial state, Act *is a set of actions, and* $\rightarrow_\Box, \rightarrow_\Diamond \subseteq S \times \text{Act} \times S$ *are respectively* must *and* may *transition relations.* T *is also a* Modal Transition System *(MTS) if additionally* $\rightarrow_\Box \subseteq \rightarrow_\Diamond$.

Definition 8.2.2 (Transition Systems with Responses). *A* Transition System with Responses *(TSR) is a tuple* $T = \langle S, s_0, \text{Act}, \Box, \rightarrow \rangle$ *where* S, s_0, Act *are like above and* $\rightarrow \subseteq S \times \text{Act} \times S$ *is a transition relation,* $\Box : S \rightarrow \mathcal{P}(\text{Act})$ *defines for each state the* response actions *that must be executed. Let* $\Diamond(s) =_{def} \{a \mid \exists s'.s \xrightarrow{a} s'\}$, *i.e., the actions on transitions that may be taken from* s. *We refer to a finite or infinite sequence of transitions starting at the inital state as a run. A run is accepting if for any intermediate state* s *in the run,* $a \in \Box(s)$ *implies eventually after that state there will be a transition in the run labelled with the action* a *or a state* s' *where* $a \notin \Box(s')$.

Proposition 8.2.3. *Action-deterministic modal transition systems correspond to the subset of TSRs where for all states* s *it holds that* $\Box(s) \subseteq \Diamond(s)$. *TSRs correspond*

to the subset of mixed transition systems of the form $\langle S \uplus \{t\}, s_0 \in S, \mathrm{Act}, \rightarrow_\Box \subseteq S \times \mathrm{Act} \times \{t\}, \rightarrow_\Diamond = \rightarrow \cup \{t\} \times \mathrm{Act} \times \{t\} \rangle$*, where* $\rightarrow \subseteq S \times \mathrm{Act} \times S$*.*

An action-deterministic MTS $M = \langle S, s_0, \mathrm{Act}, \rightarrow_\Box, \rightarrow_\Diamond \rangle$ can be represented as the TSR $R(M) = \langle S, s_0, \mathrm{Act}, \Box, \rightarrow_\Diamond \rangle$ where $\Box(s) =_{def} \{a \mid \exists s' \in S . s \xrightarrow{a} s'\}$ and a TSR $T = \langle S, s_0, \mathrm{Act}, \Box, \rightarrow \rangle$ as the action-deterministic MTS $M(T) = \langle S, s_0, \mathrm{Act}, \rightarrow_\Box, \rightarrow_\Diamond \rangle$, where $\rightarrow_\Box = \{(s, a, s') \mid a \in \Box(s) \wedge s \xrightarrow{a} s'\}$ and $\rightarrow_\Diamond = \rightarrow$. A TSR $T = \langle S, s_0, \mathrm{Act}, \Box, \rightarrow \rangle$ corresponds to a mixed transition system $Mix(T) = \langle S \uplus \{t\}, s_0, \mathrm{Act}, \rightarrow_\Box, \rightarrow_\Diamond \rangle$ where $\rightarrow_\Box = \{(s, a, t) \mid a \in \Box(s)\}$ and $\rightarrow_\Diamond = \rightarrow \cup \{t\} \times \mathrm{Act} \times \{t\}$.

Definition 8.2.4 (Deadlock and Liveness). *A* deadlock *state in a TSR* $T = \langle S, s_0, \mathrm{Act}, \Box, \rightarrow \rangle$ *is a state with a non-empty* must *set, and no outgoing transitions, i.e., a state in which some actions are required but no further transitions are possible. Formally we define a predicate deadlock on* S *by* $deadlock(s) \equiv \Box(s) \neq \emptyset \wedge \Diamond(s) = \emptyset$*. A TSR is* deadlock free *if it has no reachable deadlock state. A* live *state is one from which there exists an accepting run. A TSR is live if all reachable states are live.*

Definition 8.2.5 (Refinement of TSRs). *A binary relation* $\mathcal{R} \subseteq S_1 \times S_2$ *between the state sets of two transition systems with responses* $T_j = \langle S_j, i_j, \mathrm{Act}, \Box_j, \rightarrow_j \rangle$ *for* $j \in \{1, 2\}$ *is a refinement if* $i_1 \mathcal{R} i_2$ *and* $s_1 \mathcal{R} s_2$ *implies*

1. $\forall s_1 \xrightarrow{a}_1 s_1'$ *and* $a \in \Box_1(s_1)$ *implies* $\exists s_2 \xrightarrow{a}_2 s_2'$*,* $a \in \Box_2(s_2)$ *and* $s_1' \mathcal{R} s_2'$*,*
2. $\forall s_2 \xrightarrow{a}_2 s_2'$ *implies* $\exists s_1 \xrightarrow{a}_1 s_1'$ *and* $s_1' \mathcal{R} s_2'$

The refinement $\mathcal{R}$ *is safe if it reflects deadlock states, so* $deadlock(s_2) \implies deadlock(s_1)$ *whenever* $s_1 \mathcal{R} s_2$*.*

Proposition 8.2.6. *Given two TSRs* $T_i = \langle S_i, s_{i,0}, \mathrm{Act}, \Box_i, \rightarrow_i \rangle$ *for* $i \in \{1, 2\}$*, if* $\mathcal{R} \subseteq S_1 \times S_2$ *is a safe refinement then* T_2 *is deadlock free if* T_1 *is deadlock free.*

An example of a safe refinement of the TSR in Figure 8.1(a) is the TSR obtained by removing **order test** transitions. Since the first **prescribe** is not in $\Box$, another example is a TSR with just the initial state.

An example of a non-safe refinement of the TSR in Figure 8.1(b) is the TSR obtained by removing transition labelled with **?sign**. However, note that this action belongs to the *interface* of the TRS, i.e., it is controlled by the environment. This suggests as a next step studying a variant of refinement for TSRs where the interface actions are preserved akin to partial bisimulation [Rutten 2000] or alternating simulation [Alfaro & Henzinger 2001].

8.3 Discussion and Future Work

In the previous sections we have presented two generalisations of transition systems with certain degree of flexibility when describing specifications, as well as refinement as a way of relating different specifications with different degrees of information.

They emerged from the simple motivation of having denotational models that allow for descriptions of systems that capture precisely the behaviour of a system but still allows certain degree of flexibility by including actions that may appear in implementations of the system. The first approach introduced open mixed trees as a generalisation of labelled trees that capture internal and external choices, termination and a set of open states. A refinement relation for open mixed trees then captures cases where an implementation accomplish all the actions involved in the standard specification, but can as well include optional actions left underspecified in open sets. The second approach introduced Transition Systems with Responses (TSRs) as a new generalisation of Modal Transition Systems which represents a restricted class of mixed transition systems that are much simpler than general mixed transition systems, and yet which remain expressive and allow natural definitions of deadlock freedom and liveness for infinite computations. We have proposed a notion of refinement, exemplified by a medical workflow consisting of two interacting TSRs.

As a future work, we aim at a unified framework were characterisations of refinement relations over transition systems with different characteristics (mixed behaviours, deadlock freedom, termination) can be studied. In particular, we believe that a category- theory view of concurrent processes can be suitable for the study of specifications with mixed behaviours, and in particular the evolution of concurrent systems with partial information. In a categorical framework, one could study specification models isolating each of their features alone, and show how the features can be combined, in the same style it has been done for functional simulation and bisimulation relations in [Joyal *et al.* 1993, Fiore *et al.* 1999, Winskel 2005].

With respect to the study of transition systems with responses, we aim at a further study of deadlock and liveness properties, as well as the detail of refinement and bisimulation for TSRs, and the relation to other models with liveness, such as DCR Graphs in [Hildebrandt & Mukkamala 2010, Hildebrandt *et al.* 2011] and Harel's Live Sequence Charts (LSCs) [Damm & Harel 2001]. This will include lifting the restriction to action-deterministic systems, which can be done by considering TSRs with transitions carrying labelled events (as in asynchronous transition systems and labelled event structures) and response sets being sets of events, not actions.

