An Experiment on Creating Enterprise Specific BPM Languages and Tools

Steen Brahe

IT University Technical Report Series **TR-2008-102**

ISSN 1600–6100 **March 2008**

ISSN 1600–6100

ISBN 978-87-7949-161-8

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web `www.itu.dk`

An Experiment on Creating Enterprise Specific BPM Languages and Tools

Steen Brahe
Danske Bank and IT University of Copenhagen
Holmens Kanal 2-12
1092 Copenhagen K, Denmark
stbr@danskebank.dk

ABSTRACT

Many enterprises use their own domain concepts in modeling business process and use technology in specialized ways when they implement them in a Business Process Management (BPM) system. In contrast, BPM tools used for modeling and implementing business processes often provide a standard modeling language, a standard implementation technology and a fixed transformation that may generate the implementation from the model. This makes the tools inflexible and difficult to use. This paper presents another approach. It applies the basic model driven development principles of direct representation and automation to BPM tools through a tool experiment in Danske Bank, a large financial institute; We develop business process modeling languages, tools and transformations that capture Danske Banks specific modeling concepts and use of technology, and which automate the generation of code. An empirical evaluation shows that Danske Bank will possibly gain remarkable improvements in development productivity and the quality of the implemented code. This leads us to the conclusion that BPM tools should provide flexibility to allow customization of languages, tools and transformations to the specific needs of an enterprise.

1. INTRODUCTION

Business Process Management (BPM) is currently receiving much focus from the industry. Top management demands to understand and control their business processes and agility to adjust them when market conditions change. This can be achieved through Process Aware Information Systems (Dumas et al. (2005)). A Workflow Management (WFM) system (Jablonski and Bussler (1996); Leymann and Roller (2000)) is one example of such a system. A WFM system allows execution and automation of a business process that can be described explicitly as an executable workflow.

Although the hype about BPM and process automation is high, reality has shown that it is relatively complex to understand, model and implement a business process as an executable workflow (Brahe (2007)). First the process must be understood, second it must be formalized and modeled at a highly conceptual and logical level, and third the process design must be transferred to technology.

Two different worlds are meeting: Business and IT. Transferring business requirements to an IT solution is challenging. For business processes this is especially clear as the IT solution should actually execute the business process. Hence, it is crucial to transfer the business requirements and knowledge to the IT development team in a precise manner and with a certainty that the requirements will be implemented.

Many software vendors have complete BPM tool suites for modeling and implementing business processes. Such tools are mostly based on a predefined process modeling language like BPMN (White (2006)) for capturing the business process at the conceptual level and one technology like BPEL (BPEL (2003)) for implementing the process. These tools also assume a fixed development process where only two models exist, i.e. the conceptual business process and the implementation.

Using such tools causes two challenges for an enterprise that has specific requirements to its development process, uses its own modeling concepts and uses technology in specialized ways; First, a standardized modeling notation does not allow users to use domain concepts and may contain too many modeling constructs which makes the tool difficult to use. The models may also be difficult to understand and use as a communication media. Second, transformation of a model into implementation must be done manually as the enterprise may use a variety of technologies to implement the process and not only e.g. BPEL as many state-of-the-art tools support today. Even if one technology as e.g. BPEL is used, the enterprise may be using domain specific implementation patterns which cannot be generated because the transformations are hard-coded into the BPM tools.

The approach behind current BPM tools is similar to the extinct Computer Aided Software Engineering (CASE) tools from the 90es. They also often used a standard modeling language, one implementation technology and a standardized transformation. Their limited flexibility in supporting enterprise specific standards was one of the reasons why they were never accepted (Windsor (1986); Flynn et al. (1995)).

This paper takes another approach than state-of-the-art BPM tools. In order to avoid the CASE trap we must come up with an approach that allows an enterprise to use its own modeling notations and specific use of technology. Our hypothesis is that this can be achieved through applying the

basic model driven development (Stahl et al. (2006)) principles of direct representation and automation (Booch et al. (2004)) to BPM tools; An enterprise should be able to model its business processes directly in enterprise specific concepts, decide on a target platform and write transformations that encapsulate its specific use of technology, and that automate the generation of code.

This leads us to the research question which we will answer through this paper: *Does an enterprise specific BPM tool improve the efficiency and quality of modeling and implementing business processes, how difficult is it to create, and is it worth the effort?*.

We will through an experiment develop BPM languages, tools and transformations for a specific enterprise and evaluate advantages and drawbacks of this approach. We use Danske Bank, the second largest financial institute in northern Europe, as a case study. In lack of sufficient industrial standards, Danske Bank has defined its own development process and uses a number of different tools to support it. This has cause several challenges as described by Brahe (2007).

A prototype tool was developed to show that it is possible and provide value to develop BPM tools fitted for the needs of a specific enterprise. The prototype illustrates that it is possible to do model driven development of a business process with nearly 100% code generation. The prototype is fitted specially for Danske Banks development process and consists of three different Domain Specific Languages (DSLs) (Mernik et al. (2005)) and corresponding editors that are used to model a business process and related information. Furthermore, the tool provides transformations between the DSLs and a transformation to BPEL. These transformations capture implementation patterns specific for Danske Banks modeling standards and use of the implementation technology. Manual changes can be introduced into the generated BPEL code by a persistence utility feature.

The prototype tool is used in three steps; First, the business process is modeled at the conceptual level.

Second, the physical specification, i.e. information required to implement the model as a workflow, is specified. This includes documentation, technical attributes and required additional functionality.

Third, the conceptual model and physical specification are merged and transformed directly to executable BPEL and WDSL documents where only mapping of data and control flow logic need to be specified. A developer can regenerate the BPEL code completely without loosing the manual changes previously introduced when the logical business process model has been changed.

We use a fictitious project called Customer Quick Loan throughout the paper. First, we illustrate the current development process in Danske Bank and the observed challenges of using current BPM tools. Second, we show how the prototype tool eliminates these challenges. We conclude that BPM tools customized to a specific enterprise potentially have a huge effect on the efficiency of a project team and will result in implementations with fewer errors.

However, we also conclude that developing BPM tools from scratch requires high expertice and much effort and is a strategic decision that many enterprises will not take. What we need is a tool based framework that allows the enterprise to *customize* languages, transformations and tools to their specific needs instead of *creating* them from scratch. Such a framework has been described by Brahe and Østerbye (2006) and Brahe and Bordbar (2006). Implementation and utilization of this framework in commercial tools would allow flexibility and customizability for an enterprise like Danske Bank and it would still offer standardized modeling tools to those who prefer this.

The rest of the paper is organized as follows. Section 2 introduces Danske Bank and its development process at a high conceptual level. Section 3 describes the fictitious Customer Quick Loan project and illustrates how the project team develops modeling artifacts during the development process. In section 4 we discuss challenges regarding the development process and used modeling tools. Section 5 abstracts the development process into metamodels and algorithms for transforming models into code. Domain concepts and necessary information required to implement a business process using Danske Banks infrastructure are formalized. Based on this abstraction, the developed tool suite is described in section 6 and the example is used to illustrate how the customized tools make the project team more efficient. Section 7 describes an empirical evaluation of the tool. Section 8 discusses the results, section 9 contains related work and section 10 summarizes the report and outlines future work.

2. CASE STUDY: DANSKE BANK

This section introduces Danske Bank and describes how the enterprise models and implements business processes from understanding current work practice over high level modeling down to implementation of executable workflows.

2.1 Danske Bank

Danske Bank has grown to become the largest financial group in Denmark - and one of the largest in northern Europe. It comprises a variety of financial services such as banking, mortgage credit, insurance, pension, capital management, leasing and real estate agency.

To support and fulfill its IT strategy, Danske Bank has adopted a Service Oriented Architecture (SOA) at which all new system development is targeted and where existing legacy systems are service enabled. Applications and services developed for one part of the group can through a central service library and repository be located and used by other parts of the group. As Danske Bank started out implementing SOA before the web services standard was defined, it has developed its own proprietary standard for service specifications. Although proprietary, the standard is web service compliant. Currently, the enterprise has several thousand different service operations.

Support for executing and automating business processes can be achieved through different process execution engines. One of them is is batch execution of process implementations in PL1 and COBOL. Another one a BPM system from IBM, where the business processes are implemented using BPEL (BPEL (2003)). The BPM system has been extended in areas where business requirements were not fulfilled. For example, the enterprise has created its own Human Task Manager to handle and distribute human tasks that are part of an executable workflow and its own task portal where process participants claim and execute human tasks. Furthermore, it has specific uses of BPEL fault handlers andhas defined specific strategies of capturing business and technical faults during process execution. Compensation is not used and process instances are not allowed to fail. If an error occurs

during execution, the fault is caught by a fault handler, and the failing node is forced into a stopped state. A person or an application subsequently has to repair the failure and restart the failed node.

2.2 Business and IT Development Process

A part of the Danske Banks strategy is to use standardized methods, tools and technologies when they are considered mature and fulfills the enterprises requirements. However, no standardized development process as e.g. the Rational Unified Process (Kroll and Kruchten (2003)) was sufficient to fulfill Danske Banks requirements. Danske Bank therefore defined its own development process. Business and IT solutions are developed as one for any business problem. When a project team is established to address a business problem, all important stakeholders are represented including business process participants, business analyst, solution architects, system developers and test specialists. It is based on service oriented principles (Erl (2005)) where business requirements are mapped into required business services and processes.

Models are an important part of the development process. Most requirements and design decisions are captured in models. They are used for documentation of all important decisions in the project and for communicative purposes but they are also used as blueprints for the implementation. The development process includes specialized modeling notations and creation of different modeling artifacts.

Here we only look at the part of the development process that targets business process modeling and implementation although the development process also covers areas like functionality- and user interface development.

A project team works on a project through four stages; Requirements and functionality and process building blocks are identified in the *preanalysis* phase. In the first half of the *specification* phase, the project continues to identify and detail requirements, and logical solutions are modeled and described that fulfills the requirements. The logical design is mapped to physical artifacts, and information about concrete implementation technologies are specified during *physical design* which takes place in the last half of the specification phase. The systems are constructed in the *construction* phase and implemented in the organization in the *implementation* phase.

3. EXAMPLE: CUSTOMER QUICK LOAN

The fictitious project "Customer Quick Loan" will be used for illustrative purposes throughout the rest of the paper.

Changes in consumer patterns have required immediate action for introducing a new type of customer loans. The new loans can be requested from email and mobile phones with possible immediately approval and transfer of the requested amount to the customers account. A project team is established which includes a loan specialist, a business analyst, a solution architect, system developers and a project manager. They name the project *Customer Quick Loan*.

In the following we will see how the project team follows the enterprise specific development process to model requirements, design the solution, specify the physical design and implement it as an executable workflow.

3.1 Business Events

In the preanalysis phase, the business analyst defines all possible business events that may occur for a given business case. The events are described in a model called an event map. If an event will initiate another event, it is modeled as an arrow between them.

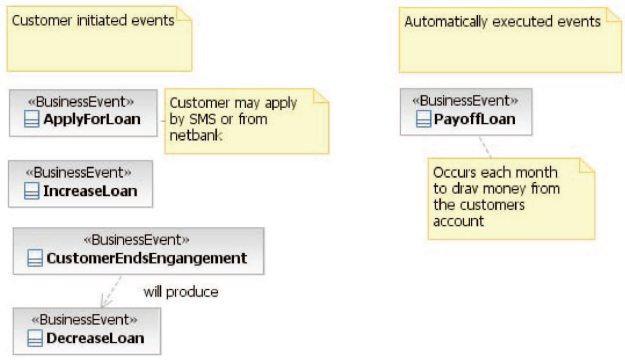


Figure 1: Business events for the Customer Quick Loan project modelled as a UML class diagram in Rational Software Modeler

For the Customer Quick Loan the primary events are ApplyForLoan, which is initiated when a customer requests a loan, and PayoffLoan, which occurs each month after the a loan has been created. The business analyst or the solution architect defines an event map as a UML class diagram in Rational Software Modeler (Swithinbank et al. (2005)). Figure 1 shows the event map and how events are organized manually in columns, which indicate different groups of events. The analyst decides on how to group the events. Textural nodes are used to describe scenarios. A simple UML profile has been applied to the model to allow modeling of business events using a stereotype. As a generic UML class diagram editor is used, the modeler first has to create a new class diagram, apply the relevant UML profile, model classes and then add the right stereotype to the classes. Additional information about the events is specified in MS Word documents. An event is classified as *external* if it is invoked by an actor as e.g. a customer or another department in the enterprise and it is classified as *timedependent* if the event is occurring at a certain point in time. The ApproveForLoan event is external as it is invoked by a customer while the Payoff loan is time dependent as it is invoked once a month. For time dependent events, the time of the occurrence must be specified as well as conditions that may initiate the event. An external event must have specified information about business possibilities and decisions that may initiate the event.

3.2 Solution Process Flow

Each business event, e.g. a customer requesting a loan, will involve execution of some business logic. For example, a back office worker or customer advisor that receives an email with a loan request from a customer will have to go through certain steps to address the request. For each business event, the business analyst will model such business logic as *asis* and *tobe* process models of current and possible future work practice. These models are made in cooperation with process participants who are involved in the real work practice. They are at a high conceptual level expressing creative thoughts about the business requirements. These

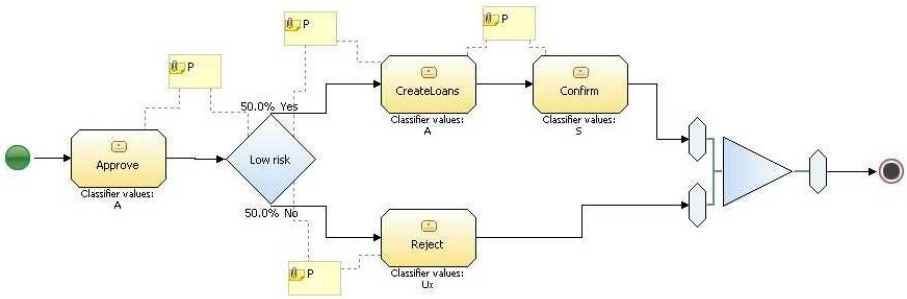


Figure 2: Solution Process Flow for the *ApproveForLoan* business event modeled in Websphere Business Modeler. MS Word is used to describe System Uses Cases for each task and for the process

models are only used for communicative means. Based on the creative *tobe* process, the business analyst together with a solution architect and possible a system developer model a precise business process at a logical level which expresses all logical steps in the business process including all possible exceptional cases. This model is called a Solution Process Flow (SPF). Each task in the SPF must either be of type *Automatic*, which means handled automatically by a service invocation, *Manual* as e.g. moving papers from a table to an archive or *UserExperiance* as e.g. creating a loan using an application user interface. Asis and tobe processes are defined in the preanalysis phase while Solution Process Flows are modeled in both the preanalysis phase and the first half of the specification phase. The IBM Websphere Business Modeler is used to define these models. The used task types are indicated by use of classifiers, a tooling concept that allows the modeler to classify tasks with textural descriptions.