Chapter 9

Final words

9.1 Conclusions

Communication-centred programming is becoming a central matter of research today, at a time where the design of complex computation systems becomes more and more a task of defining communication and coordination protocols among entities. The fact that architectures for such entities face a high level of decentralisation makes the description of such protocols quite difficult in practice, and requires protocol designers to work at different levels abstraction that need to be interrelated. In this thesis we explored two of them: Descriptions featuring a global view of interactions between participants (choreographies) and descriptions featuring local views on how each participant reacts with respect to an environment (orchestrations). Along the previous 8 chapters we have explored programming languages techniques for choreographies and orchestrations, with the ultimate goal of integrating imperative descriptions of communication-centred programs with their declarative counterparts in terms of logical descriptions. We found out that although both imperative and declarative approaches for specifying communication-centred programs are quite mature, little has been done towards establishing the relation between them. The aspects taken into consideration during the writing of this document have been diverse, and topics here exposed range between (variants) of process calculi for communicating processes, logical characterisations of message-passing concurrency, type systems, behavioural (refinement) relations between processes, and timing specifications. The results in this thesis can be summarised below:

Logical Characterisations of Communication-Centred Programming Starting from imperative and declarative ways of specifying communication-centred programs, we established connections between them by means of logical characterisations of choreographies and orchestrations. With respect to choreographies: we introduced a modal logic that allows for flexible descriptions of interactions in a global setting. With respect to orchestrations, we explored the connections of the specifications of services with respect to modal and linear temporal logics. These connections allow for checking the conformance of an already running specification (the imperative view) against a logical formula describing the minimal set of constraints required in such scenario (the declarative view). Moreover, a mapping from logics for choreographies to logics for orchestrations is provided, in a way that one can project the formula characterising the good behaviour of a global specification, and check the conformance of their associated end-points.

Behavioural Types for Communication and Security One of the main interests of this thesis is to provide for mechanisms where communications can evidence certain properties regarding the correctness of their interactions. Apart from logics, we explore type systems to ensure such properties. First, we build upon languages for session types to guarantee that protocol descriptions conform to a certain control flow, and that allows for mappings between global and local views. The connection between session typed calculi and the logical characterisations here presented allow one to describe meaningful mappings: that is, global specifications that respect a certain explicit behaviour (global type) and correspond to a logical formula, map correctly to meaningful descriptions of orchestrations (end-point projections) and correspond to the logical projection of global formulae. A second type system here explored describes particular aspects of communications in concurrent constraint languages: Here we show that a typing discipline restricting the use of variables in a specification of a system allows for guarantees about the good behaviour of a system, describing appropriately communication and security protocols. This is (we believe) the first work on behavioural type systems for a language of concurrent constraints, and opens the landscape for the description of more complex typing scenarios.

Timing and Exceptional Behaviour in Structured Communications We started explorations on concepts like compensating behaviours and timed specifications of communication-centred programs. The study of temporal and exceptional behaviour was performed by adding minimal extensions to languages of communication-centred programming with primitives that allow distribution of time and compensating behaviour. From such studies we found that both exceptional and timing behaviour need to be consider together, and not in isolation when describing models of interactions. Moreover, the description of interactions with multiple levels of nesting (something particularly useful in service oriented architectures) greatly complicates the semantics of the compensation behaviour, and requires a more flexible treatment than the one used to control exceptions in imperative languages.

We hope this work evidences the connections between declarative and imperative styles in the description of communication protocols, and works as an initial contribution where research in this important area can be built upon.

9.2 Current and Future work

In this dissertation we have already pointed towards different directions for future work. In this section we conclude by providing some comments on which directions we believe are particularly interesting. Some of them are the object of current work.

Model Checking Communication-Centred Programs Along this whole thesis we have advocated for connections between specifications and logics when describing communication protocols. The most natural idea expanding from this work is the development of automated reasoning tools that allow one to mechanise the satisfiability

checking of formulae in specifications of communication-centred programs. In order to provide such tools both theory and practice have to be advanced. As we presented in Chapter 5 with $\mathcal{GL}$, we face undecidability issues when trying to verify programming languages for communication-centred programs that feature recursion and restriction operators. It is necessary to restrict our models to a subset of the language, or to develop reasonable approximations regarding the models to avoid state space explosion problems. One strand of work will be to complement the results presented in Chapter 5 and Chapter 6 with model checking tools that allow us to verify the satisfaction of logics for communication-centred programs in an automated way.

More on Type Disciplines Type systems provide mechanisms to restrict the behaviour of programming languages in such a way that well-typed programs exhibit guarantees about the good behaviour of systems specified. Three directions regarding the use of type disciplines come out as inspiration of this work.

First, the results in Chapter 4 presented a first approach towards behavioural types for a language of concurrent constraints. One might expect that the use of Concurrent Constraint Programming languages for communication-centred programming require the adaptation of session types such that communications in CCP follow a certain order previously established in the communication protocol. The adaptation of session types to CCP is by no means straightforward, and one need to take into consideration that CCP is a general model of concurrency, that is specialised with constraint languages depending on their application. One might expect session types in such setting will emerge a specialisation's of a general class of type systems, on the same lines as general type systems for Bigraphs [Elsborg *et al.* 2009] and the Ψ Calculus [Hüttel 2011].

Second, the results involving time analysis in communication-centred programs presented in Chapter 7 opens an important question regarding the application of current type disciplines when involving timed specifications. The work in [Berger & Yoshida 2007] proposes typing analysis techniques for a variant of the asynchronous π-calculus with locations and time windows. A linear/affine type discipline presents a way to integrate time and linearity conditions in the analysis of interactions: by typing timed processes, one is able to provide further guarantees about the liveness conditions of the systems under consideration, A further step derived from this research involves the integration of a theory of timed types for global descriptions involving idealised, global time, accessible for all participating nodes in the protocol, that further can be projected to infrastructures where synchronisation of real, physical clocks is taken into consideration.

Third, we aim at exploring further connections between logics and behavioural types. We are particularly interested in the application of refinement types for description of communication-centred programs. Refinement Types are dependent types that embed first-order logic formulae [Gordon & Fournet 2009]. This framework provides a uniform setting to increase the expressive power of type systems by defining subsets of types through logical separation. Its recent introduction opens interesting

perspectives for the application of refinement types to session and security types, allowing one to express properties of protocols that go beyond the control flow behaviour of messages exchanged in a protocol.

Open Specifications, and General Visions of Refinement The development of a logical vision for communication-centred programs have left us with questions about the correct set of operators that we want to have in the logic. In this document we explored derivations of Hennessy-Milner Logics, where the main properties of interest involved action and may formulae both at the level of choreographies and end-points. The may operator tells us important information about the existence of an evolution where a property is fulfilled, but sometimes it can fall short by allowing other evolutions of the system that do not comply with the property. In [Carbone *et al.* 2011] we started studies on stronger versions of the may modality, where one is allowed to express that a property is fulfilled in all possible executions in an eventual state, and their implementation as part of the operators in $\mathcal{GL}$ is foreseen. Other improvements to the logics proposed include the use of fixed points, essential for describing state-changing loops, and auxiliary axioms describing structural properties of a choreography.

As future work, we aim at a unified framework where characterisations of refinement relations over transition systems with different characteristics (mixed behaviours, deadlock freedom, termination) can be studied. In particular, we believe that a categorical view of concurrent processes can be suitable for the study of specifications with mixed behaviours, and in particular the evolution of concurrent systems with partial information. In a categorical framework, one could study specification models isolating each of their features alone, and show how the features can be combined, in the same style it has been done for functional simulation and bisimulation relations in [Joyal *et al.* 1993, Fiore *et al.* 1999, Winskel 2005].

Nominal Concurrent Constraint Programming One of the long-withstanding goals in the research of CCP languages has been the correct representation of *mobile* behaviour over concurrent constraint programs. When referring to mobile behaviour we can consider either *link mobility*, the ability of the network to reconfigure the connections between nodes, or *process mobility*, the ability to reconfigure the topology of the network. Link mobility has been modelled in utcc [Olarte & Valencia 2008a]. In the classical setting, CCP-like calculi have modelled the logical view of a restriction operator as an existential quantifier over a constraint store. By this formulation one can say that a variable x is *private* from the constraint store c as no other process can know the contents of x in $\exists x$ c, except the one that imposed the constraint. In [Palamidessi *et al.* 2006] it has been noticed that such a logical characterisation of name restriction using the existential quantifier does not ensure uniqueness in the fragment of the π-calculus with mismatch: given $\exists x\, \exists y\, \mathrm{c}(\mathrm{x}, \mathrm{y})$ we cannot say that x is different than y, therefore the freshness of name generation cannot be guaranteed (as previously discussed in Chapter 4).

Given the importance of freshness and uniqueness conditions when dealing with

sessions, we aim for a reformulation of the name hiding on cc-calculi using a different conception. For doing so, we started working in a new variant of ccp-calculi, called *Fresh CCP*, to deal with name generation. In fresh CCP, fresh name generation is achieved by a redefinition of the underlying constraint system and the denotational model of CCP with the use of *nominal logic* [Pitts 2003], an extension of first order logic with bundled notions of name swapping and fresh terms.

Bibliography

[Abadi & Fournet 2001] Martín Abadi and Cédric Fournet. *Mobile values, new names, and secure communication*. In POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 104–115, New York, NY, USA, 2001. ACM Press. (Cited on page 99.)

[Abadi & Gordon 1999] Martín Abadi and Andrew D. Gordon. *A Calculus for Cryptographic Protocols: The SPi Calculus*. Inf. Comput., vol. 148, no. 1, pages 1–70, 1999. (Cited on pages 80 and 98.)

[Aceto *et al.* 2011] L. Aceto, I. Fábregas, D. de Frutos Escrig, A. Ingólfsdóttir and M. Palomino. *Relating modal refinements, covariant-contravariant simulations and partial bisimulations*. Fundamentals of Software Engineering, FSEN, 2011. (Cited on page 193.)

[Alfaro & Henzinger 2001] Luca de Alfaro and Thomas A. Henzinger. *Interface Automata*. In Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), pages 109–120, Vienna, Austria, September 2001. ACM Press. (Cited on page 196.)

[Andrews *et al.* 2003] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte*et al. Business process execution language for web services, version 1.1*. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003. (Cited on pages 14 and 20.)

[Antonik *et al.* 2008a] A. Antonik, M. Huth, K.G. Larsen, U. Nyman and A. Wasowski. *20 years of modal and mixed specifications*. European Association for Theoretical Computer Science. Bulletin, vol. 2, no. 95, 2008. (Cited on page 191.)

[Antonik *et al.* 2008b] Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman and Andrzej Wąsowski. *20 Years of Modal and Mixed Specifications*. Bulletin of EATCS, vol. 95, June 2008. Available at `http://processalgebra.blogspot.com/2008/05/concurrency-column-for-beatcs-june-2008.html`. (Cited on page 194.)

[Baeten *et al.* 2010] J.C.M. Baeten, D.A. van Beek, S.P. Luttik, J. Markovski and J.E. Rooda. *Partial Bisimulation*. SE Report 2010-04, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2010. (Cited on page 191.)

[Bartoletti & Zunino 2010] Massimo Bartoletti and Roberto Zunino. *A Calculus of Contracting Processes*. In LICS, pages 332–341. IEEE Computer Society, 2010. (Cited on page 20.)

[Bell & LaPadula 1973] D.E. Bell and L.J. LaPadula. *Secure computer systems: Mathematical foundations and model*. The MITRE Corporation Bedford MA Technical Report M74244 May, vol. 1, no. M74-244, 1973. (Cited on page 93.)

[Berger & Honda 2000] Martin Berger and Kohei Honda. *The Two-Phase Commitment Protocol in an Extended pi-Calculus*. Electr. Notes Theor. Comput. Sci., vol. 39, no. 1, 2000. (Cited on pages 15, 162, 182 and 183.)

[Berger & Yoshida 2007] Martin Berger and Nobuko Yoshida. *Timed, Distributed, Probabilistic, Typed Processes*. In Proc. APLAS, volume 4807 of *LNCS*, pages 158–174. Springer, 2007. (Cited on pages 19, 183 and 201.)

[Berger *et al.* 2001] Martin Berger, Kohei Honda and Nobuko Yoshida. *Sequentiality and the Pi-Calculus*. In S. Abramsky, editor, TLCA 2001, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer, Berlin Heidelberg, 2001. (Cited on page 68.)

[Berger *et al.* 2008] Martin Berger, Kohei Honda and Nobuko Yoshida. *Completeness and Logical Full Abstraction in Modal Logics for Typed Mobile Processes*. In Luca Aceto, editor, ICALP'08, number 5126 of LNCS, pages 99–111. Springer-Verlag, Berlin Germany, 2008. (Cited on pages 18, 19, 68, 107, 108, 118, 130, 153 and 154.)

[Biba 1977] KJ Biba. *Integrity considerations for secure computer systems. USAF Electronic System Division, Hanscom Air Force Base*. Technical report, Tech. Rep.: ESD-TR-76-372, 1977. (Cited on page 93.)

[Blanchet 2001] Bruno Blanchet. *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*. In 14th IEEE Computer Security Foundations Workshop (CSFW-14), pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society, Los Alamitos (2001). (Cited on pages 80 and 99.)

[Bocchi *et al.* 2010] Laura Bocchi, Kohei Honda, Emilio Tuosto and Nobuko Yoshida. *A theory of design-by-contract for distributed multiparty interactions*. In CONCUR'10: Proceedings of the 21st International Conference on Concurrency Theory, Lecture Notes in Computer Science. Springer - Verlag, August 2010. (Cited on pages 18, 118, 153 and 154.)

[Bonelli *et al.* 2005] E. Bonelli, A. Compagnoni and E. Gunter. *Correspondence assertions for process synchronization in concurrent communications*. Journal of Functional Programming, vol. 15, no. 2, pages 219–247, 2005. (Cited on pages 18 and 153.)

[Boreale *et al.* 2006] M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara and D. Sangiorgi. *SCC: a service centered calculus*. Proceedings of WS-FM, vol. 4184, pages 38–57, 2006. (Cited on pages 14, 56 and 120.)

[Boreale *et al.* 2008] M. Boreale, R. Bruni, R. De Nicola and M. Loreti. *Sessions and pipelines for structured service programming*. In Gilles Barthe and Frank de Boer, editors, Formal Methods for Open Object-Based Distributed Systems, volume 5051 of *Lecture Notes in Computer Science*, pages 19–38. Springer Berlin / Heidelberg, 2008. (Cited on page 14.)

[Bravetti & Zavattaro 2007] M. Bravetti and G. Zavattaro. *Towards a unifying theory for choreography conformance and contract compliance*. In Proceedings of the 6th international conference on Software composition, pages 34–50. Springer-Verlag, 2007. (Cited on page 191.)

[Bravetti & Zavattaro 2008] Mario Bravetti and Gianluigi Zavattaro. *A Foundational Theory of Contracts for Multi-party Service Composition*. Fundam. Inform., vol. 89, no. 4, pages 451–478, 2008. (Cited on page 20.)

[Bravetti & Zavattaro 2009] Mario Bravetti and Gianluigi Zavattaro. *On the expressive power of process interruption and compensation*. Mathematical Structures in Computer Science, vol. 19, no. 3, pages 565–599, 2009. (Cited on page 171.)

[Brogi *et al.* 2004] A. Brogi, C. Canal, E. Pimentel and A. Vallecillo. *Formalizing Web Service Choreographies*. In Proc. 1st. International Workshop on Web Services and Formal Methods, volume 105, pages 73–94. Elsevier, 2004. (Cited on page 5.)

[Broy 2007] Manfred Broy. *Interaction and Realizability*. In Jan van Leeuwen, Giuseppe Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack and František Plášil, editors, SOFSEM 2007: Theory and Practice of Computer Science, volume 4362 of *Lecture Notes in Computer Science*, pages 29–50. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-69507-3_3. (Cited on page 21.)

[Bruni & Mezzina 2008] Roberto Bruni and Leonardo Mezzina. Types and deadlock freedom in a calculus of services, sessions and pipelines. Submitted for Publication - AMAST 2008, February 2008. (Cited on page 5.)

[Bruni 2009] Roberto Bruni. *Calculi for Service-Oriented Computing*. In Marco Bernardo, Luca Padovani and Gianluigi Zavattaro, editors, Formal Methods for Web Services, volume 5569 of *Lecture Notes in Computer Science*, pages 1–41. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-01918-0_1. (Cited on page 13.)