The Solution Process Flow for the primary business event, *ApplyForLoan*, is illustrated in Figure 2. It consists of four logical tasks; first the automatic *ApproveLoan* task will make a risk calculation of the customer. If the risk is high, the loan request is rejected; process participant will be notified by the *Reject* task of type UserExperience, and will have to send an rejection message to the customer using an application interface. If the risk is low, the loan, or possible several loans applied for at once, will be created by the automatic *CreateLoans* tasks, and a confirmation will be sent to the customer by the *Confirm* task.

The project team has examined the local service repository for existing services and has found that two existing service operations called in a sequence will fulfill the requirement for a Confirm task in the Solution Process Flow. Therefore, the *Confirm* task is further broken down and modeled in a separate sub process as illustrated in Figure 3.

First, a service operation is invoked to create the content of the confirm message, and second, a service operation is invoked to send the message by SMS, email or letter.

Further from specifying the Solution Process Flow, the project team also details the description of each task in System Use Cases (SUC), one for the process and one for each task. A System Use Case describes the purpose of a task and is also used to link to other related models. For example, it will contain a link to a service contract model for an automatic task and for a user experience task it will contain a link to a Graphical User Interface model. For the ApplyForLoan process, five MS Word documents are defined. For

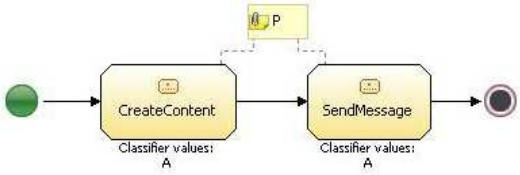


Figure 3: The Confirm task modeled as a sub process

instance, in the SUC for the *CreateLoans* task, it is written, that the task automatically creates all loans the customer has requested for. This is done by invoking a service operation multiple times. All models and descriptions from the specification phase are completely logical as it has not yet been decided which implementation technology to use.

3.3 Physical Design

The project team decides on how to implement the conceptual business process model during physical design. This takes place in the second half of the specification phase. Some business processes may be implemented in BPEL, others may be implemented in PL1 or COBOL for batch execution, and finally some processes may not be implemented by IT at all.

Although a WFM system allows automatic execution of the coordination of a business process, often the process is organizationally implemented with only partial IT support in form of graphical user interfaces or batch programs implemented. Actually, implementation of a business process using BPEL is only chosen for a small percentage of all modeled business processes.

However, this paper is about implementing business processes as executable workflows. The example as well as the prototype to be presented later only presents a Solution Process Flow that can be completely implemented as an executable workflow. In reality, it may be decided to implement only a part of a Solution Process Flow or merge several into one executable workflow. For the ApplyForLoan process, the project team decides to automate the execution of the process by implementing it in BPEL. Two kind of physical specifications now have to be made: BPEL process design, also called Control Flow Behavior, and a Workflow Specification, which contains additional information required to implement the Solution Process Flow and all its tasks in the WFM system.

3.4 Control Flow Behavior

The Solution Process Flow described in section 3.2 is the starting point for the Control Flow Behavior, a model of the physical implementation in BPEL which is created using Websphere Business Modeler. In cases where one SPF is completely implemented as a BPEL process, it will be quite similar to the Control Flow Behavior.

However, three physical design decisions make them different; First, the decision on implementing an SPF sub process in a separate BPEL process or as an inlined process flow in the main BPEL process. Second, the introduction of additional technological dependent functionality required by the BPEL implementation, and third, specification of implementation patterns which is how each task is implemented. We will discuss these physical design decisions in the next three sections:

3.4.1 Separate versus Inlined Subprocess

When designing a BPEL process it must be decided if the process should be implemented as one BPEL process, or if it should be broken down into several. In the case of the ApplyForLoan process, the developer decides that the *Confirm* sub process should be implemented as an inlined sub process in the ApplyForLoan BPEL process. This is illustrated in Figure 4. He or she could also have decided to implement the sub process as a separate BPEL process. Extracting parts of the process into sub processes has advantages: More than one developer can simultaneously work on the construction, it is easier to make a change and deploy a small sub process compared to change and deploy the main process, and a sub process can be reused by other processes. But extensively use of sub processes has the disadvantages of maintaining and operating several processes instead of one main process. This causes an overhead and introduces complexity regarding change management.

3.4.2 Technology Dependent Functionality

Additional functionality that is required for the technical implementation in a specific technology should not be modeled in the Solution Process Flow. In the case of BPEL, this could be complex data transformations inside in a BPEL process that are externalized as separate service invocations, it could be synchronization of data between different systems that make up the extended WFM system in Danske Bank, or it could be a specific service invocation that updates the business state for the specific process instance, a feature of the extended WFM system. For the ApplyForLoan process, an additional service invocation has to be inserted in the Control Flow Behavior after the AssessRisk service invocation to update the business state of the process instance. A service operation that is part of the WFM system must be invoked to set the business state to either "Approved" or "Rejected". This state information can be viewed by employees in the enterprise through the Human Task Portal that is part of the extended WFM system. If the Solution Process Flow had been implemented using another technology like COBOL, this additional functionality had not been required (see Figure 4).

3.4.3 Implementation Patterns

Each task in the Solution Process Flow has to be mapped to a task in the Control Flow Behavior. As this is a model of the implementation, each task must express how it should be implemented in BPEL, i.e. which implementation pattern to use. In this context implementation patterns are patterns, or code templates and rule used by Danske Bank to implement tasks of different types. An *Automatic* task type can be implemented by three different patterns; *ServiceOperation*, *MultipleInstances* and *Bundle*. Tasks of type *Manual* and *UserExperience* are always implemented using a *HumanTaskManual* or a *HumanTaskGUI* pattern. These patterns are illustrated in Table 2 and will be explained in section 3.6. When modeling the Control Flow Behavior, these pattern names are used to classify all tasks in the same way as the *Automatic*, *Manual* and *UserExperience* classifiers were used in the SPF model. Table 1 lists how tasks from the ApplyForLoan Solution Process Flow have been mapped to the Control Flow Behavior.

Table 1: Solution Process Flow tasks mapped into Control Flow Behavior

Task	Implementation pattern
AssessRisk	ServiceOperation
CreateLoans	Bundle
CreateContent	ServiceOperation
SendMessage	ServiceOperation
Reject	HumanTaskGUI

The implementation pattern to be used in the physical design is determined from the task type in the SPF and the description of the task in the corresponding System Use Case.

3.5 Workflow Specific Information

Much information has to be specified to implement the Control Flow Behavior in the BPM system. For a *ServiceOperation* task this includes information about which service operations to invoke, exception handling and escalation of errors, if the task must be restarted in case of failures during service invocation etc. It must also be specified what data is needed to invoke the operation and what data will be returned, as well as possible business and technical faults. Some of the required information has already been specified in other (functional) models such as which name of service operation to invoke, but to keep the example simple, we will provide this information below instead of introducing more models.

A tasks of type *HumanTaskManual* or *HumanTaskGUI* is a task handled by humans. Process participants will be able to list, claim and execute such a task from a task portal. For both type of tasks following information is needed; groups allowed to claim and execute a task are defined as Allocation Rules, labels, descriptions and data values in three to five different languages must be described to be presented to the business participants in the task portal, and rules about earliest start of the task and a possible deadline and several others also has to be specified. The *HumanTaskGUI* task further has a link to an existing application interface where the process participant has to handle the task. It must also be specified which data values from within the running BPEL process instance the link should transfer to the business system. For the process itself, additional information also has to be specified. This includes input data for the process, allocation rules, and description in several

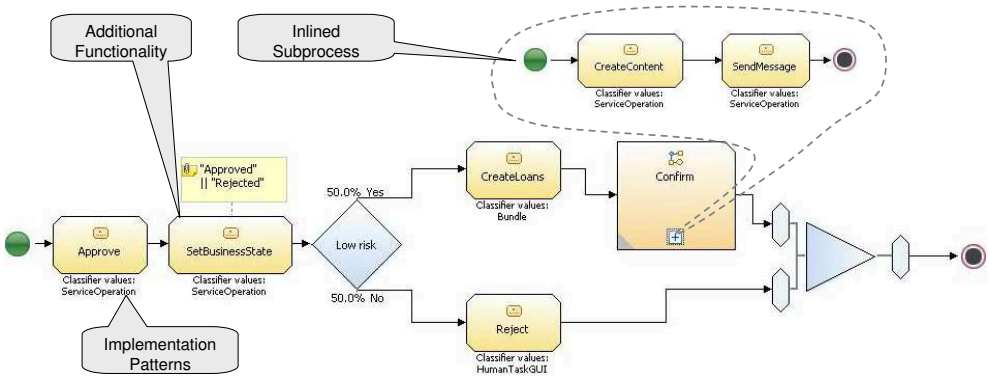


Figure 4: Controlflow behavior of the ApplyForLoan SPF

languages for presentation in the task portal.

All information for one task is specified in a MS Word document and is called a Workflow Task Specification. For each type a Word template is available for defining the required information. For the ApplyForLoan process six such task specifications are created.

3.6 BPEL Construction

After the Control Flow Behavior and Workflow Specification have been completed all required information and design decisions are available, and the BPEL process can be constructed.

A system developer now maps the Control Flow Behavior into a BPEL process. From the workflow specification that contains information about the controlflow, he is able to specify input/output data, set attributes about the process as e.g. when it is valid from, if it is a long running process etc. Also other systems as e.g. Danske Banks proprietary Human Task Manager can be populated with allocation rules specified in this document. The developer maps each task defined in the Control Flow Behavior to a BPEL implementation based on his knowledge of how BPEL is used in Danske Bank, i.e. the implementation patterns, and the information specified in the workflow specification documents. Each task type introduced in section 3.4.3 has a certain BPEL template and an algorithm for how to implement it. The pattern names and corresponding BPEL templates are illustrated in Table 2. Algorithms for how to populate these templates with data are described in section 5.

The Service Operation pattern invokes a service operation and incorporates specific way of using logging and fault handling. All service operations in Danske Bank throw a Technical Fault, which is caught by the fault handler for the Invoke node. The fault handler forces the invoke node into stopped state.

The MultipleInstances pattern is a loop containing a service invocation as implemented by the Service Operation pattern. It is similar to the workflow pattern "Multiple instances without priori runtime knowledge" (van der Aalst et al. (2003)).

The service operation invoked in the loop may initiate another process or thread that runs concurrently. For some business scenarios the main business process is not allowed to continue before all initiated processes behind these service invocations have finished. Danske Bank has extended the

Table 2: Danske Bank specific BPEL implementation patterns. The dots are replaced with information from the Control Flow Behavior and the workflow specification documents

Pattern name	BPEL template

ServiceOperation	`<assign name=..>..</assign>`
	`<invoke name=..>`
	`  <catch faultName=TechFault ..>`
	`  ..`
	`  </catch>`
	`</invoke>`

MultipleInstances	`<assign name=..>..</assign>`
	`<while name=..>`
	`  <assign name=..>`
	`  <invoke name=..>`
	`  <catch faultName=TechFault>`
	`  ..`
	`  </catch>`
	`  </invoke>`
	`</while>`

Bundle	`<assign name=..>..</assign>`
	`<while name=..>`
	`  <assign name=..>`
	`  <invoke name=..>`
	`  <catch faultName=TechFault>`
	`  ..`
	`  </catch>`
	`  </invoke>`
	`</while>`
	`<receive name=..>..</receive>`

HumanTask	`<assign name=..>..</assign>`
	`<invoke name=SetTaskInQueue>`
	`  <catch faultName=TechFault>`
	`  ..`
	`  </catch>`
	`</invoke>`

WFM system with infrastructure functionality that allows such a mechanism. In the BPEL process it is called the *Bundle* pattern and is implemented as the MultipleInstances pattern followed by an event. At runtime after invoking the service operation a number of times, the main process will wait until all the initiated concurrently running processes have finished. The WFM infrastructure extension will be notified about the state change and will fire the event that will cause the BPEL process to continue executing.

The HumanTaskManual and GUI patterns are implemented by invoking a specific service operation exposed by Danske Banks Human Task Manager followed by an event node. The translation of a task and its related information is purely manual, even though it is the same patterns that are implemented multiple times.

Above descriptions only show a subset of the implementation steps that the developer has to go through when implementing the tasks from the Control Flow Behavior. Com-

mon for all patterns is that data mapping also has to be specified before invoking a service operation.

Control flow logic also has to be specified by the developer. This is described at the edges that connect the tasks in the Control Flow Behavior model.

4. A NEED FOR CUSTOMIZED TOOLS

The models and information are developed and specified using a number of commercial modeling and implementation tools. The event map is modeled in Rational Software Modeler (RSM) using a UML class diagram with an applied profile. It is further detailed in a MS Word document. Business process models, i.e. asis, tobe and Solution Process Flows are modeled in Websphere Business Modeller (WBM) which uses an internal IBM notation. To support Danske Banks own modeling notation, nodes and connections are manually annotated with textual descriptions. WBM is also used to model Control Flow Behavior, but with other classifiers than for the conceptual business process models. System Use Cases and Workflow Specifications are described using Microsoft Word templates. As previously mentioned the SUC contains links to other models, but these links are purely textual. Websphere Integration Developer (WID) is used to implement Control Flow Behavior models in BPEL. The development team faces several challenges by using these different tools:

Difficult to use domain concepts Danske Bank has defined its own concepts for modeling business processes, but it is not possible to create models by directly using these concepts. Tools have been twisted and tweaked to force them to behave as desired. The usability is low and it is hard to use the models for communicative means. For instance, to model an automatic task a business analyst must first create a general task, view its properties and apply an "Automatic" classifier value. Visualization of classifiers is restricted to only textural representation.