[Buchholtz *et al.* 2004] M. Buchholtz, H. Riis Nielson and F. Nielson. *A calculus for control flow analysis of security protocols*. International Journal of Information Security, vol. 2, no. 3, pages 145–167, 2004. (Cited on pages 80, 85 and 96.)

[Buscemi & Montanari 2007] Maria Grazia Buscemi and Ugo Montanari. *CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements*. 16th

European Symposium on Programming (ESOP'07), 2007. (Cited on pages 15, 20 and 59.)

[Busi *et al.* 2006] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi and Gianluigi Zavattaro. *Choreography and Orchestration Conformance for System Design*. In Paolo Ciancarini and Herbert Wiklicky, editors, COORDINATION, volume 4038 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2006. (Cited on pages 14 and 120.)

[Caires & Cardelli 2001] L. Caires and L. Cardelli. *A spatial logic for concurrency (part I)*. In Theoretical Aspects of Computer Software, pages 1–37. Springer, 2001. (Cited on pages 108, 130 and 143.)

[Caires & Pfenning 2010] L. Caires and F. Pfenning. *Session Types as Intuitionistic Linear Propositions*. In CONCUR 2010-Concurrency Theory: 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010, Proceedings, volume 6269, page 222. Springer-Verlag New York Inc, 2010. (Cited on pages 18 and 153.)

[Caires & Vieira 2010] Luís Caires and Hugo Torres Vieira. *Conversation Types*. Theor. Comput. Sci., 2010. To appear. (Cited on pages 19, 163 and 183.)

[Caires *et al.* 2008] Luís Caires, Carla Ferreira and Hugo Torres Vieira. *A Process Calculus Analysis of Compensations*. In Proc. of Trustworthy Global Computing (TGC'08), volume 5474 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2008. (Cited on pages 162 and 172.)

[Campadello *et al.* 2006] S. Campadello, L. Compagna, D. Gidoin, S. Holtmanns, V. Meduri, J.C. Pazzaglia, M. Seguran and R. Thomas. *Scenario Selection and Definition*. Research report A7.D1.1, SERENITY consortium, 2006. (Cited on pages 173 and 174.)

[Capecchi *et al.* 2010] Sara Capecchi, Elena Giachino and Nobuko Yoshida. *Global Escape in Multiparty Sessions*. In Kamal Lodaya and Meena Mahajan, editors, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 338–351, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cited on pages 19, 162 and 182.)

[Carbone *et al.* 2004] Marco Carbone, Mogens Nielsen and Vladimiro Sassone. *A Calculus for Trust Management*. In Kamal Lodaya and Meena Mahajan, editors, FSTTCS, volume 3328 of *Lecture Notes in Computer Science*, pages 161–173. Springer, 2004. (Cited on pages 41 and 137.)

[Carbone *et al.* 2006] Marco Carbone, Kohei Honda and Nobuko Yoshida. *A Calculus of Global Interaction based on Session Types*. In 2nd Workshop on Develop-

ments in Computational Models (DCM), ENTCS, 2006. (Cited on pages 38, 39, 102, 103, 104, 124 and 125.)

[Carbone *et al.* 2007] Marco Carbone, Kohei Honda and Nobuko Yoshida. *Structured communication-centred programming for web services*. In 16th European Symposium on Programming (ESOP), volume 4421 of *LNCS*, pages 2–17, Braga, Portugal, March 2007. Springer, Berlin Heidelberg. (Cited on pages 2, 5, 10, 11, 18, 38, 39, 40, 45, 49, 51, 103, 104, 106, 110, 113, 118, 120, 121, 124, 125, 126, 127, 128, 133, 136, 138, 139, 140, 143 and 163.)

[Carbone *et al.* 2008] Marco Carbone, Kohei Honda and Nobuko Yoshida. *Structured Interactional Exceptions in Session Types*. In Proceedings of the 19th international conference on Concurrency Theory, pages 402–417. Springer-Verlag, 2008. (Cited on pages 19, 162, 164 and 182.)

[Carbone *et al.* 2009] M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown and S. Ross-Talbot. *A Theoretical Basis of Communication-Centred Concurrent Programming*. Web Services Choreography Working Group mailing list, to appear as a WS-CDL working report, 2009. (Cited on pages 5 and 107.)

[Carbone *et al.* 2010] Marco Carbone, Thomas Hildebrandt, Davide Grohmann and Hugo A. López. *A logic for Choreographies*. In 3rd Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), 2010. (Cited on pages 5, 11, 23 and 134.)

[Carbone *et al.* 2011] Marco Carbone, Thomas Hildebrandt and Hugo A. López. *Open Mixed Refinement*. In Nordic Workshop of Programming Theory (NWPT), Västerås, Sweden, November 2011. (Cited on pages 24, 153, 189 and 202.)

[Carbone *et al.* 2012] Marco Carbone, Thomas T. Hildebrandt, Hugo A. López, Gian Perrone and Andrzej Wasowski. *Refinement for Transition Systems with Responses*. In 4th International Workshop on Foundation of Interface Technologies (FIT), 2012. Accepted for publication. (Cited on pages 24 and 189.)

[Carbone 2008] Marco Carbone. *Session-based Choreography with Exceptions*. In N. Yoshida and V.T. Vasconcelos, editors, PLACES'08: Procs. of the 1st Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, volume 241 of *ENTCS*, pages 35–55, 2008. (Cited on pages 19 and 182.)

[Cardelli & Gordon 2000] Luca Cardelli and Andrew D. Gordon. *Anytime, Anywhere: Modal Logics for Mobile Ambients*. In POPL, pages 365–377, 2000. (Cited on pages 116 and 136.)

[Cardelli & Gordon 2006] Luca Cardelli and Andrew D. Gordon. *Ambient Logic*. Mathematical Structures in Computer Science, 2006. (Cited on page 143.)

[Castagna & Padovani 2009] Giuseppe Castagna and Luca Padovani. *Contracts for Mobile Processes*. In Mario Bravetti and Gianluigi Zavattaro, editors, CONCUR, volume 5710 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2009. (Cited on page 20.)

[Castagna *et al.* 2005] G. Castagna, R. De Nicola and D. Varacca. *Semantic Subtyping for the Pi-Calculus*. In Proc.. 20th Annual IEEE Symposium on Logic in Computer Science, LICS, pages 92–101, 2005. (Cited on page 20.)

[Castagna *et al.* 2008] Giuseppe Castagna, Nils Gesbert and Luca Padovani. *A theory of contracts for web services*. SIGPLAN Not., vol. 43, pages 261–272, January 2008. (Cited on page 20.)

[Cerone & Hennessy 2010] Andrea Cerone and Matthew Hennessy. *Process Behaviour: Formulae vs. Tests (Extended Abstract)*. In Sibylle B. Fröschle and Frank D. Valencia, editors, EXPRESS'10, volume 41 of *EPTCS*, pages 31–45, 2010. (Cited on page 53.)

[Cesari *et al.* 2010] L. Cesari, R. Pugliese and F. Tiezzi. *A tool for rapid development of WS-BPEL applications*. ACM SIGAPP Applied Computing Review, vol. 11, no. 1, pages 27–40, 2010. (Cited on page 15.)

[Charatonik & Talbot 2001] Witold Charatonik and Jean-Marc Talbot. *The Decidability of Model Checking Mobile Ambients*. In Laurent Fribourg, editor, CSL, volume 2142 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2001. (Cited on pages 111, 113 and 134.)

[Cook *et al.* 2006] W. Cook, S. Patwardhan and J. Misra. *Workflow patterns in Orc*. In Coordination Models and Languages, pages 82–96. Springer, 2006. (Cited on page 14.)

[Coppo & Dezani-Ciancaglini 2009] Mario Coppo and Mariangiola Dezani-Ciancaglini. *Structured Communications with Concurrent Constraints*. In TGC'08, volume 5474 of *LNCS*, pages 104–125. Springer, 2009. (Cited on pages 15, 18, 59 and 153.)

[Corin & Etalle 2002] Ricardo Corin and Sandro Etalle. *An Improved Constraint-based system for the verification of security protocols*. In M. V. Hermenegildo and G. Puebla, editors, 9th Int. Static Analysis Symp. (SAS), volume 2477 of *LNCS*, pages 326–341, Madrid, Spain, Sep 2002. Springer, Heidelberg. (Cited on pages 80 and 100.)