Difficult to comprehend infomation A number of different tools are used to describe and specify how a business process should be implemented. The developer and the architect therefore need to look into several different tools and models to find relevant information. For example, for each service invocation the developer needs to locate the word document containing workflow specific information for the specific service operation read it through and re-enter the information into a WSDL document.

Missing traceability and consistency It is difficult to find relevant models because traceability between models is handled by textual descriptions. Furthermore, a model created in one tool cannot refer to models created in other tools. Consistency between models must therefore be handled manually. When changes appear to a Solution Process Flow, the architect must manually apply the changes to the Control Flow Behavior. The same challenge exists between the Control Flow Behavior and the BPEL implementation.

Imprecise data definitions in Word Imprecise data definitions can only be interpreted by humans. For instance, the name and version of the service operation is specified in a workflow specification document. The lack of a metamodel for this information makes it non-readable by machines. Therefore the developer must read the information manually and reenter it into a WSDL document.

Because of above challenges, transformation of models and information from specification and physical design into physical artifacts as e.g. BPEL have to be done purely manually. The system developer needs to open models in RSM and WBM and retrieve information manually, and he/she must open many MS Word document to get detailed information about design decisions. Although model driven development is the goal of the development process, the result is mere a document driven development process.

For the simple example of the ApproveForLoan business process, the number of models and documents get high even for a simple example with only four tasks. One RSM model, two WBM models and about 10 word documents make up the specification. It is quite difficult to comprehend the large amount of distributed information required for constructing the BPEL code. Further, the construction process is inefficient and error-prone as much of the information from the specification has to be manually reentered into the physical artifacts.

The core of the problem is that the commercial tools used presume one development process defined by the software vendor, a fixed set of modeling languages and a specific way to use the implementation technology. This is in deep contrast to the requirements from Danske Bank who found the standard development process and standard notations insufficient for their needs. They need to build their own development process into the tools, to use their own modeling notations and artifacts and to define their own use of technology.

In the next section we will make an abstraction of Danske Banks development process and the modeling artifacts, which will form the basis for the tool chain that has been developed specifically for Danske Bank. As we will observe later, the tool chain make the project team more productive as domain concepts are directly available within editors, traceability between models is provided, the use of Word has been replaced by models and the construction of the BPEL implementation is done automatically by customized transformations. This further reduces the risk for inconsistency between models and code, as information is taken directly from models without having to be reentered.

5. ABSTRACTED DEVELOPMENT PROCESS

Based on the Customer Quick Loan example described above we will now abstract the development process into metamodels and transformations. First, we give an overview of the current development process and describe requirements to a model driven development process. Second, we introduce the abstracted development process, which uses the metamodels and transformations that we will develop in this section. Last, we define these metamodels and describe algorithms of how to carry out the transformations. The metamodels and transformation algorithms form the basis for the prototype tool suite described in next section, which has been developed specific for Danske Bank.

5.1 Current Development Process

Figure 5 gives an overview of the described development process in previous section, and illustrates the created artifacts as well as design decisions. The artifacts are depictured with rounded boxes to indicate they are not precisely modeled, and the clouds indicate decisions that are not documented but instead put directly into models or code. Much

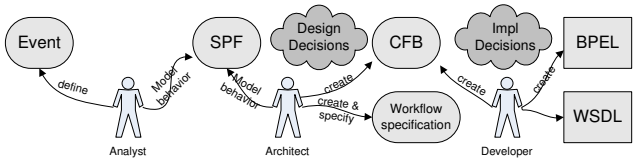


Figure 5: Current development process with main modeling artifacts and decision points. The clouds indicate that decisions are not documented, and the rounded boxes indicate that no metamodels are used.

of the information required though the development process is described as plain text. A human must read and interpret it to be able to construct the implementation. The cloud between the Solution Process Flow and the Control Flow Behavior illustrates that decisions about how the physical design are taken by the architect or developer; First, for each sub process modeled in the SPF it must be decided if it should be implemented as an inlined flow or as a separate process. Second, additional functionality must be specified. By defining the Control Flow Behavior from scratch, but inspired by the Solution Process Flow, the possibility to have tool based consistency check between the them is lost. The Control Flow Behavior model needs to be manually updated each time the Solution Process Flow changes. The cloud between the Control Flow Behavior model and the BPEL code indicates decision taken by the developer about BPEL specific information as e.g. the name of the project where the code is being developed, default package name, target namespace to use in the BPEL process, if generated WSDL files are kept in separate directories, etc.

5.2 Requirements to a Model Driven Development Process

One of the main ideas behind model driven development is to have tools that can transform platform independent models to platform specific models, and then generate the implementation code. In our example this means transformation of a Solution Process Flow into a Control Flow Behavior from which the BPEL implementation and related documents can be generated. In general, three basic requirements must be fulfilled to enable an efficient model driven development process:

1. Information and design decisions must be specified precisely in models.
2. Transformation between models must be formally described.
3. Information added to generated models or code must survive future transformations.

Creating precise models requires availability of languages or metamodels that support modeling standards and which allow modeling of all required information in a precise manner. As Danske Bank has created its own notations and use technology in specific ways, they need to be able to express this in their models.

5.3 Abstracted Development Process

Figure 6 illustrates the model driven development process that we will describe through the rest of this section. It uses metamodels, called Eventmap, SPF and WFSpec, for modeling event maps, Solution Process Flows and workflow specifications. It further uses a ModelInjection metamodel, and a BPELCodeGen metamodel. They are used to capture decisions currently taken in the "clouds". The metamodels form the basis for algorithms that can generate models and code. The BPELCodeGen metamodel is used to describe

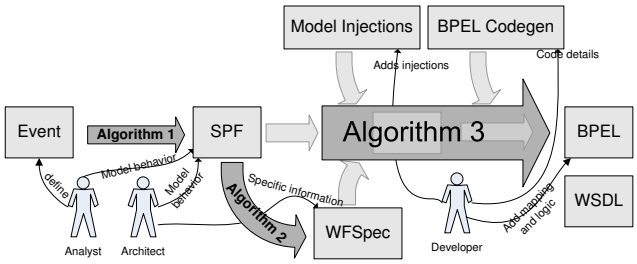


Figure 6: New development process with metamodels and transformations. Information is specified precisely by using the SPF, WFSpec, BPELCodeGen and ModelInjections metamodels

specific BPEL implementation decisions, while the WFSpec metamodel and the ModelInjection metamodel are used to describe the three differences between the Solution Process Flow and the Control Flow Behavior described in section 3.4:

1. Decisions about how to implement sub processes modeled as part of the Solution Process Flow is captured by the WFSpec metamodel. A WFSpec model is used to specify workflow specific information for each task in the Solution Process Flow. It also has an element to specify if a sub process should be implemented as an inlined BPEL flow or as a separate BPEL process.
2. Additional technical functionality is modeled as separate process fragments. Each process fragment is also called a model injection as it is to be injected at a specific point in the Solution Process Flow to generate the Control Flow Behavior. Process fragments are modeled using the Solution Process Flow metamodel. The relation between a process fragment and where to inject it is captured by a ModelInjection metamodel.
3. The implementation patterns to be used for implementing tasks in the Solution Process Flow are documented by the WFSpec metamodel, for instance that an *Automatic* task is implemented by the *ServiceOperation* or the *Bundle* implementation pattern.

The development process illustrated in Figure 6 has been implemented in a prototype tool that uses the metamodels to capture information precisely and transformations to automate the generation of the Control Flow Behavior and the BPEL implementation. After a business analyst has created an event in an event map, he or she uses Algorithm 1

to create a template for the corresponding Solution Process Flow. This template is used to for modeling the business process behavioral. The architect uses Algorithm 2 to generate the WFSpec model with default information, which he updates with correct information for the concrete business process. The architect and the developer may create a ModelInjection model to specify additional functionality to be injected when the implementation code is generated. Further, the developer creates a BPELCodegen model and defines attributes for the BPEL implementation. The developer uses these four models together with Algorithm 3 to generate the BPEL code. The transformation arrow for Algorithm 3 illustrates that the Control Flow Behavior model has disappeared as it is indirectly generated from the Solution Process Flow, the WFSpec model and the ModelInjection model. The prototype tool will be described in details in section 6

5.4 Metamodels

The five metamodels introduced above will now be described. They have been developed by analyzing the current development process. This includes discussions with development teams, enterprise architects and examination of educational material.

5.4.1 Eventmap Metamodel

The Eventmap metamodel, depictured in Figure 7 expresses how events can be modeled and related to each other. The metamodel has incorporated all information that previously was described as plain text in MS Word documents. As previously described, an event could be classified as external if it was initiate by an actor or internal if it was a time dependent event. Inheritance has been used to define these two event types and requirement for different information. An abstract *Event* metaclass contains attributes for information common for both types of events while the *TimedependentEvent* and the *ExternalEvent* subclasses contain specific attributes. Information has to be defined for the two event types. For the *TimedependentEvent*, time of occurrence must be defined as well as an optional condition for when the event can occur. For the *ExternalEvent*, information about business possibilities and occurrence has to be defined. Other information like priority and the responsible actor musts be defined for both types of events.

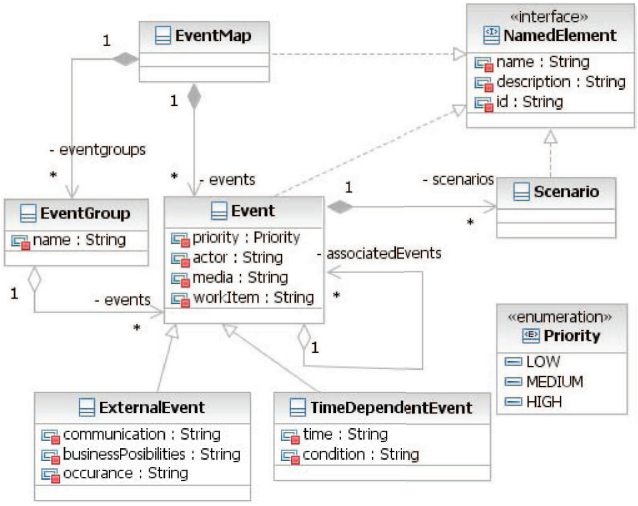


Figure 7: Event map (EventMap) metamodel. An event can either be external or timedependent and consists of a number of scenarios.

5.4.2 Solution Process Flow Metamodel

A Solution Process Flow is constructed for each event modeled in the event map. The SPF metamodel is illustrated in Figure 8. It is a simple flow based metamodel that reminds much of a UML activity diagram. The difference is the use of the domain specific task types, i.e. *Automatic*, *Manual* and *UserExperience*, and the domain specific edges, i.e. *ProcessConnection*, *DialogConnection* and *ProcessDialogConnection*. This paper and the implemented prototype only deals with modeling and transformation of an SPF model that can be completely transformed to an executable workflow, which means the it must only contain edges of type *ProcessConnection*. The other connection types can be used to model a process where part of it should be implemented in a dialog wizard framework instead of a WFM system.

5.4.3 Workflow Specification Metamodel

The Workflow Specification (WFSpec) metamodel, illustrated in Figure 9 is a formalization of the Workflow Specification previously defined in Word documents. A WFSpec model refers directly to a Solution Process Flow model instead of referring to a Control Flow Behavior model, as this is not explicitly modeled after the introduction of the Model Injection concept in section 5.3. Much information is required by the WFSpec metamodel, therefore only selected parts of it are described here. The *SPF4WFM* metaclass is the main element. It refers to a Solution Process Flow model and has several attributes specifying information required for implementing the BPEL process, e.g. a deadline rule, process lifetime information, allocation rules about process responsibility, department owner, process type etc. Many of these attributes are specific for Danske Bank as a BPEL process implemented in the WFM system is a part of a larger proprietary case system that extends the commercial BPM system with additional functionality. The *SPF4WFM* metaclass also specifies input and output data structures for the BPEL process. These data structures are modeled using the DataStructure metamodel which is described later. The reason to have a separate DataStructure metamodel is for reuse issues. In the prototype implementation it is used by the WFSpec metamodel as well as (a mock up of) Danske Banks service repository.

The *SPF4WFM* metaclass contains a number of TaskSpecification elements. A TaskSpecification can either be an *AutomaticSpecification*, *ManualSpecification*, *UserExperienceSpecification* or a *SubProcessSpecification*. An Element of one of these metaclasses refer to a task of type *Automatic*, *Manual*, *UserExperience* and *SubProcess* respectively. A TaskSpecification specifies required additional information for the implementation in BPEL and which implementation pattern to use. Previously, information about the implementation pattern was stored directly in the Control Flow Behavior while additional information was stored in Word documents.

The *SubProcessSpecification* metaclass is the simplest one, which only requires a type attribute specifying if the sub-

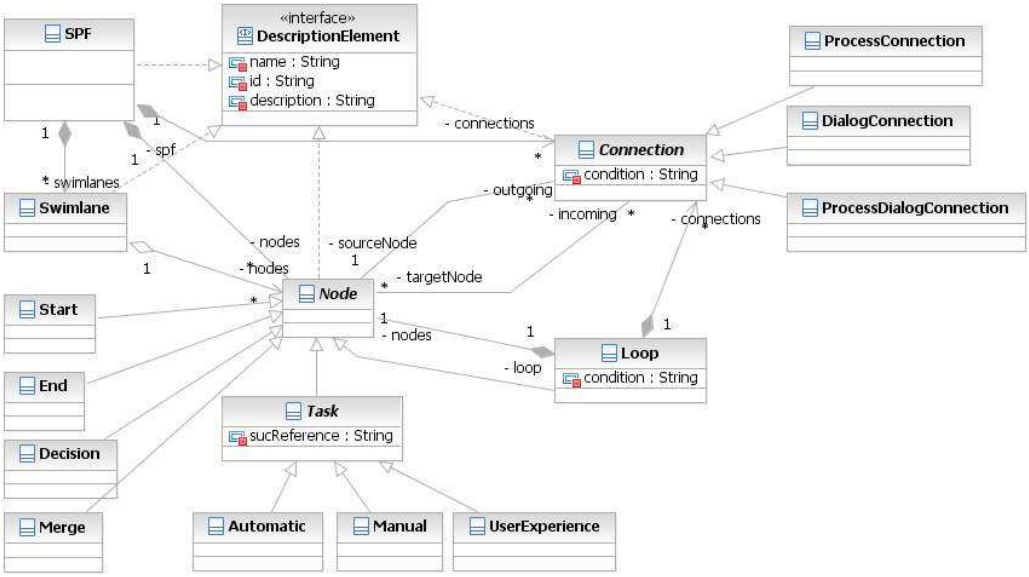


Figure 8: Solution Process Flow (SPF) metamodel. Tasks are modeled by the Automatic, Manual, UserExperience and SubProcess tasks types and connected in a control flow by using edges of type Process, Dialog or ProcessDialog.

process should be implemented as an inlined flow or as a sub process.

The *AutomaticSpecification* metaclass requires following attributes

- *retriesBeforeEscalation*: Service invocations occasionally fail due to network timeouts. This attribute indicate how many retries should be executed before escalating the failure.
- *escalationType*: The type of escalation in case a technical failure occurs. Can either be *Manual* indicating the a human activity is started from within the workflow, *Peregrine* which is a failure report generated by the Danske Banks extension to the workflow engine, or *Both*.
- *timeout*: A timeout indicator in milliseconds indicating when the service invocation has failed due to network failure.
- *restartable*: A boolean indicating if the service invocation must be restarted in case of failures. As the environment in which the workflow system operates is heterogeneous, two phase commit is not an option. If a service invocation fails the workflow engine cannot be certain if the operation was invoked or not. Therefore restartable must only be true is case the service implementation is immune for several similar invocations.

It further requires a *TechnicalFault*, an *Operation* and *InputSchema* and *OutputSchema* objects. The *TechnicalFault* object specifies an exception that the service operation may throw when invoked. The *Operation* object specifies name and version of the service operation to be invoked and the pattern to be used for implementation, i.e. *ServiceOperation*, *MultipleInstances* or *Bundle*. The *InputSchema* and *OutputSchema* objects specify data structures for the service operation. These are modeled using the DataStructure metamodel.

The *ManualSpecification* and *UserExperienceSpecification* metaclasses are equal except that the *UserExperienceSpecification* contains a link to the business application where a process participant has to execute the task. For both specifications it must be defined if the task can be cancelled or skipped by the user and if the business application is aware of the workflow system so the task can be completed directly from the business system (wfLevel: basic, lite or full). Further, presentation of a task in the task portal is specified by using the *ListPresentation* and *DetailsPresentation* metaclasses. Here, textural descriptions to be used by the task portal is described in three to five languages, and it is described which data values from within the workflow that has be visible in the task portal, as e.g. loan amount and customer name. The *AllocationRule* metaclass is used to define who is allowed to claim, execute and complete the task and the *DeadlineRule* metaclass is used to specify the task deadline and earliest start.

5.4.4 DataStructure Metamodel

The DataStructure metamodel (Figure 10) has been extracted from the WSUC metamodel to allow reuse of the metamodel by several tools. A *DataModel* contains a number of *DataStructure* classes. A *DataStructure* class is used to model input and output data for a service operation. It is quite simple; A *DataStructure* consists of a number of elements which can either be *SimpleElement*, which is a string, an integer or a boolean, or *ComplexElement* which itself contains other elements.

5.4.5 BPEL Code Generation Metamodel

The BPEL code generation (BPELCodeGen) metamodel is used to store decisions of how to implement a physical design in BPEL (Figure 11). It refers to a Solution Process Flow and specifies target namespace to use for the BPEL process, name of the project that should contain the BPEL process, base package name to define the BPEL process in, if WSDL

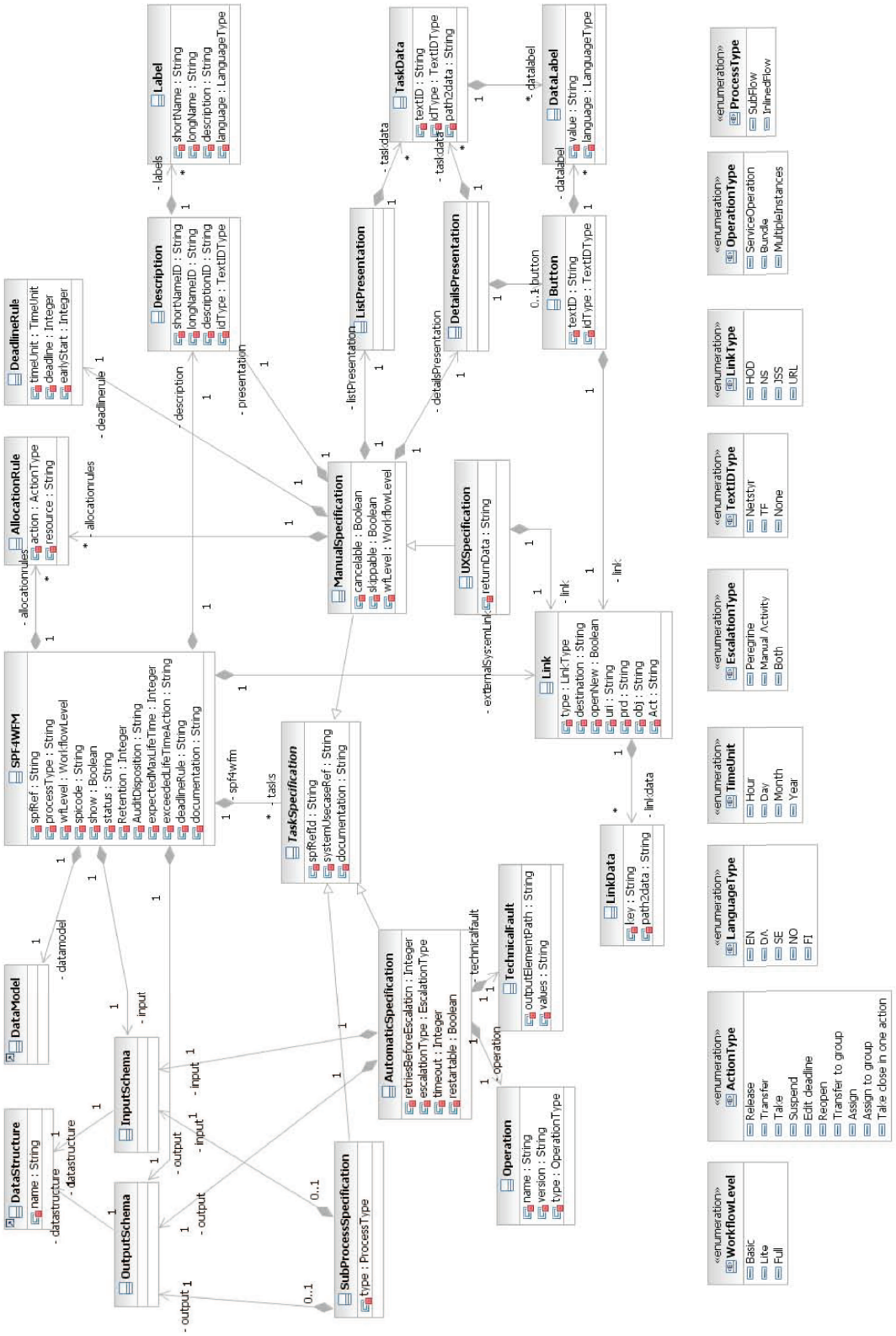


Figure 9: Workflow specification (WFSpec) metamodel. The SPF4WFM metaclass refers to a Solution Process Flow and contain a number of TaskSpecifications. A task specification refers to a task in a Solution Process Flow and can be of type Manual, UX, Automatic or SubProcess

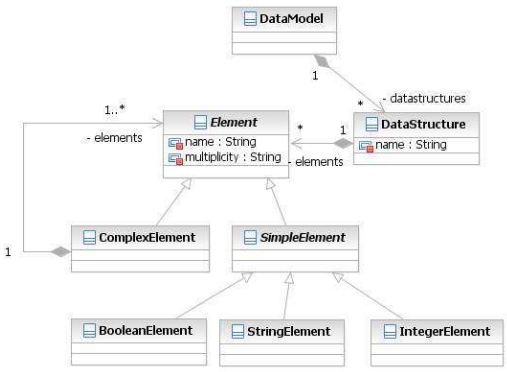


Figure 10: Data structure (DataStructure) metamodel

files should be located in the separate folders, and the name of the base WDSL folder name. While these attributes ex-

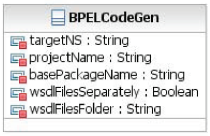


Figure 11: BPEL code generation (BPELCodeGen) metamodel

presses some of the basic decisions of BPEL implementation, the metamodel could be extended to provide several more variation points.

5.4.6 Model Injection Metamodel

In section 5.3 it was described how the Control Flow Behavior could be generated based on the Solution Process Flow model and model injections. The ModelInjection metamodel illustrated in Figure 12 keeps track of all process fragments to inject and where to inject them in a Solution Process Flow model. The ModelInjection metaclass has a reference back to the Solution Process Flow (spfRef) and contain a number of Injections. An Injection class specifies the point in the SPF where the process fragment should be injected. The injection point (JointPointSpfConnectionId) is specified by the ID of a connection in the SPF. It further contains a

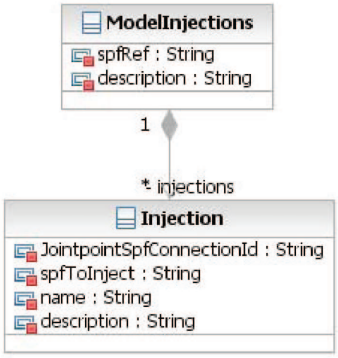


Figure 12: Model injections (ModelInjections) meta model

file reference to the process fragment to inject (spfToInject). As mentioned earlier, process fragments are modeled in separate files using the SPF metamodel.

5.5 Transformations

Now, when all information required during the development process can be stored precisely in models, we are able to describe algorithms for how to transform a Solution Process Flow and related models into BPEL. We may further describe an algorithm to generate an empty Solution Process Flow from an event and another algorithm to generate a Workflow Specification (WFSpec) model based on a Solution Process Flow.

Previously, knowledge of the transformation algorithms was only implicitly available in the minds of architects and developers. This knowledge has been extracted into explicit algorithmic descriptions in pseudo code (Algorithm 1,2 and 3) which form the basis for creating tool based transformations.

5.5.1 From Event to Solution Process Flow

One Solution Process Flow model has to be created for each event in the eventmap. The model is given the same name as the event and is stored under a folder named "SPF". As the event map is the first artifact to be created in the project, the transformation described by Algorithm 1 only creates an empty Solution Process Flow model with the correct name. After executing Algorithm 1, the analyst and the architect

Algorithm 1 From Event to SPF

```
procedure EVENT2SPF(event::Event)
    for all events in eventmap do
        create spf :: SPF
        set spf.name = event.name
        spf.nodes.add(create start :: StartNode)
        spf.nodes.add(create end :: EndNode)
    end for
end procedure
```

model the behavior of the business process inside the generated Solution Process Flow model.

5.5.2 From SPF to Workflow Specification

The physical specification, i.e. the Workflow Specification (WFSpec) model, can be generated based on the Solution Process Flow model. It is named and stored according to Danske Banks standards. A simplified algoritm is specified in Algorithm 2. A TaskSpecification class is generated and added to the WFSpec model for each task in the Solution Process Flow. The WFSpec model contains all required information, but all attributes contain default values. Successively it therefore has to be filled with correct information by the architect. As the architect changes a generated model, there is a risk that this information is lost when the model is regenerated. This can be avoided by letting the transformation algorithm recognize if a WFSpec model already exists and if the TaskSpecification has already been defined.

5.5.3 Generation of the BPEL implementation

The physical design and specification is finished after information has been filled into the WFSpec model, process fragments has been modeled and the ModelInjection model

Algorithm 2 From SPF to Workflow Specification

```
procedure SPF2WFSpec(spf::SPF)
    create spf4wfm :: SPF4WFM
    spf4wfm.spfRef = fileLocation(spf)
    spf4wfm.allocRules.add(create AllocationRule)
    spf4wfm.extSystemLink =create SystemLink
    spf4wfm.input =create InputSchema
    spf4wfm.output =create OutputSchema
    for all tasks :: Automatic in spf do
        create spec :: AutomaticSpecification
        create spec.technicalFault :: TechnicalFault
        create spec.operation :: Operation
    end for
    for all tasks :: Manual in spf do
        create spec :: ManualSpecification
        spec.allocRules.add(create AllocationRule
        spec.deadlineRules.add(create DeadlineRule
        create spec.presentation :: Description
        create spec.listPres :: ListPresentation
        create spec.detailsPres :: DetailsPresentation
    end for
    for all tasks :: UserExperience in spf do
                    ▷ Same steps as for ManualSpecification
        create spec :: UserExperienceSpecification
        create spec.link :: Link
    end for
    for all tasks :: SubProcess in spf do
        create spec :: SubProcessSpecification
    end for
end procedure
```

created, and a BPELCodeGen model has been created with information about the BPEL implementation. All information required to implement the BPEL process is now present in models. Four different kinds of models contain the necessary information to create the BPEL implementation; a model of the Solution Process Flow and models of its sub processes, a model of the workflow specification, a model of model injections, and a model of BPEL code generation decisions.

Algorithm 3 describes at a high level how these four models can be merge together to generate the BPEL implementation where only control flow logic and data mapping has not been taken care of. The full algorithmic implementation is much more complex than this description and contains many steps to e.g. manage namespaces for several web service documents, generation of XSD schema definitions for data types, setting up correct relations between BPEL partnerlinks and WSDL documents etc.

The algorithm contains a recursive graph transformation procedure, $graphTransform()$. This procedure first locates the initial node in the Solution Process Flow and then it recursively traverse the complete control flow graph and generates the corresponding BPEL implementation. The graph transformation delegates the responsibility for the task transformation to a separate pattern transformation procedures based on the implementation pattern specified in the workflow specification. The described algorithm has been limited to transformation of simple control flow graphs. Cyclic behavior, loops and other often used constructs have not been described nor implemented.