[Cousot & Cousot 1977] Patrick Cousot and Radhia Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In POPL, pages 238–252, 1977. (Cited on page 191.)

[Crazzolara & Winskel 2001] Federico Crazzolara and Glynn Winskel. *Events in security protocols*. In ACM Conference on Computer and Communications Security, pages 96–105, 2001. (Cited on pages 80, 81, 93 and 96.)

[Damm & Harel 2001] W. Damm and D. Harel. *LSCs: Breathing Life into Message Sequence Charts*. Formal Methods in System Design, vol. 19, no. 1, pages 45–80, 2001. (Cited on page 197.)

[Dams 1996] Dennis Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, July 1996. (Cited on page 194.)

[de Boer *et al.* 2000] FS de Boer, M. Gabbrielli and M.C. Meo. *A Timed Concurrent Constraint Language*. Information and Computation, vol. 161, no. 1, pages 45–83, 2000. (Cited on page 32.)

[De Nicola & Hennessy 1984] R. De Nicola and M.C.B. Hennessy. *Testing equivalences for processes*. Theoretical Computer Science, vol. 34, no. 1-2, pages 83–133, 1984. (Cited on pages 12, 53, 54, 98 and 180.)

[de Vries *et al.* 2010] Edsko de Vries, Vasileios Koutavas and Matthew Hennessy. *Communicating Transactions - (Extended Abstract)*. In Proc. CONCUR, volume 6269 of *Lecture Notes in Computer Science*, pages 569–583. Springer, 2010. (Cited on page 184.)

[Denning 1976] Dorothy E. Denning. *A lattice model of secure information flow*. Commun. ACM, vol. 19, pages 236–243, May 1976. (Cited on page 93.)

[Dezani-Ciancaglini & De'Liguoro 2010] Mariangiola. Dezani-Ciancaglini and Ugo De'Liguoro. *Sessions and session types: an overview*. In Proceedings of the 6th international conference on Web services and formal methods, pages 1–28. Springer-Verlag, 2010. (Cited on page 10.)

[Diaz *et al.* 1998] J.F. Diaz, C. Rueda and F. Valencia. *A calculus for concurrent processes with constraints*. CLEI Electronic Journal, vol. 1, no. 2, 1998. (Cited on page 59.)

[Dijkman *et al.* 2007a] Remco M. Dijkman, Marlon Dumas and Chun Ouyang. *Formal Semantics and Analysis of BPMN Process Models using Petri Nets*. Preprint 7115, Queensland University of Technology, April 2007. (Cited on page 21.)

[Dijkman *et al.* 2007b] R.M. Dijkman, M. Dumas and C. Ouyang. *Formal Semantics and Automated Analysis of BPMN Process Models*. Technical Report 5969, Queensland University of Technology, 2007. (Cited on page 21.)

[Dolev & Yao 1981] Danny Dolev and Andrew C. Yao. *On the security of public key protocols*. Technical report, Dept. of Computer Science, Stanford University, Stanford, CA, USA, 1981. (Cited on pages 94 and 98.)

[Dragoni & Mazzara 2010] Nicola Dragoni and Manuel Mazzara. *A Formal Semantics for the WS-BPEL Recovery Framework*. In Cosimo Laneve and Jianwen Su, editors, Web Services and Formal Methods, volume 6194 of *Lecture Notes in Computer Science*, pages 92–109. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-14458-5_6. (Cited on page 15.)

[Elsborg *et al.* 2009] E. Elsborg, T. Hildebrandt and D. Sangiorgi. *Type systems for bigraphs*. Trustworthy Global Computing, pages 126–140, 2009. (Cited on page 201.)

[Emerson 1991] E.A. Emerson. *Temporal and modal logic*. In Handbook of theoretical computer science (vol. B), page 1072. MIT Press, 1991. (Cited on pages 108, 130 and 152.)

[European Commission 2007] European Commission. *ICT - Information and Communication Technologies*. Work Programme 2007–2008, 2007. (Cited on page 1.)

[Fábregas *et al.* 2010] Ignacio Fábregas, David de Frutos Escrig and Miguel Palomino. *Logics for Contravariant Simulations*. In Formal Techniques for Distributed Systems: Joint 12th IFIP WG 6.1 International Conference, FMOODS 2010 and 30th IFIP WG 6.1 International Conference, FORTE 2010, Amsterdam, The Netherlands, June 7-9, 2010, Proceedings, volume 6117, page 224. Springer-Verlag New York, 2010. (Cited on pages 53 and 193.)

[Fantechi *et al.* 2008] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese and F. Tiezzi. *A model checking approach for verifying COWS specifications*. In Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering, pages 230–245. Springer-Verlag, 2008. (Cited on page 15.)

[Ferreira *et al.* 2010] Carla Ferreira, Ivan Lanese, António Ravara, Hugo Torres Vieira and Gianluigi Zavattaro. *Advanced Mechanisms for Service Combination and Transactions*. In Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA project on Software Engineering for Service-Oriented Computing, LNCS, page 25 pages. Springer, 2010. submitted. (Cited on pages 15, 162 and 182.)

[Field & Kathleen N. Lohr 1990] Marilyn J. Field and Institute of Medicine Kathleen N. Lohr Editors; Committee to Advise the Public Health Service on Clinical Practice Guidelines. Clinical practice guidelines:directions for a new program. The National Academies Press, 1990. (Cited on page 6.)

[Fiore & Abadi 2001] Marcelo Fiore and Martin Abadi. *Computing symbolic models for verifying cryptographic protocols*. Proc. 14th IEEE Computer Security Foundations Workshop, pages 160–173, 2001. (Cited on page 80.)

[Fiore *et al.* 1999] Marcelo Fiore, Gianluca. Cattani and Glynn Winskel. *Weak bisimulation and open maps*. In 14th Symposium on Logic in Computer Science (LICS), pages 67–76. IEEE, 1999. (Cited on pages 197 and 202.)

[Gordon & Fournet 2009] A.D. Gordon and C. Fournet. *Principles and applications of refinement types*. Technical Report MSR–TR–2009–147, Microsoft Research, Cambridge, UK, October 2009. (Cited on pages 18, 19, 153, 154 and 201.)

[Gordon & Jeffrey 2003] A.D. Gordon and A. Jeffrey. *Typing correspondence assertions for communication protocols*. Theoretical Computer Science, vol. 300, no. 1-3, pages 379–409, 2003. (Cited on pages 18 and 153.)

[Guidi *et al.* 2006] Claudio Guidi, Roberto Lucchi, Roberto Gorrieri, Nadia Busi and Gianluigi Zavattaro. *SOCK: A Calculus for Service Oriented Computing*. In Asit Dan and Winfried Lamersdorf, editors, ICSOC, volume 4294 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006. (Cited on page 14.)

[Harel & Thiagarajan 2004] D. Harel and P. Thiagarajan. *Message sequence charts*. UML for Real, pages 77–105, 2004. (Cited on page 21.)

[Heinl *et al.* 1999] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein and M. Teschke. *A comprehensive approach to flexibility in workflow management systems*. ACM SIGSOFT Software Engineering Notes, vol. 24, no. 2, pages 79–88, 1999. (Cited on page 6.)

[Hennessy & Milner 1980] Matthew Hennessy and Robin Milner. *On Observing Nondeterminism and Concurrency*. In Proceedings of the 7th Colloquium on Automata, Languages and Programming, pages 299–309. Springer-Verlag London, UK, 1980. (Cited on pages 11 and 102.)

[Hennessy & Milner 1985] Matthew Hennessy and Robin Milner. *Algebraic laws for nondeterminism and concurrency*. Journal of the ACM (JACM), vol. 32, no. 1, pages 137–161, 1985. (Cited on page 31.)

[Hennessy 2007] Matthew Hennessy. A distributed pi-calculus. Cambridge Univ Press, 2007. (Cited on pages 40 and 136.)

[Hildebrandt & López 2009] Thomas Hildebrandt and Hugo A. López. *Types for Secure Pattern Matching with Local Knowledge in Universal Concurrent Constraint Programming* . In 25th International Conference on Logic Programming (ICLP), volume 5649 of *Lecture Notes in Computer Science*, pages 417–431. Springer, Berlin Heidelberg, 2009. (Cited on pages 12 and 23.)

[Hildebrandt & Mukkamala 2010] 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*, page 59, 2010. (Cited on pages 16, 194 and 197.)