6. TOOL CUSTOMIZATION

In this section we describe a tool suite, called Danske Bank Workbench (DBW) that implements the metamodels and transformations described in last section. Hence, it directly supports Danske Banks development methodology and domain concepts. It has been built on the Eclipse platform (Eclipse (2008)) and various Eclipse open source projects. The Eclipse Modeling Framework (EMF) (Budinsky et al. (2003))has been used for defining the abstract syntax, or metamodels of the DSLs, that have been implemented. The concrete syntax of the DSLs and editor support have been implemented by using the Graphical Modeling Framework (GMF (2008)), while openArchitectureWare (oAW) (oAW (2007)) has been used to implement the semantics of the DSLs as model-to-model and model-to-text transformations.

We start be introducing the Eclipse projects related to model driven development, on which the DBW is based. This is a necessary introduction before we describe the overall architecture of DBW and how metamodels, editors, transformations and functional utilities that support easy development have been build. Last, we illustrate the usage of the tools by revisiting the Customer Quick Loan project.

6.1 Model Driven Development in Eclipse

Many Eclipse projects exist for supporting the development of tools for model driven development. The MOF MOF (2006) and the UML UML2.0 (2004) specification have been implemented under the Eclipse Modeling Framework (EMF) project and the UML project. It is possible to define and create metamodel implementations using both these implementation. For Danske Bank Workbench it was decided to define and implement metamodels based on EMF as both the openArchitectureWare (oAW) framework and the Graphical Modeling Framework (GMF) supports EMF directly. Now, we introduce four important projects, that the Danske Bank Workbench builds upon.

6.1.1 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) (Budinsky et al. (2003)) forms the basis for most of the modeling projects within Eclipse. EMF is an implementation of a subset of the MOF specification and contains a meta-meta model called Ecore, which is used for defining ones own metamodels. All metamodels described in section 5 have been implemented using EMF. The EMF project provides wizards for generating an Ecore model from annotated Java interfaces, a UML model and an XSD schema. The EMF framework provides code generation facility which can generate Java code from an Ecore model. This code is used by Eclipse based tools to create, modify and persist models. The EMF framework is responsible for serialization and deserialization of models as well as event notifications when a model changes. A part of EMF, the EMF.Edit framework forms a bridge between the Eclipse UI framework and the EMF core framework and comes as a part of the EMF project. The Ecore model can be used to generate the EMF.Edit Java classes which are used by editors and viewers to display and edit a model. The EMF and the EMF.Edit framework can be used to create editors for creating and editing models. The EMF code generation facility is also able to generate a simple tree based editor that takes advances of the generated EMF and EMF.Edit implementations.

The tree based EMF editor quickly becomes insufficient.

Algorithm 3 Constructing BPEL implementation

```
procedure ConstructBPEL(spf::SPF, wfspec::WFSpec, modelinj::ModelInjections, bpelCodeGen::BPELCodeGen)
    create bpel, bpel.wsdlinterface
    create bpel.variables[input, output]fromspf
    for all tasks :: Automatic in spf, modelinj do
        wsdlfile = createWDSL(task, wfspec, bpelCodeGen)
        bpel.partnerlinks.add(createPartnerLink(task))
        bpel.variables.add(create request/response)
    end for
    if count(tasks :: HumanTask > 0) then
        wdslfile = createWSDL("HumanTask")
        bpel.partnerlinks.add(createPartnerLink("HumanTask"))
    end if
    node = findInitialNodeInSPF(spf)
    graphTransform(node, wfspec, modelinj)
    save bpel in location based on bpelCodeGen
end procedure

procedure graphTransform(node::Node, wfspec::WFSpec, modelinj::ModelInjections)
    for all node.outgoings as edge do
        child = edge.targetNode
        taskSpec = getTaskSpec(wfspec, child)
        if child instanceof Automatic then
            if taskspec...type == Serv..Operation then
                Serv..OperationPattern(child,taskSpec)
            else if taskspec...type == Bundle then
                BundlePattern(child,taskSpec)
            end if
        else if child instanceof Manual then
            HumanTaskPattern(child,taskSpec)
        else if child instanceof UserExperience then
            HumanTaskPattern(child,taskSpec)
        else if child instanceof SubProcess then
            SubProcessPattern(child,taskSpec)
        end if
        if hasModelInjection(edge, modelinj) then
            Load process fragment from modelinj
            Transform process fragment into bpel
            Insert generated bpel fragment into bpel.
            Break edge into two to connect to fragment
        end if
        graphTransform(child, wfspec, modelinj)
    end for
end procedure

procedure ServiceOperationPattern(node::Automatic, taskspec::AutomaticSpecification)
    Create WSDL document, XSD schemas, partnerlinks,
    input/output variables. Customize the BPEL
    template by defining attributes.
end procedure

procedure BundlePattern(node::Automatic, taskspec::AutomaticSpecification)
    Create WSDL document, XSD schemas, partnerlinks,
    input/output variables and customize the BPEL
    template by defining attributes
end procedure

procedure HumanTaskPattern(node::Manual, taskspec::ManualSpecification)
    Create input/output variables
    Define allocation rules and data in Human Task Manager
    Define XML document with all data
end procedure
```

Instead, the Graphical Editing Framework (GEF) and the Graphical Modeling Framework (GMF) can be used for creating graphical modeling editors.

6.1.2 Graphical Editing Framework

GEF is a framework for building graphical editors. It is quite advanced and difficult to learn and much low level implementation code has to be written. GEF is meant to be used with an arbitrary metamodel. Hence, it is not aware of the structure and functionality of EMF models. It takes much effort to integrate it with the EMF and the EMF.Edit framework. The developed editors only implicitly depend on GEF. They have instead been build upon GMF.

6.1.3 Graphical Modeling Framework

The GMF framework was build to address the challenges experienced with combining EMF and GEF. It is a bridge between EMF and GEF and provides a generative component and runtime infrastructure for developing graphical editors. GMF itself is using EMF. In a number of models a tool developer defines how the editor and the metamodel to be used by the editor relates. The GMF generative component can now generate the editor implementation based on these models. This code relies on both the GMF runtime infrastructure as well as the GEF framework. The developed editors for the Eventmap, the SPF and the WFSpec metamodels have been built by using GMF.

6.1.4 openArchitectureWare

The openAchitectureWare (oAW) project is an Eclipse based MDD generator framework. It supports parsing and generating arbitrary models, and has a strong support for EMF based models. It consists of a number of specialized languages to support model checking (Check), model-to-model transformations (Xtend) and model-to-text transformations (Xpand). These languages can be combined with external Java code if needed. At its core oAW has a workflow engine that executes transformation workflows. A workflow may specify source models for the transformation, components to check the validity of models and which transformation scripts to be used for either model-to-model or model-to-text transformations. The transformational algorithms have all been implemented in oAW and are directly executable from Danske Bank Workbench.

There are many other projects related to model driven development in Eclipse, but as they are not used in the prototype, they will not be discussed here.

6.2 Tool architecture

Danske Bank Workbench is build on Eclipse 3.3 Europa edition with corresponding versions of EMF, GEF and GMF. Version 4.2 of OAW is used. Metamodels have been implemented using the Eclipse Modeling Framework (EMF), graphical editors have been generated by using the Graphical Modeling Framework (GMF), and transformations have been implemented using openArchitectureWare (oAW).

Danske Bank Workbench consists of several independent tools for developing the different artifacts in the development process. These are depictured in Figure 13, which also illustrates dependencies to other Eclipse projects. The names of the projects conform to the names of the metamodels previously described. Each project contains several Eclipse plug-ins which are named as follows:

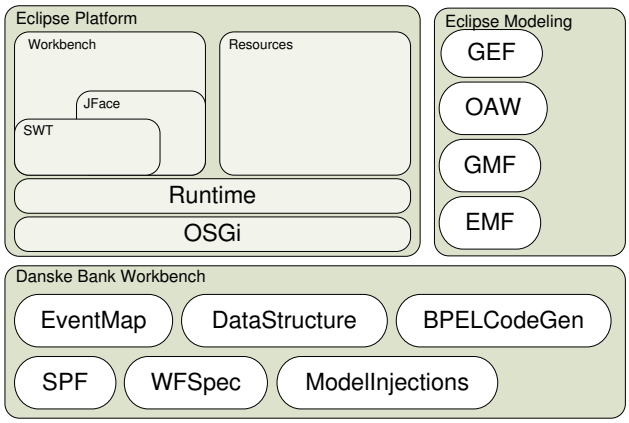


Figure 13: Overview of Danske Bank Workbench and its dependencies of other Eclipse projects

- **<projectname>.model**. Contains the Ecore model, GMF models and the EMF Java implementation
- **<projectname>.edit**. Generated EMF.Edit project based on the Ecore model.
- **<projectname>.editor**. Generated EMF editor based on the Ecore model. Is only used by the DataStructure, ModelInjections and BPELCodegen metamodels, for which there have not been developed a graphical editor.
- **<projectname>.diagram**. Generated GMF editor based on the GMF models contained in the <projectname>.model project.
- **<projectname>.generator**. Contains OAW transformation definitions for the EMF model defined in the <projectname>.model project. It also contains utility tools for executing transformations and to enrich the model. Generator plugins have been created for the EventMap, the SPF and the WFSpec project, and contain transformation definitions for Algorithm 1, 2 and 3

6.3 Metamodels and editors

All metamodels have been modeled in Rational Software Architect as UML class diagrams. Each of these were exported as an XMI representation of UML and imported into Eclipse by using the EMF model creation wizard which comes as a part of the EMF project. This wizard generates an Ecore model based on the UML model, and it also generates another EMF model, the codegen model, that is used for generating code. Using this model, it is possible to specify attributes for the EMF code generator that is not a part of the Ecore model as e.g. name of the Eclipse project and the base package to generate the code in. Actually, the BPELCodeGen metamodel was inspired by the EMF codegen model.

Models for creating a GMF editor are created after finishing an EMF metamodel. The *tool* model is used to define which tools that are available in the editor. The *graph* model is used to define graphical representations to be used in the editor. For instance, the shape of a figure to represent an allocation rule has been specified as a polygon that illustrates

a human. A shape may also contain a number of text labels. The *mapping* model is used to combine the tool model, the graph model and the domain (EMF) model. For example it can be defined that tool A creates a graphical figure B on the canvas and that it represents domain element C. The mapping model is also used to map a label on a graphical figure to an attribute on a domain element. For example, the allocation rule figure has a label that is mapped to the action attribute at the AllocationRule element in the WFSpec metamodel. Hence, the action attribute of an AllocationRule object is visible and directly editable from the AllocationRule figure at the canvas. The mapping model is used to generate a *generator* model. This model contains all information necessary for generating the GMF editor. It is similar to the EMF codegen model that is contains additional information about the code generation that is not present in any of the other models. The GMF editor implementation is generated from this model.

Figure 14, 15 and 16 illustrate the GMF based Eventmap, SPF and Workflow Specification (WFSpec) editors in action.

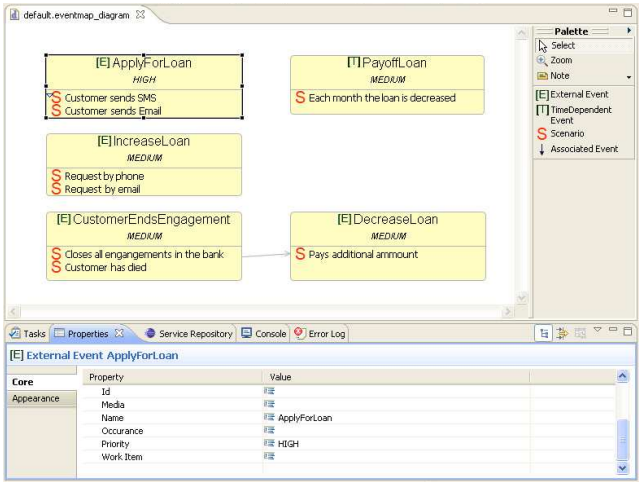


Figure 14: Event map editor with events for the CustomerQuickLoan project. External and Timedependent events can be modeled directly from the tool palette and required information can be specified as properties.

6.4 Transformations

The three transformation algorithms described previously have been implemented in oAW. Algorithm 1 and 2 have been implemented as model-to-model transformations using the Xtend language. Algorithm 3, which is supposed to generate BPEL code has been implemented as a model-to-text transformation using the Xpand language. The implementation is quite complex. It is implemented as a graph transformation that recursively runs through the SPF controlflow starting with the initial node. When a model injection or a sub process is detected, the corresponding process fragment or sub process is be interpreted and BPEL code generated, which must then be merged into the partly generated BPEL code. It requires much book keeping to handle the associations between models as four different models, i.e. SPF, WFSpec, ModelInjection and BPELCodeGen are used as input to the transformation. A number of utility functions

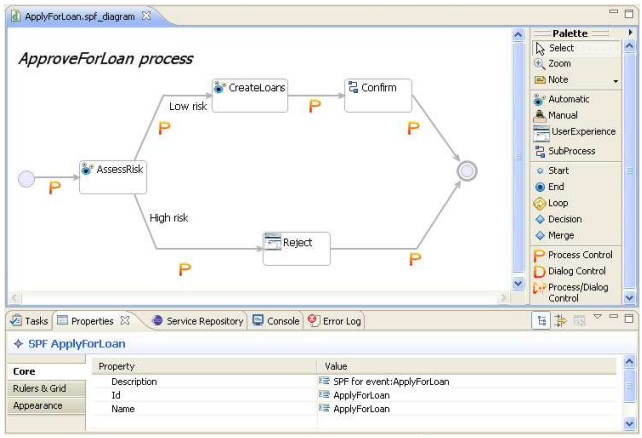


Figure 15: Solution Process Flow editor. The ApproveForLoan process is modeled. Task and connection types are available from the tool palette. The concrete syntax is customized for tasks as well as edges.

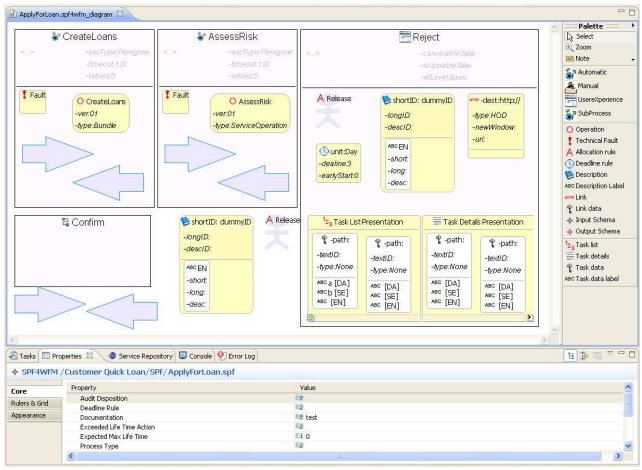


Figure 16: WFSpec editor with task specifications for the ApproveForLoan process. Information can be modeled precisely for Automatic, Manual and UserExperience tasks.

have been written in Xtend and in Java to support this.

Each transformation is executed by an oAW workflow. The workflows as well as the transformation definitions in Xtend and Xpand are located in the *generator* plugins.

6.5 Tool Utilities

Several tool utilities have been developed to enhance usability of Danske Bank Workbench and to smoothen the use of the different tools. The users of the tools are guided from one step in the development process to the next by using these utilities.

6.5.1 Transformation Execution

One kind of tool utility is the generation of "the next" development artifact in the development process, i.e. execution of transformation workflows that implement Algorithm 1, 2 and 3. These are implemented as actions that appears on the context menu when the user right-clicks on an event in

an eventmap, at the canvas for a SPF and at the canvas for a WFSpec model.

6.5.2 Service Repository Data Extract

Another task to be handled by a modeler is to find definitions of input and output data structures for service operations and put them into the WFSpec model. The user right-clicks on a task specification for an *Automatic* task and chooses "Retrieve Repository Data". The executed action looks up the defined service operation in (a mock up of) Danske Banks centralized service repository, retrieve definitions of data structures and updates the WFSpec model with these.

6.5.3 Persistence of manually changed BPEL code

Generated BPEL code needs to be updated with data mapping and control flow logic. A small persistence framework has been developed that allows the developer to persist logic from within an assign node or a control link in a separate file. The developer simply right-clicks on the assign node or control link and chooses "Persist element". The action creates a separate file where the assign or control flow logic is persisted. Next time the transformation that implements Algorithm 3 is executed, the changed BPEL code is overridden, but successively, the persisted changes are copied into the newly generated BPEL code.

6.6 Customer Quick Loan Retooled

Danske Bank Workbench will now be illustrated by applying it at the example. Figure 17 illustrates a workflow of the development process with the artifacts that are created and the transformations between them. The letter tags in the figure refer to screen dumps of tool utilities and artifacts developed for the ApproveForLoan process. They are depictured in Figure 19 which can be found in Appendix A.

First, a business analyst creates a new eventmap. All business events are now modeled as either *external* or *time-dependent*, and scenarios are added to each event (Figure 19a). The editor provides direct support for these concepts from the tool palette. The analyst simply drags and drops events and scenarios to the canvas. The property view reflects properties for the selected event type, where e.g. priority can be selected as low, medium or high and business possibilities can be described. Event types and properties directly reflect the defined metamodel for an eventmap. Previously, the analyst had to model the events in one tool and define all information in a separate textural document.

After finishing the event map, the business analyst has to create a Solution Process Flow for each event. It must be named correctly and placed in a certain folder structure. The analyst simply right-clicks on the event, for instance the ApproveForLoan event, in the eventmap editor and chooses "Generate SPF" (Figure 19b). An empty Solution Process Flow is now generated in a subfolder named "SPF" and is given the same name as the business event. It is then modeled by either the business analyst or the solution architect. Tasks may now be modeled directly as *Automatic*, *Manual* or *UserExperience* (as defined by the SPF metamodel) by dragging them directly onto the canvas from the tooling palette. Connections of type *Process*, *Dialog* or *ProcessDialog* are also directly available. The usability and the preciseness have increased compared to the general modeling tool previously used.

The architect right-clicks at the Solution Process Flow when it is complete (Figure 19c) and chooses "Generate WFSpec model"(Figure 19d). A WFSpec model is now generated under the Implementation folder and a subfolder named after the Solution Process Flow. It contains task specifications for all tasks and has been populated with default data. The task specifications and all objects inside them conform directly to the WFSpec metamodel.

Now, information has to be entered into the specification. For example, the architect defines that the CreateLoans task must be implemented by the Bundle pattern; he or she selects the Operation object in the CreateLoans task specification and in the properties view changes the type from *ServiceOperation* to *Bundle* (Figure 19i).

The Confirm task is modeled as a SubProcess task type in the Solution Process Flow. The architect chooses that it must be implemented as an inlined flow in the BPEL process by selecting the Confirm task specification in the WFSpec model and sets the *Type* property to *InlinedFlow* (Figure 19j). The subprocess to which the Confirm task refer is generated by right clicking on it and choose "Generate SubProcess". An empty sub process is created and opened automatically. It is named accordingly to the name of the Confirm task and optionally put in a sub directory if the *path* property at the Confirm task has a value. The subprocess is now modeled as a sequence of two automatic activities (Figure 19g), and its workflow specification can be generated (Figure 19h).

The architect and the developer recognize that an additional task is needed in the physical implementation. The task should set the business state of the process instance to either "Approved" or "Rejected" depending of the outcome of the AssessRisk task.

They right click on the control link in the Solution Process Flow that connects the AssessRisk task with the CreateLoans, and the Reject task and choose "Create Injection". An empty model is created under the Injection folder and is automatically opened in an SPF editor. The developer models the process fragment as just one automatic activity (Figure 19l) and generates the workflow specification for it (Figure 19m).

The book keeping of Model injections are handled by the ModelInjection model. This model is illustrated in Figure 19q. It contains one injection that has two important properties; the injection point in the Solution Process Flow, which is the ID of the control link, and the process fragment to inject (SPF to Inject) at the injection point.

Before having the WFSpec metamodel and editor, all the design decisions were modeled in the Control Flow Behavior without any reuse of the Solution Process Flow, and required additional information was defined in textural documents.

Now, the project team has modeled three processes; one SPF, one subprocess and one process fragment. They are all modeled in the same language and have each a corresponding WFSpec model.

All automatic task specifications must be synchronized with the centralized Service Repository to obtain correct input and output data definitions. Figure 19o shows the selected CreateContent task specification in the WFSpec model for the Confirm subprocess. The modeler has right-clicked on it and selected "Retrieve Repository Data". The operation name for the CreateContent specification has been set to CREATECONTENT. The same operation name ex-

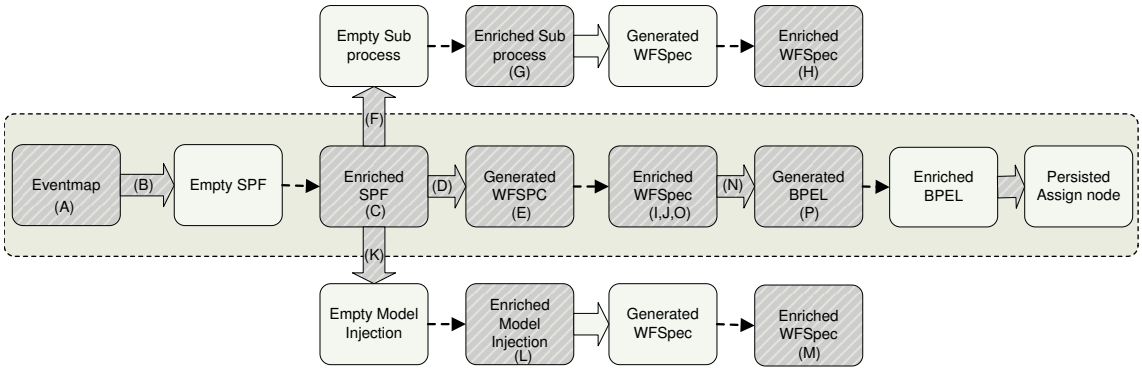


Figure 17: Workflow for using tools through the customized development process. Letter tags refer to screen dumps in Figure 19 which can be found in Appendix A. A thick arrow indicates a tool utility while a thin arrow indicates that a human has to enrich the model

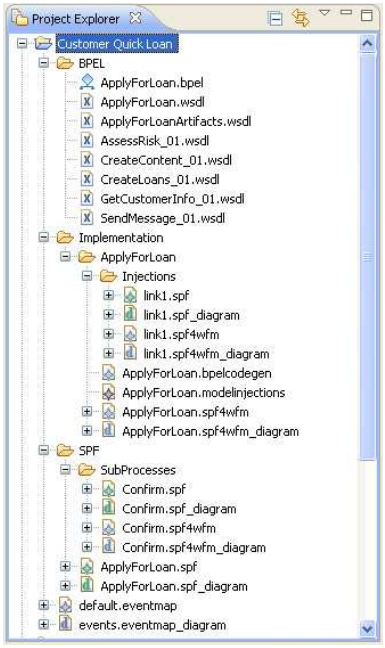


Figure 18: File structure of the Customer Quick Loan project containing all generated files.

ists in the service repository, which can also be seen in the figure. The action now retrieves data definitions from the repository and populates the WFSpec model with these. Subsequently, the BPEL code generator can use these data structures to create a correct WSDL document for the service operation.

Without this import utility, the developer had to find data definitions and create XSD schemas manually.

The developer sets parameters for the code generation in a BPELCodeGen model before executing the BPEL code generation. Previously, these design decisions were not documented. Figure 19r illustrates that the developer has selected default values for the code generation; The BPEL code will be generated in the same project as where the models are, and WSDL files will be located in the same directory as the BPEL file. The developer generates the BPEL code by right-clicking on the WFSpec model and choose "Generate BPEL" (Figure 19n).

Figure 18 shows the Customer Quick Loan project and all files generated through the development process and Figure 19p shows the generated BPEL code opened in the Eclipse open source BPEL editor.

Only the event map, which was the first artifact to be created, has been created manually. The rest of the artifacts have been created by tool utilities supporting the enterprise specific development process. Hence, the file structure follows specified standards, and the traceability between models can be ensured. Without Danske Bank Workbench these artifacts and all the information bookkeeping are handled manually by the project team.

7. EVALUATION

The tool was evaluated through an empirical test which involved five people. They have all worked as workflow developers. Two of them have experiences from working as - or closely together with - a business analyst, and one of them is a solution architect. They used Danske Bank Workbench to model an event map, a Solution Process Flow and a workflow specification and generate the BPEL code. The business scenario was the same as presented in this paper. They got a one-page description of how to use the tool. From the description they used about 30-40 minutes to complete the exercise. They filled out questionnaires with 12 questions and were interviewed about their experiences with the tool. Each question asked about the experience of using the tool compared to current practice in Danske Bank. The questions were answered by a rating from 1 to 6, where 1 is "Much worse", 2 is "worse", 3 is "a little worse", 4 is "a little better", 5 is "better" and 6 is "much better". The questions and their ratings can be found in Table 3.

The developers believed that the tool would improve their productivity significantly and it would be easier to work with. Especially, they were happy with the Danske Bank specific modeling capabilities. It was much easier and intuitive to work with domain specific modeling. Further was it easier to comprehend the workflow specific information that had to be specified for the Solution Process Flow. This is reflected in question 1 to 5. They all got a mean score around 5.4 out of 6, which is between "better" and "much better". Some of the developers suggested that validation

Table 3: Questions and answers for the empirical evaluation. The ratings were: 1 is "Much worse", 2 is "worse", 3 is "a little worse", 4 is "a little better", 5 is "better" and 6 is "much better"

	Question	*Mean value*
1	How is the Danske Bank specific syntax to work with compared to Websphere Business Modeler?	5.5
2	How is it to work with the WFSpec editor compared to MS Word?	5.6
3	Is the information easier to comprehend and access?	5.2
4	How is it to comprehend the number of modeling artefacts and locate where they are?	5.4
5	Are the tool utilities helpful and support the developement process	5.4
6	Is the code generation to prefer over manual translation	5.0
7	Do you believe in model driven development as the right direction to go in?	5.0
8	How is the quality of generated code compared to manually written code?	3.8
9	Do you prefer to model and generate the solution instead of manually implement it?	5.0
10	Does the tool eliminate tedious work?	5.0
11	Will the tool influence on the development productivity	5.0
12	Will the tool decrease the number of errors in implemented code?	4.8

rules would improve the development process as a modeler would be caught if required information was not specified. They also came up with suggestions of how to improve the usability of the modeling tools.

In general, they believed in the idea of model driven development and that it would help improving their daily work. This is reflected in question 6, 7, 9, 10 and 11. They all got a mean score at 5 out of 6, which is equal "Better". One of the developers suggested that data and control link logic should also be modeled to allow 100% code generation.

They liked the code generation capability, but this was of less importance compared to the modeling capabilities. However, as one of them stated, new workflow developers would benefit much from this capability as they could easily get from the model to the code. This would possible reduce the requirements for education. An experienced workflow developer would also benefit from the code generation, but he or she would be able to make the translation manually.

Some of the developers were quite skeptical about the quality of the generated code as they suspected that manually written code would perform better than generated one. This is reflected in the answers to question 8. It got a mean score at 3.8 which is between "a little worse" and "a little better". However, they believed that the number of errors would be lower in generated code compared to manually written code. This is reflected in question 12. It got a mean score at 4.8, which is between "a little better" and "better".

8. DISCUSSION

We have used Danske Bank Workbench for modeling and implementing the ApproveForLoan business process. The exemplification of the tool and the empirical evaluation has shown that the development process becomes more efficient as the different experts are supported directly in their work. They are able to use familiar domain concepts directly in the modeling tools, they are guided to provide correct information, and execution of the transformation algorithms has been automated. We have shown that it is possible to define and utilize a number of DSLs and tools to effectively support an enterprise specific development process for business process modeling and implementation.

Danske Bank Workbench is a prototype and therefore it has a number of limitations and points for improvements;

Consistency checking We have not defined methods, nor implemented tools to check consistency between models, although such tool support would provide much value to a project team.

Validation and modeling constraints Model validation is an important area that has not been dealt with. Validation rules and model constraints should be specified by the team responsible for defining the metamodels, and these rules and constraints should be handled by the modeling tool to avoid creation of invalid models.

Controlflow The prototype only supports transformation of certain types of controlflow. For instance, the transformation cannot handle cyclic behavior, or loop constructs. It further cannot transform a decision/merge structure into a switch block in BPEL, a construction that is often desired over a BPEL graph implementation.

Data mapping Data mappings are added manually to the BPEL implementation and persisted by the developed persistence utility framework. It might improve the prototype and the development process to abstract the definition of data mappings to either the WFSpec model or to a generated Java class which would be responsible for the data mapping.