[Hildebrandt *et al.* 2011] Thomas Hildebrandt, Raghava Mukkamala and Tijs Slaats. *Safe Distribution of Declarative Processes*. In Gilles Barthe, Alberto Pardo and Gerardo Schneider, editors, Software Engineering and Formal Methods, volume 7041 of *Lecture Notes in Computer Science*, pages 237–252. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-24690-6_17. (Cited on pages 16, 194 and 197.)

[Hoare 1983] C. A. R. Hoare. *Communicating Sequential Processes*. Commun. ACM, vol. 26, no. 1, pages 100–106, 1983. (Cited on page 21.)

[Højsgaard & Hallwyl 2012] Espen Højsgaard and Tim Hallwyl. *Core BPEL: Syntactic Simplification of WS-BPEL 2.0*. In Proceedings of the 27th ACM Symposium on Applied Computing (SAC'12)., 2012. (Cited on page 7.)

[Honda *et al.* 1998] Kohei Honda, Vasco T. Vasconcelos and Makoto Kubo. *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In 7th European Symposium on Programming (ESOP): Programming Languages and Systems, pages 122–138. Springer-Verlag London, UK, 1998. (Cited on pages 4, 10, 11, 15, 17, 18, 21, 36, 37, 40, 42, 45, 57, 59, 61, 67, 76, 106, 120, 122, 128, 136, 139, 153, 166 and 182.)

[Honda *et al.* 2008] Kohei Honda, Nobuko Yoshida and Marco Carbone. *Multiparty asynchronous session types*. In George C. Necula and Philip Wadler, editors, POPL, pages 273–284. ACM, 2008. (Cited on pages 19 and 154.)

[Hongli *et al.* 2007] Yang Hongli, Zhao Xiangpeng, Cai Chao and Qiu Zongyan. *Exploring the Connection of Choreography and Orchestration with Exception Handling and Finalization/Compensation*. In John Derrick and Jüri Vain, editors, Formal Techniques for Networked and Distributed Systems – FORTE 2007, volume 4574 of *Lecture Notes in Computer Science*, pages 81–96. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73196-2_6. (Cited on pages 5, 15, 120 and 182.)

[Hüttel 2011] Hans Hüttel. *Typed Ψ-calculi*. In Joost-Pieter Katoen and Barbara König, editors, CONCUR 2011 – Concurrency Theory, volume 6901 of *Lecture Notes in Computer Science*, pages 265–279. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-23217-6_18. (Cited on page 201.)

[Jensen 1994] Kurt Jensen. *An Introduction to the Theoretical Aspects of Coloured Petri Nets*. In A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, pages 230–272, London, UK, 1994. Springer-Verlag. (Cited on page 16.)

[Joyal *et al.* 1993] A. Joyal, M. Nielson and G. Winskel. *Bisimulation and open maps*. In Eighth Annual IEEE Symposium on Logic in Computer Science (LICS), pages 418–427. IEEE, 1993. (Cited on pages 29, 197 and 202.)

[Kavantzas *et al.* 2004] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon and C. Barreto. *Web services choreography description language version 1.0*. W3C Working Draft, vol. 17, pages 10–20041217, 2004. (Cited on pages 5, 9, 38, 103, 120 and 124.)

[Keller 1976] R.M. Keller. *Formal verification of parallel programs*. Communications of the ACM, vol. 19, no. 7, pages 371–384, 1976. (Cited on page 29.)

[Kitchin *et al.* 2009] D. Kitchin, A. Quark, W. Cook and J. Misra. *The Orc programming language*. Formal techniques for Distributed Systems, pages 1–25, 2009. (Cited on page 14.)

[La Rue 2011] F. La Rue. *Report of the Special Rapporteur on the promotion and protection of the right to freedom of opinion and expression*. United Nations General Assembly Human Rights Council, 2011. (Cited on page 1.)

[Lamport 1994] Leslie Lamport. *The Temporal Logic of Actions*. ACM Transactions on Programming Languages and Systems, vol. 16, no. 3, pages 872–923, May 1994. (Cited on page 8.)

[Lanese *et al.* 2007] I. Lanese, V.T. Vasconcelos, F. Martins and A. Ravara. *Disciplining Orchestration and Conversation in Service-Oriented Computing*. Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM'2007), pages 305–314, 2007. (Cited on pages 5, 14, 15 and 56.)

[Lanese *et al.* 2008] Ivan Lanese, Claudio Guidi, Fabrizio Montesi and Gianluigi Zavattaro. *Bridging the Gap between Interaction- and Process-Oriented Choreographies*. In Antonio Cerone and Stefan Gruner, editors, SEFM, pages 323–332. IEEE Computer Society, 2008. (Cited on page 5.)

[Lanese *et al.* 2010] Ivan Lanese, Cátia Vaz and Carla Ferreira. *On the Expressive Power of Primitives for Compensation Handling*. In Proc. of ESOP, volume 6012 of *Lecture Notes in Computer Science*, pages 366–386. Springer, 2010. (Cited on page 184.)

[Laneve & Padovani 2007] Cosimo Laneve and Luca Padovani. *The must preorder revisited*. CONCUR 2007–Concurrency Theory, pages 212–225, 2007. (Cited on page 53.)

[Laneve & Padovani 2008] Cosimo Laneve and Luca Padovani. *The Pairing of Contracts and Session Types*. In Pierpaolo Degano, Rocco De Nicola and José Meseguer, editors, Concurrency, Graphs and Models, volume 5065 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2008. (Cited on page 20.)

[Laneve & Zavattaro 2005] Cosimo Laneve and Gianluigi Zavattaro. *Foundations of Web Transactions*. In Vladimiro Sassone, editor, Foundations of Software Science and Computational Structures, volume 3441 of *LNCS*, pages 282–298. Springer, Berlin Heidelberg, 2005. (Cited on pages 15 and 183.)

[Lapadula *et al.* 2007a] Alessandro Lapadula, Rosario Pugliese and Francesco Tiezzi. *A calculus for orchestration of web services*. In Proc. of 16th European Symposium on Programming (ESOP'07), volume 4421 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2007. (Cited on pages 5, 14, 15, 56, 120, 122 and 182.)

[Lapadula *et al.* 2007b] Alessandro Lapadula, Rosario Pugliese and Francesco Tiezzi. *COWS: a timed service-oriented calculus*. In Proceedings of the 4th international conference on Theoretical aspects of computing, pages 275–290. Springer-Verlag, 2007. (Cited on pages 15 and 182.)

[Lapadula *et al.* 2008] Alessandro Lapadula, Rosario Pugliese and Francesco Tiezzi. *A Formal Account of WS-BPEL*. In Doug Lea and Gianluigi Zavattaro, editors, COORDINATION, volume 5052 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2008. (Cited on page 14.)

[Larsen & Thomsen 1988] Kim Guldstrand Larsen and Bent Thomsen. *A Modal Process Logic*. In 3rd. Annual Symposium on Logic in Computer Science, 5-8 July, pages 203–210, Edinburgh, Scotland, UK, 1988. IEEE Computer Society. (Cited on page 194.)

[Larsen *et al.* 2007] Kim Guldstrand Larsen, Ulrik Nyman and Andrzej Wasowski. *On Modal Refinement and Consistency*. In Luís Caires and Vasco Thudichum Vasconcelos, editors, CONCUR, volume 4703 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2007. (Cited on page 191.)

[López & Pérez 2011] Hugo A. López and Jorge A. Pérez. *Timed, Compensable Conversations*. In 4th Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), 2011. (Cited on page 23.)

[López & Pérez 2012] Hugo A. López and Jorge A. Pérez. *Time and Exceptional Behavior in Multiparty Structured Communications*. In Marco Carbone and Jean-Marc Petit, editors, Web Services and Formal Methods (WS-FM), volume (To appear) of *Lecture Notes in Computer Science*. Springer, 2012. (Cited on pages 11, 12 and 23.)

[López *et al.* 2006] Hugo A. López, Jorge A. Pérez, Catuscia Palamidessi, Camilo Rueda and Frank D. Valencia. *A Declarative Framework for Security: Secure Concurrent Constraint Programming*. In Sandro Etalle and M. Truszczyński, editors, 22th International Conference on Logic Programming (ICLP'06), volume 4079 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 2006. (Cited on page 80.)