Restrictions on the SPF The prototype only supports a one-to-one relationship between an SPF and a BPEL implementation. In reality there are often cases where an SPF might be divided to several BPEL processes or several SPF's may be merged into one BPEL implementation. For such cases there is a need for an explicit Control Flow Behavior model as it cannot be generated from the logical model(s). The BPEL code can still be generated by a (slightly modified) Algorithm 3, but the tool-based consistency insurance between the logical and the physical model is then lost.

The prototype has been build by using a language workbench that provided basic frameworks for defining metamodels, graphical editors and transformations. This has been very useful. However, Danske Bank Workbench is a prototype and it requires much work to be ready for production use. This was also mentioned through the empirical evaluation. The tool further needs to implement several useful features such as constraint validation, synchronization checking, etc. Building a tool chain for production use is

a challenging task that requires much expertise and effort. Building a business process modeling language and editor from scratch is very difficult. Many enterprises will neither have the expertise, the size, nor the wish to depend so much on internally developed tools.

What tool vendors should recognize is that enterprises need to be able to customize tools to their specific needs. In case of process modeling, the process modeling notation should be extensible, as e.g. UML activity diagrams and profiles provide. But instead of using general UML tools for a DSL based on e.g. an activity diagram, a commercial tool should be customized in an easy manner to provide the types and the visual presentation required by the enterprise.

Similarly, vendors should not assume one specific way to use technology. Enterprises often have a variety of technologies and systems and need to control how to use the technology to implement models. The model transformations should therefore be open and extensible for the enterprises.

A commercial model driven development tool suite should be customizable to an extent where tools like the ones presented in this paper may be defined and used. Brahe and Østerbye (2006) and Brahe and Bordbar (2006) have described an approach that allows such flexibility.

9. RELATED WORK

Only limited work has previously been made at customizing business process modeling notations and corresponding tools to a specific domain and enterprise. In general, most modeling languages like Petri nets (Murata (1989)), Event-driven Process Chains (EPC) and the Business Process Modeling Notation (BPMN) (White (2006)) only have one notation which all domains have to follow. An exception is UML activity diagrams that can be extended by a profile for a specific domain. UML Activity diagrams are used by both academia and industry for its extensibility and available tool support in form of general UML modeling tools. Dumas and Hofstede (2001) argues based on workflow patterns that the expressiveness of activity diagrams as a workflow language is large and Guntama et al. (2003) has extended activity diagrams to enable flexible business workflow modeling. There are also various UML profiles for business process modeling, e.g. List and Korherr (2005) who considers both the business process flow as well as the business process context.

Brahe and Østerbye (2006) uses UML activity diagrams as the semantic base for creating domain specific modeling languages for business process modeling based on UML profiles. They suggest that many enterprises need their own modeling notations and present a prototype tools that allows metamodeling of domain-specific workflow-based languages and automatically generation of domain-specific tool support in form of editors. Jablonski and Götz (2007) has a similar approach. They present a flexible and extensible metamodel and the concept of *perspective oriented business process visualization* that allows multiple visual presentations of a business process model

Model Driven Development and the Model Driven Architecture have been used extensively in transforming business process models to implementations, particularly from UML activity diagrams to service composition languages. Bézivin et al. (2004) uses the ATL transformation language to transform UML models into three different target platforms; Java, Web Services and Java Web Service Developer Pack. Bordbar and Staikopoulos (2004a,b) studies transformation of activity diagrams to BPEL and WSCI based on their MOF compliant metamodels. Skogan et al. (2004) proposes a method that uses activity diagrams to design web service compositions and transforms them into different service composition languages. The method also builds on transforming WSDL descriptions into UML, which can then be used to build the service compositions. The Danske Bank Workbench utility that retrieves service operation information from the service repository and populates the workflow specification model is an implementation of this idea. Koehler et al. (2003, 2005) has worked on model driven generation of BPEL implementations based on activity diagrams using techniques originating from compiler theory and declarations of rules in the Object Constraint Language. The BMNM specification contains a chapter that specifies how BPMN models can be mapped to BPEL. Using BPMN ensures using a language which is specifically designed for business process modeling.

Common for above transformational approaches is the use of a fixed modeling notation and a fixed transformation. In contrast, Brahe and Bordbar (2006) presents a transformation framework that builds upon the use of domain specific business process modeling languages and customized transformations. They also introduce a prototype implementation that allows definition and execution of customized and pattern-based transformations for a domain specific modeling language.

Fowler (2005) talks about Language Workbenches as tools that allow definition and usage of domain specific languages, editors and transformation between languages. Several of such workbenches exist such as MetaEdit+ (Tolvanen and Rossi (2003)), GME (Ledeczi et al. (2001)), Microsoft DSL tools, and many others. The Eclipse projects used to build Danske Bank Workbench can also be considered as a language workbench. The research presented in this paper follows cutting edge trends in language workbenches; models should be constructed in domain, or enterprise specific concepts and transformed into an implementation.

10. SUMMARY AND FUTURE WORK

In the introduction of the paper we postulated that general purpose business process modeling and implementation tool suites are not feasible for many enterprises. Using the case study of Danske Bank and an example we showed that a development team faces many challenges when they use standard modeling languages and tools but have to use enterprise specific modeling notations, follow an enterprise specific development process and use technology in specific ways.

We abstracted the development process into metamodels and transformational algorithms and developed a tool called Danske Bank Workbench, fitted specially for Danske Bank. The tool implemented the model driven development principles of direct representation and automation as it allowed creating models directly in Danske Bank specific concepts and it automated the generation of lower level models and code.

By using Danske Bank Workbench at the example, we saw that it is possible to achieve an efficient model driven development process where process participants collaborate to create different modeling abstractions of a business process with tool based transformations and ensured synchronicity between the different modeling abstractions. Using the tool,

information only has to be defined once, and it is easy to comprehend as it is stored in traceable models. Knowledge of implementation patterns is reused by automated transformation, and several tool utilities support the development process which makes Danske Bank Workbench very efficient to work with. An empirical evaluation of the tool confirmed this. Hence, we have confirmed the hypothesis that was set up in the introduction, which stated that applying the basic model driven development principles of direct representation and automation to BPM tools would solve many of the experienced challenges.

Danske Bank Workbench was not difficult to build as many language workbenches exist for build metamodels, editors and transformations (though it did require deep insight in various Eclipse technologies and MDD concepts). However, it has several limitations, and it only addresses a small subset of business processes that may be modeled. To make it a production ready tool that can be used by the organization requires much more effort. Despite a promising prototype, our guess is that only a very limited number of enterprises will go the way and implement their own tools. While it may be economical beneficial to develop ones own tools, there might be political reasons not to do so.

To answer the research question set up in the introduction we can now say,

> "Defining and developing a model driven development tool to support an enterprise specific business process development process is possible and seems promising. It will heighten the productivity of development teams and probably cause fewer errors in implementations. However, it requires a high degree of expertise in model driven development methodology and technology to develop such a tool. It will probably be unachievable for most enterprises"

Although language workbenches provide huge support in development of model driven development tools, it should be much easier to customize ones own languages and tools for a specific area as business process modeling and implementation. For future research we therefore suggest to work on tool-based frameworks that feature extensions of predefined languages, editors and model visualizations to a certain domain. It would require less investment and it would be easer for an enterprise without experienced tool developers to *customize* BPM tools instead of *develop* develop them from scratch.

References

Booch, G., Brown, A., Iyengar, S., Rumbaugh, J., and Selic, B. (2004). An MDA Manifesto. *Business Process Trends - MDA Journal, http://www.bptrends.com/publicationfiles/05-04%20COL%20IBM%20Manifesto%20-%20Frankel%20-3.pdf.*

Bordbar, B. and Staikopoulos, A. (2004a). Modelling and transforming the behavioural aspects of web services. In *Third Workshop in Software Model Engineering (WiSME 2004) at UML*, Lisbon, Portugal.

Bordbar, B. and Staikopoulos, A. (2004b). On Behavioural Model Transformation in Web Services. In *Conceptual Modelling for Advanced Application Domain (eCOMO)*, pages 667–678, Shanghai, China.

BPEL (2003). Business Process Execution Language for Web Services (BPEL4WS). Version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

Brahe, S. (2007). BPM on top of SOA: Experiences from the Financial Industry. In G. Alonso, P. D. and Rosemann, M., editors, *BPM2007*, volume 4714 of *LNCS*, pages 96–111. Springer, Heidelberg.

Brahe, S. and Bordbar, B. (2006). A Pattern-based Approach to Business Process Modeling and Implementation in Web Services. In Georgakopoulos, D., editor, *ICSOC 2006*, volume 4652 of *LNCS*, pages 161–172. Springer, Heidelberg.

Brahe, S. and Østerbye, K. (2006). Business Process Modeling: Defining Domain Specific Modeling Languages by use of UML Profiles. In Rensink, A. and Warmer, J., editors, *ECMDA-FA 2006*, volume 4066 of *LNCS*, pages 241–255. Springer, Heidelberg.

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., and Grose, T. J. (2003). *Eclipse Modeling Framework: A Developer's Guide.* Addison Wesley.

Bézivin, J., Hammoudi, S., Lopes, D., and Jouault, F. (2004). An Experiment in Mapping Web Services to Implementation Platforms. Technical report, LINA, University of Nantes.

Dumas, M. and Hofstede, A. H. M. (2001). UML Activity Diagrams as a Workflow Specification Language. In *UML 2001*, volume 2185 of *Lecture Notes in Computer Science*, pages 76–90.

Dumas, M., van der Aalst, W., and Hofstede, A. (2005). *Process-Aware Information Systems: Bridging People and Software through Process Technology.* John Wiley & Sons, Inc.

Eclipse (2008). The Eclipse project, www.eclipse.org.

Erl, T. (2005). *Service Oriented Architecture: Concepts, Technology and Design.* Prentice Hall.

Flynn, D., Vagner, J., and Vecchio, O. D. (1995). Is CASE technology improving quality and productivity in software development? *LOGISTICS INFORMATION MANAGEMENT*, 8(2):8–21.

Fowler, M. (2005). Language Workbenches: The Killer-App for Domain Specific Languages? http://martinfowler.com/articles/languageWorkbench.html.

GMF (2008). Graphical Modeling Framework project. http://www.eclipse.org/gmf.

Guntama, E., Chang, E., Jayaratna, N., and Pudhota, L. (2003). Extension of Activity Diagrams for Flexible Business Workflow Modeling. *International Journal of Computer Systems Science & Engineering*, 18(3):137–152.

Jablonski, S. and Bussler, C. (1996). *Workflow Management - Modeling Concepts, Architecture and Implementation.* Intl. Thomson Computer Press, London.

Jablonski, S. and Götz, M. (2007). Perspective oriented business process visualization. In *3rd International Workshop on Business Process Design (BPD) in conjunction with the 5th International Conference on Business Process Management (BPM 2007). Brisbane, Australia*.

Koehler, J., Hauser, R., Kapoor, S., Wu, F. Y., and Kumaran, S. (2003). A Model-Driven Transformation Method. In *7th International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 186–197.

Koehler, J., Hauser, R., Sendall, S., and Wahler, M. (2005). Declarative Techniques for Model-Driven Business Process Integration. *IBM Systems Journal*, 44(1):47–65.

Kroll, P. and Kruchten, P. (2003). *The Rational Unified Process Made Easy. A Practitioner's Guide to the RUP*. Addison Wesley.

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., and Volgyesi, P. (2001). The Generic Modeling Environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary, http://www.isis.vanderbilt.edu/Projects/gme/GME2000Overview.pdf*.

Leymann, F. and Roller, D. (2000). *Production Workflow: Concepts and Techniques*. Prentice Hall.

List, B. and Korherr, B. (2005). A UML 2 Profile for Business Process Modelling. In *Perspectives in Conceptual Modeling, ER 2005 Workshops*, volume 3770 of *Lecture Notes in Computer Science*, pages 85–96.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

MOF (2006). Meta Object Facility (MOF) 2.0 Core Specification, available at http://www.omg.org/cgi-bin/doc?formal/2006-01-01.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580.

oAW (2007). openarchitectureware, http://www.openarchitectureware.org.

Skogan, D., Grønmo, R., and Solheim, I. (2004). Web Service Composition in UML. In *Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC'04)*, pages 47–57.

Stahl, T., Völter, M., Bettin, J., Haase, A., and Helsen, S. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.

Swithinbank, P., Chessell, M., Gardner, T., Griffin, C., Man, J., Wylie, H., and Yusuf, L. (2005). *Patterns: Model-Driven Development Using IBM Rational Software Architect*. IBM Redbooks, available at http://www.redbooks.ibm.com/abstracts/sg247105.html?Open.

Tolvanen, J.-P. and Rossi, M. (2003). MetaEdit+: defining and using domain-specific modeling languages and code generators. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 92–93, New York, NY, USA. ACM.

UML2.0 (2004). UML 2.0 Superstructure Specification, Final Adopted Specification, available at http://www.omg.org/docs/formal/05-07-04.pdf.

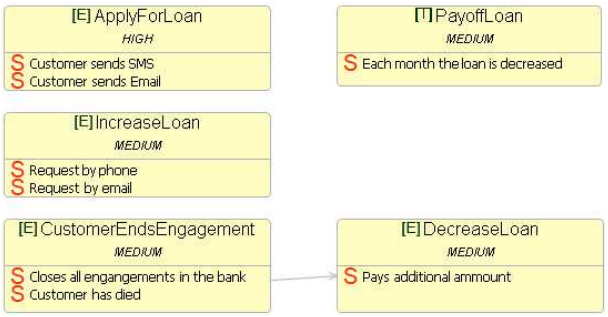
van der Aalst, W. M. P., Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003). Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51.

White, S. (2006). Business Process Modeling Notation, Version 1.0, final adopted version, avaiblable at http://www.bpmn.org/Documents/OMG-02-01.pdf.

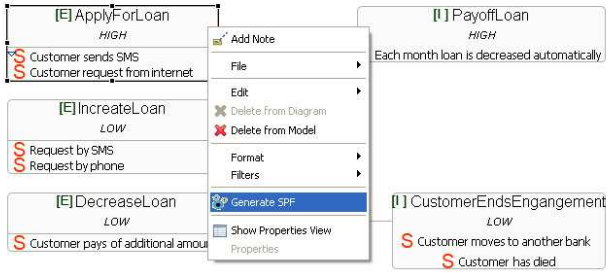
Windsor, J. (1986). Are automated tools changing systems analysis and design? *Journal of Systems Management*, 37(11):28–33.