[López *et al.* 2009] Hugo A. López, Fabio Massacci and Nicola Zannone. *Goal-Equivalent Secure Business Process Re-engineering*. In E. Di Nitto and M. Ripeanu, editors, Service-Oriented Computing - ICSOC 2007 Workshops, volume

4907 of *LNCS*, pages 212—223, Berlin, Heidelberg, January 2009. Springer - Verlag. (Cited on pages 24, 173 and 174.)

[López *et al.* 2010] Hugo A. López, Carlos Olarte and Jorge A. Pérez. *Towards a Unified Framework for Declarative Structured Communications*. In 2nd Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), volume 17 of *EPTCS*, pages 1–15, 2010. (Cited on pages 11, 23, 106, 118, 128 and 182.)

[López 2010] Hugo A. López. *Models for Trustworthy Service and Process Oriented Systems*. In 26th International Conference on Logic Programming (ICLP), volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 270–276, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (Cited on page 23.)

[Lowe 1995] Gavin Lowe. *An attack on the Needham-Schroeder public-key authentication protocol*. Inf. Process. Lett., vol. 56, no. 3, pages 131–133, 1995. (Cited on page 96.)

[Lyng *et al.* 2008] Karen Marie Lyng, Thomas Hildebrandt and Raghava Rao Mukkamala. *The Resultmaker Online Consultant: From Declarative Workflow Management in Practice to LTL*. In Proc. of 1st Intl. Workshop on Dynamic and Declarative Business Processes (DDBP), Munich, Germany, 2008. (Cited on pages 6, 16 and 121.)

[Lyng *et al.* 2009] Karen Marie Lyng, Thomas T. Hildebrandt and Rao Mukkamala. *From paper based clinical practice guidelines to declarative workflow management*. In Business Process Management Workshops, pages 336–347. Springer, 2009. (Cited on page 162.)

[Manna & Pnueli 1992] Zohar Manna and Amir Pnueli. The temporal logic of reactive and concurrent systems: Specification. Springer, 1992. (Cited on pages 6, 35, 45, 62 and 64.)

[Miller 2003] Dale Miller. *Encryption as an abstract data type: An Extended Abstract*. In Foundations of Computer Security (FCS), volume 84 of *Electronic Notes in Theoretical Computer Science*, pages 3–15. Springer, Heidelberg, 2003. (Cited on pages 80 and 100.)

[Milner *et al.* 1992] Robin Milner, Joachim Parrow and David Walker. *A Calculus of Mobile Processes, Parts I and II*. Journal of Information and Computation, vol. 100, pages 1–77, September 1992. (Cited on pages 10, 84 and 162.)

[Milner 1991] Robin Milner. *The Polyadic Pi Calculus: A Tutorial*. Technical Report 100(1), Information and Computation, 1991. (Cited on page 17.)

[Milner 1995] Robin Milner. Communication and concurrency. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995. (Cited on page 27.)

[Milner 1999] Robin Milner. Communicating and Mobile systems. the Pi Calculus. Cambridge University Press, 1999. (Cited on pages 27, 40, 52, 106, 127, 136 and 180.)

[Misra & Cook 2006] Jayadev Misra and William R. Cook. *Computation Orchestration: A Basis for Wide-Area Computing*. Journal of Software and Systems Modeling, May 2006. (Cited on pages 5, 13 and 120.)

[Montangero & Semini 2006] Carlo Montangero and Laura Semini. *A Logical View of Choreography*. In Coordination models and languages: 8th international conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006: proceedings, volume 4038 of *Lecture Notes in Computer Science*, pages 179–193. Springer-Verlag New York, 2006. (Cited on pages 5, 16 and 118.)

[Montesi *et al.* 2007] Fabrizio Montesi, Claudio Guidi and Gianluigi Zavattaro. *Composing Services with JOLIE*. In ECOWS, pages 13–22. IEEE Computer Society, 2007. (Cited on page 15.)

[Muehlen & Recker 2008] M. Muehlen and J. Recker. *How much language is enough? Theoretical and practical use of the business process modeling notation*. In Advanced Information Systems Engineering, pages 465–479. Springer, 2008. (Cited on page 7.)

[Nielsen *et al.* 2002] Mogens Nielsen, Catuscia Palamidessi and Frank Valencia. *Temporal Concurrent Constraint Programming: Denotation, Logic and Applications*. Nordic J. of Computing, 2002. (Cited on pages 56, 57, 62 and 76.)

[Nørgaard *et al.* 2005] A. K. Nørgaard, L. Pedersen and P. Strøiman. *Method for generating a workflow on a computer, and a computer system adapted for performing the method*. Patent, 05 2005. US 6895573. (Cited on pages 6 and 121.)

[Object Management Group 2011] Object Management Group. *Business Process Model and Notation (BPMN), Version 2.0*. Available at `http://www.omg.org/spec/BPMN/2.0/`, January 2011. (Cited on pages 3, 5 and 20.)

[Olarte & Valencia 2008a] C. A. Olarte and F. D. Valencia. *Universal Concurrent Constraint Programming: Symbolic Semantics and Applications to Security*. In 23rd Annual ACM Symposium on Applied Computing (SAC), 2008. (Cited on pages 10, 11, 32, 33, 34, 35, 57, 62, 65, 80, 81, 82, 84, 99 and 202.)

[Olarte & Valencia 2008b] Carlos Alberto Olarte and Frank D. Valencia. *The Expressivity of Universal Timed CCP*. In 10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, Valencia, Spain, July 2008. ACM Press, New York. (Cited on pages 35, 57, 62 and 93.)

[Ouyang *et al.* 2006] C. Ouyang, W.M.P. van der Aalst, M. Dumas and A.H.M. ter Hofstede. *Translating BPMN to BPEL*. BPM Center Report BPM-06-02, BPMcenter.org, 2006. (Cited on page 20.)

[Palamidessi *et al.* 2006] C. Palamidessi, V. Saraswat, F.D. Valencia and B. Victor. *On the Expressiveness of Linearity vs Persistence in the Asychronous Pi-Calculus*. In Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, pages 59–68. IEEE Computer Society Washington, DC, USA, 2006. (Cited on pages 100 and 202.)

[Pérez *et al.* 2012] Jorge A. Pérez, L. Caires, Frank Pfenning and Bernardo Toninho. *Linear Logical Relations for Session-Based Concurrency*. In European Symposium on Programming, 2012. To appear. (Cited on pages 18 and 153.)

[Pesic & van der Aalst 2006] M. Pesic and W.M.P. van der Aalst. *A Declarative Approach for Flexible Business Processes Management*. Lecture Notes in Computer Science, vol. 4103, page 169, 2006. (Cited on pages 6, 16, 56, 75, 76 and 121.)

[Pesic 2008] M. Pesic. *Constraint-based workflow management systems: shifting control to users*. PhD thesis, Technische Universiteit Eindhoven, 2008. (Cited on pages 16 and 17.)

[Peterson 1977] James L. Peterson. *Petri Nets*. ACM Comput. Surv., vol. 9, no. 3, pages 223–252, 1977. (Cited on page 16.)

[Pierce & Sangiorgi 1993] B. Pierce and D. Sangiorgi. *Typing and subtyping for mobile processes*. In Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on, pages 376–385. IEEE, 1993. (Cited on page 17.)

[Pitts 2003] A.M. Pitts. *Nominal logic, a first order theory of names and binding*. Information and computation, vol. 186, no. 2, pages 165–193, 2003. (Cited on page 203.)

[Plotkin 1981] G. D. Plotkin. *A Structural Approach to Operational Semantics*. Technical report, University of Aarhus, 1981. (Cited on pages 29, 103 and 125.)

[Pnueli 1977] Amir Pnueli. *The temporal logic of programs*. In 18th Annual Symposium on Foundations of Computer Science, volume 17, pages 46 –57. IEEE, 1977. (Cited on page 44.)

[Post 1944] Emil L. Post. *Recursively enumerable sets of positive integers and their decision problems*. Bulletin of the American Mathematical Society, vol. 50, pages 284–316, 1944. (Cited on page 111.)

[Prandi & Quaglia 2007] D. Prandi and P. Quaglia. *Stochastic cows*. Service-Oriented Computing–ICSOC 2007, pages 245–256, 2007. (Cited on page 15.)