APPENDIX

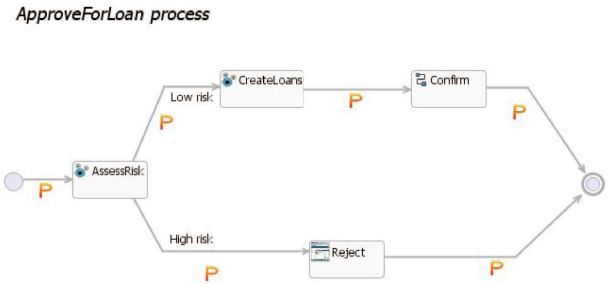
A. PROTOTYPE SCREEN DUMPS



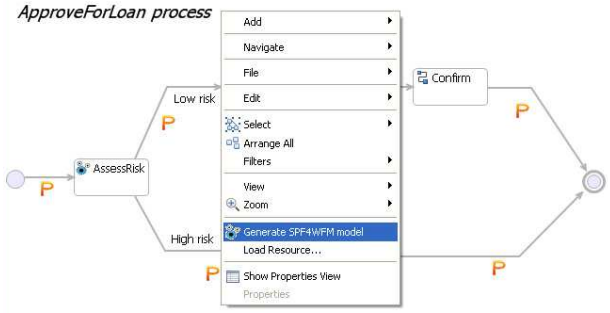
(a) Event map model. Events are modeled as External or Timedependent events and contain scenarios



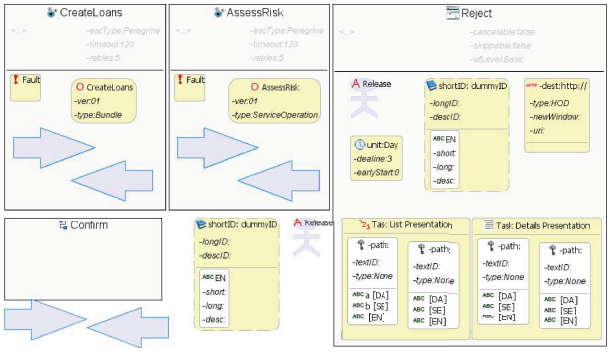
(b) Tool utility to generate a Solution Process Flow. The modeler has right-clicked at the ApproveForLoan event.



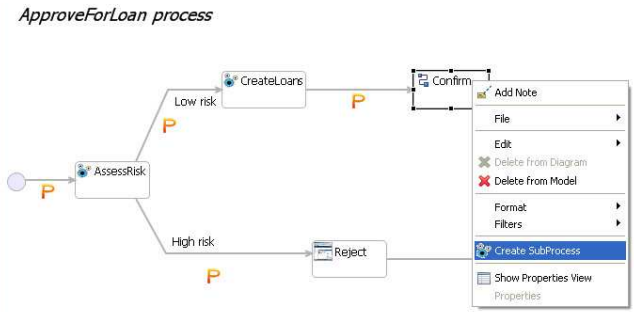
(c) Solution Process Flow for the ApproveForLoan event. The business analyst or solution architect has completed the generated model



(d) Tool utility to generate the Workflow Specification model for the ApproveForLoan Solution Process Flow. The modeler has right-clicked at the canvas

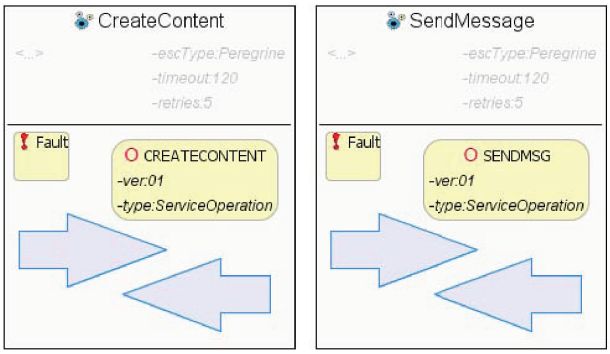


(e) The generated Workflow specification (WFSpec) model for the ApproveForLoan Solution Process Flow



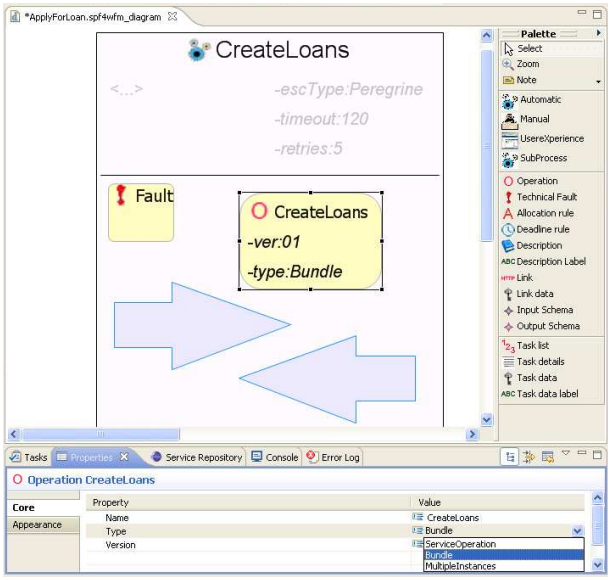
(f) Tool utility to create an empty sub process. The modeler has right-clicked at a SubProcess task in the Solution Process Flow

(g) The Confirm subprocess as modeled by a business analyst or a solution architect

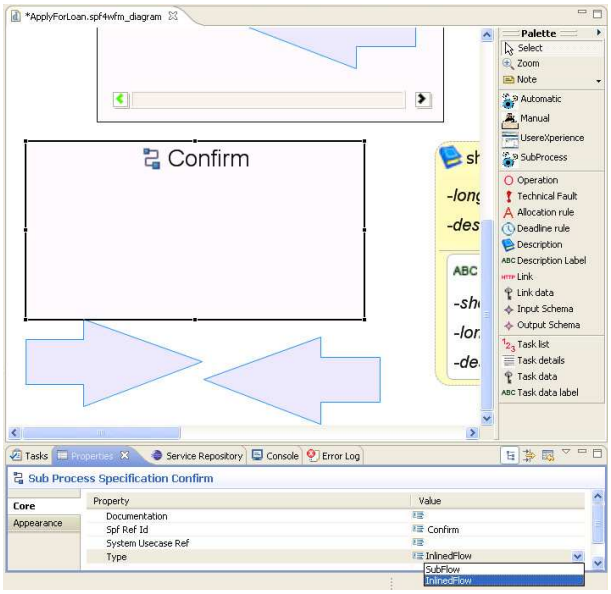


(h) Generated Workflow specification for the Confirm subprocess

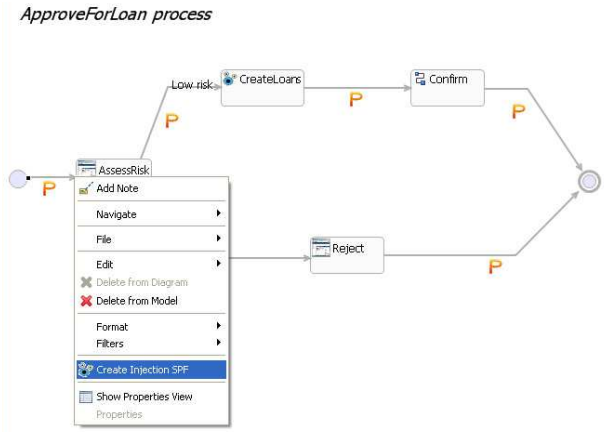
Figure 19: Using Danske Bank Workbench



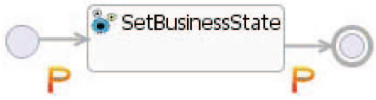
(i) Setting the CreateLoans task to be implemented by the Bundle pattern



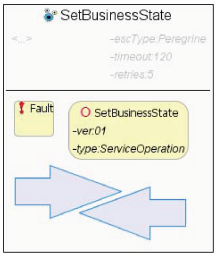
(j) Setting the confirm subprocess to be implemented in the ApproveForLoan process as in inlined flow



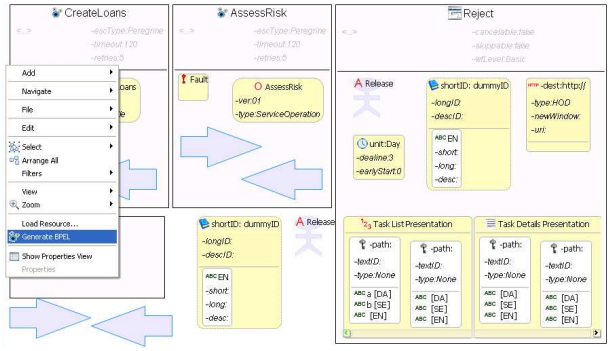
(k) Tool utility to create an empty process fragment for a model injection. The modeler has selected at the connection between the start node and the AssessRisk task as injection point by right-clicking on it



(l) Process fragment to be used as injection. The solution architect has modeled it.

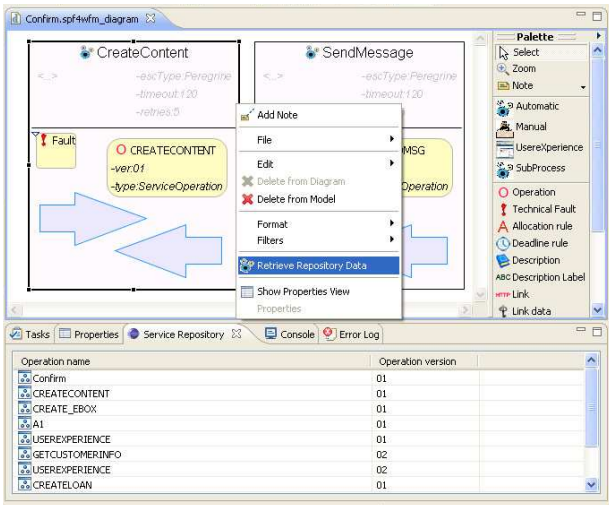


(m) Generated Workflow Specification (WFSPec) model for the injection process fragment

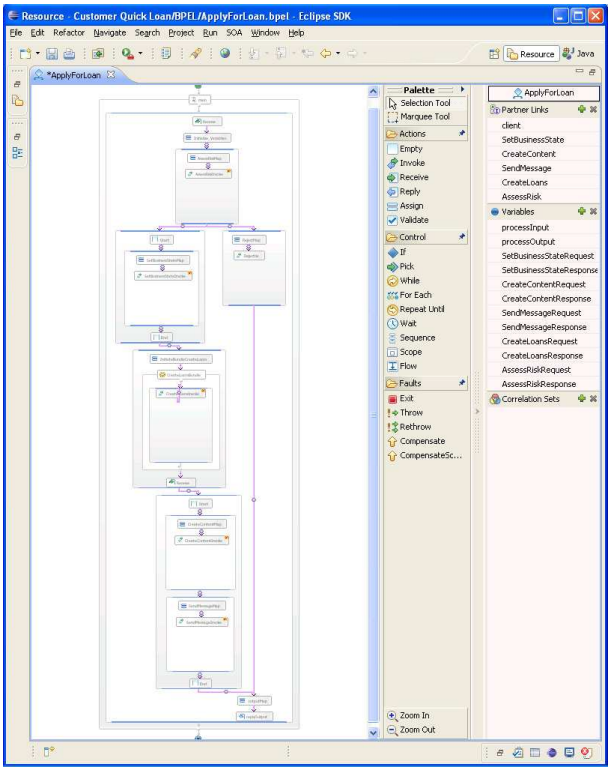


(n) Utility tool to generate BPEL implementation. The modeler has right-clicked at the canvas

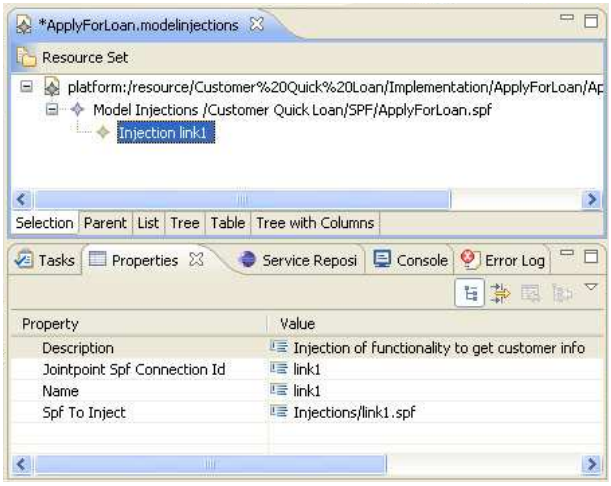
Figure 19: Using Danske Bank Workbench (cont'd)



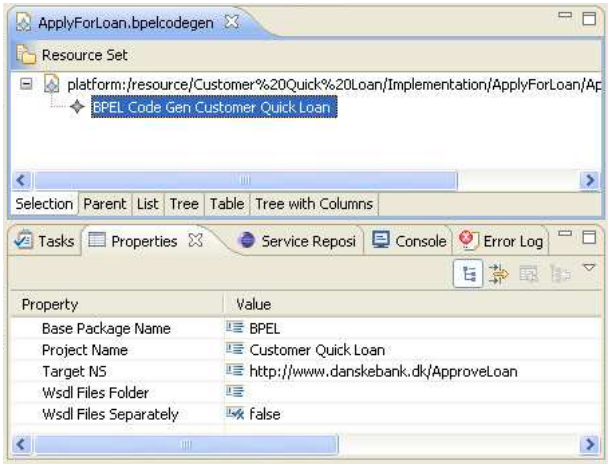
(o) Too utility to retrieve data structures from the enterprise Service Repository into the WFSpec model. The modeler has right-clicked at an AutomaticSpecification element



(p) BPEL code generated from the Solution Process Flow model, the WFSpec model, the ModelInjections model and the BPELCodeGen model



(q) ModelInjections model viewed by the default EMF editor



(r) BPELCodeGen model specifying BPEL specific values for the implementation

Figure 19: Using Danske Bank Workbench (cont'd)