[Prandi *et al.* 2008] Davide Prandi, Paola Quaglia and Nicola Zannone. *Formal analysis of BPMN via a translation into COWS*. In Coordination Models and Languages, pages 249–263. Springer, 2008. (Cited on page 21.)

[Puhlmann & Weske 2005] F. Puhlmann and M. Weske. *Using the Pi-Calculus for Formalizing Workflow Patterns*. BPM, vol. 3649, pages 153–168, 2005. (Cited on pages 17 and 120.)

[Puhlmann 2007] Frank Puhlmann. *On the Application of a Theory of Mobile Processes to Business Process Management*. PhD thesis, University of Postdam, Hasso Platner Institut, July 2007. (Cited on page 15.)

[Recker & Mendling 2006] J. Recker and J. Mendling. *On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages*. In The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium., pages 521–532, Luxembourg, Grand-Duchy of Luxembourg, 2006. Namur University Press. (Cited on page 21.)

[Reynolds 2002] JC Reynolds. *Separation logic: a logic for shared mutable data structures*. Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on, pages 55–74, 2002. (Cited on pages 108 and 130.)

[Rittenberger *et al.* 2006] Jon C Rittenberger, James E Bost and James J Menegazzi. *Time to give the first medication during resuscitation in out-of-hospital cardiac arrest*. Resuscitation, vol. 70, no. 2, pages 201–6, Aug 2006. (Cited on page 175.)

[Russell *et al.* 2009] Nick C. Russell, Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. *Designing a Workflow System Using Coloured Petri Nets*. T. Petri Nets and Other Models of Concurrency, vol. 3, pages 1–24, 2009. (Cited on page 17.)

[Rutten 2000] J. Rutten. *Coalgebra, concurrency, and control*. Discrete event systems: analysis and control, vol. 569, page 31, 2000. (Cited on pages 52, 53 and 196.)

[Rychkova *et al.* 2008] I. Rychkova, G. Regev and A. Wegmann. *Using declarative specifications in business process design*. International Journal of Computer Science and Applications, vol. 5, no. 3b, pages 45–68, 2008. (Cited on page 6.)

[Sandhu 1993] RS Sandhu. *Lattice-based access control models*. Computer, vol. 26, no. 11, pages 9–19, 1993. (Cited on page 93.)

[Sangiorgi & Walker 2001] Davide Sangiorgi and David Walker. Pi-calculus: A theory of mobile processes. Cambridge University Press, New York, NY, USA, 2001. (Cited on pages 27, 56 and 62.)

[Saraswat *et al.* 1994] Vijay A. Saraswat, Radha Jagadeesan and Vineet Gupta. *Foundations of timed concurrent constraint programming*. In Proceedings of the

Ninth Annual IEEE Symposium on Logic in Computer Science (LICS 1994), pages 71–80. IEEE Computer Society Press, July 1994. (Cited on pages 32, 56, 61 and 80.)

[Saraswat 1993] V. Saraswat. Concurrent Constraint Programming. MIT Press, 1993. (Cited on pages 10, 15, 32, 56, 57 and 62.)

[Smith 1990] Gary Wayne Smith. *The modeling and representation of security semantics for database applications*. PhD thesis, George Mason University, Faifax, VA, USA, 1990. UMI Order No. GAX90-25973. (Cited on page 93.)

[Su *et al.* 2007] J. Su, T. Bultan, X. Fu and X. Zhao. *Towards a theory of web service choreographies*. In WS-FM, volume 4937, pages 1–16. Springer, 2007. (Cited on page 5.)

[Takeuchi *et al.* 1994] Kaku Takeuchi, Kohei Honda and Makoto Kubo. *An Interaction-based Language and its Typing System*. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou and Sergios Theodoridis, editors, PARLE, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994. (Cited on page 17.)

[Terenziani *et al.* 2000] P. Terenziani, F. Mastromonaco, G. Molino and M. Torchio. *Executing clinical guidelines: temporal issues*. In Proceedings of the AMIA Symposium, page 848. American Medical Informatics Association, 2000. (Cited on page 177.)

[Thatte 2001] S. Thatte. *XLANG: web services for business process design, 2001*. Microsoft `http://www.gotdotnet.com/team/xml-wspecs/xlang-cl/default.htm`, 2001. (Cited on page 14.)

[Valencia 2002] Frank D. Valencia. *Temporal Concurrent Constraint Programming*. PhD thesis, University of Aarhus, November 2002. (Cited on page 98.)

[van der Aalst & Pesic 2006] W.M.P. van der Aalst and M. Pesic. *DecSerFlow: Towards a Truly Declarative Service Flow Language*. Lecture Notes in Computer Science, vol. 4184, page 1, 2006. (Cited on pages 6, 16, 56 and 121.)

[Van Der Aalst & Ter Hofstede 2005] W.M.P. Van Der Aalst and A.H.M. Ter Hofstede. *YAWL: yet another workflow language*. Information Systems, vol. 30, no. 4, pages 245–275, 2005. (Cited on page 17.)

[van Der Aalst *et al.* 2003] W.M.P. van Der Aalst, A.H.M. Ter Hofstede, B. Kiepuszewski and A.P. Barros. *Workflow patterns*. Distributed and parallel databases, vol. 14, no. 1, pages 5–51, 2003. (Cited on page 17.)

[van der Aalst *et al.* 2009] Wil M. P. van der Aalst, Maja Pesic and Helen Schonenberg. *Declarative workflows: Balancing between flexibility and support*. Computer Science - R&D, vol. 23, no. 2, pages 99–113, 2009. (Cited on page 16.)

[van der Aalst 1998] W.M.P. van der Aalst. *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, vol. 8, no. 1, pages 21–66, 1998. (Cited on pages 17 and 56.)

[van der Aalst 2003] W. M. P. van der Aalst. *Challenges in business process management: Verification of business processes using Petri nets*. Bulletin of the EATCS, vol. 80, pages 174–199, 2003. European Association for Theoretical Computer Science (EATCS). (Cited on page 17.)

[van Glabbeek 1990] R. J. van Glabbeek. *The linear time-branching time spectrum*. In Proceedings of the Theories of Concurrency: Unification and Extension, pages 278—297, 1990. (Cited on page 52.)

[van Riemsdijk & Wirsing 2007] M.B. van Riemsdijk and M. Wirsing. *Using Goals for Flexible Service Orchestration*. LNCS, vol. 4504, page 31, 2007. (Cited on page 5.)

[Victor & Parrow 1998] Björn Victor and Joachim Parrow. *Concurrent Constraints in the Fusion Calculus*. In Proc. of ICALP, volume 1443 of *LNCS*, pages 455–469. Springer, 1998. (Cited on pages 20 and 59.)

[Vieira *et al.* 2008] H.T. Vieira, L. Caires and J.C. Seco. *The Conversation Calculus: A Model of Service-Oriented Computation*. In Programming languages and systems: 17th European Symposium on Programming (ESOP), page 269, Budapest, Hungary, 2008. Springer-Verlag New York. (Cited on pages 10, 11, 42, 120, 162, 166, 182 and 184.)

[Vieira 2010] Hugo T. Vieira. *A Calculus for Modeling and Analyzing Conversations in Service-Oriented Computing*. PhD thesis, Universidade Nova de Lisboa, 2010. (Cited on pages 5, 42, 162, 163, 166, 167 and 169.)

[Wehrman *et al.* 2008] Ian Wehrman, David Kitchin, William R. Cook and Jayadev Misra. *A timed semantics of Orc*. Theor. Comput. Sci., vol. 402, no. 2-3, pages 234–248, 2008. (Cited on pages 14, 15 and 182.)

[Winskel 2005] G. Winskel. *Relations in Concurrency*. In 20th Annual IEEE Symposium on Logic in Computer Science (LICS), pages 2–11, 2005. (Cited on pages 197 and 202.)

[Wong 2010] P.Y.H. Wong. *Formalisations and Applications of Business Process Modelling Notation*. PhD thesis, University of Oxford, 2010. (Cited on page 21.)

[Xu *et al.* 1998] Jie Xu, Alexander B. Romanovsky and Brian Randell. *Coordinated Exception Handling in Distributed Object Systems: From Model to System Implementation*. In ICDCS, pages 12–21, 1998. (Cited on page 172.)