VxBPMN Designer: A Graphical Tool for Customizable Process Models Using the PVDI Framework

Piet den Dulk
Master's thesis

Computing science, University of Groningen
Supervisor: Prof. Marco Aiello
Co-supervisor: Prof. Alexandru C. Telea

November 29, 2013

Abstract

In today's industry, where fast and efficient delivery of high-quality products and services is a demand, business processes are at the core of medium and large-scale organizations. Assisted by IT systems, business processes coordinate human and automated activities, to fulfill the goal of products and service delivery. Since business processes reflect organizational operations, business processes need to stay in synchronization with the constant changing environment of an organization. However, many standards and technologies used in today's industry offer little support for management of change and variability in business processes. In addition, some organizations are structured with a separation of authority. One such organization is the government and its connected municipalities. Municipalities have to implement processes to support their (digital) services. These processes have to adhere to changing laws established by the government. In addition, municipalities greatly differ from one another. This thesis presents a graphical tool for modeling business processes, enhanced with template design and formal verification of process variants. The purpose is to provide a demonstration of the Process Variability - Declarative'n'Imperative (PVDI) framework. The PVDI framework enriches a process modeling language with syntax for expressing variability as constraints. In addition, the PVDI framework provides algorithms for verification of process variants derived from templates. The tool demonstrates that business processes can be templatized, customized and checked on correctness using formal verification. Additionally, the tool is compatible with other process modeling tools and business process management systems such as Cordys BOP. This compatibility allows to simulate a process management life cycle that includes variability modeling and process customization. Compatibility with Cordys BOP is achieved by using the Business Process Model and Notation for design of process models, and the XML Process Definition Language for interchange of process models among different tools.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Research Question . . . 3
- 1.4 Thesis Organization . . . 4

2 Context **5**
- 2.1 The SAS-LeG Project . . . 5
- 2.2 Scenario: Subsidized Wheelchairs . . . 6

3 Background **8**
- 3.1 Business Processes . . . 8
  - 3.1.1 Business Processes in Context . . . 8
  - 3.1.2 Business Process Model and Notation . . . 10
  - 3.1.3 XML Process Definition Language . . . 11
  - 3.1.4 Business Process Management . . . 11
- 3.2 Web Services . . . 14
- 3.3 Business Process Variability . . . 15
- 3.4 Computation Tree Logic($^+$) . . . 16

4 The PVDI Framework **20**
- 4.1 Formal Process Definition . . . 20
- 4.2 PVDI Templates . . . 21
- 4.3 Flow Constraint . . . 22
- 4.4 Parallel Constraint . . . 24
- 4.5 Group Constraint . . . 25
  - 4.5.1 Frozen Group . . . 25
  - 4.5.2 Semi-frozen Group . . . 26

5 Realization **28**
- 5.1 Architecture within the SAS-LeG Context . . . 28
  - 5.1.1 The Process Pipeline . . . 28
  - 5.1.2 System Overview . . . 29
- 5.2 Functional Requirements . . . 32

5.2.1 Use Case Diagram: System 34
5.2.2 Use Case Diagram: Process Modeling Tool 34
5.3 Development Challenges 36
5.4 Implementation 38

6 Software Prototype **39**
6.1 VxBPMN Designer 39
6.2 Demonstrating PVDI's Template Features 41
6.2.1 Flow Constraint 41
6.2.2 Parallel Constraint 44
6.2.3 Frozen Group 44
6.2.4 Optional Activities 47
6.2.5 Weak Link 50
6.2.6 Floating Activities 52
6.3 Tool Compatibility 55

7 Conclusions **60**
7.1 Future Work 63
7.1.1 Research Opportunities 63
7.1.2 Tool Improvements 65

Bibliography **67**

List of Figures

2.1 Generic process model[1]. 7
2.2 Process variant [1]. 7

3.1 A formal structure of a business process [22]. 10
3.2 Core set of BPMN elements [15]. 11
3.3 The BPM life cycle [35]. 12
3.4 The model checking approach [26]. 17
3.5 Semantics of CTL visualized [2]. 18

4.1 Flow constraint [14]. 24
4.2 Parallel constraint [14]. 24
4.3 Frozen group (left) and semi-frozen group (right) [14]. 25

5.1 The process pipeline in e-Government. 30
5.2 Deployment diagram of the SAS-LeG architecture. 33
5.3 Use case diagram: system. 35
5.4 Use case diagram: process modeling tool. 36

6.1 VxBPMN designer. 40
6.2 Flow constraint between activities a and b. 42
6.3 Flow constraint not satisfied. 43
6.4 Parallel constraint between activities a and b. 45
6.5 Parallel constraint violated. 46
6.6 Frozen group. 48
6.7 Frozen group violated by activity g. 49
6.8 Activity b is optional. 50
6.9 Optional activity b removed. 51
6.10 Semi-frozen group with two weak links. 53
6.11 Weak link allows insertion of activities. 54
6.12 Activity b is floating. 56
6.13 Floating activity b can be moved. 57
6.14 Imported process model in Cordys BOP. 58
6.15 Imported process model in Bizagi process modeler. 59

Chapter 1

Introduction

Ever since the early days of manufacturing, efficiency of production is key to success. In the industrial era, pioneers like Adam Smith and Frederick Taylor introduced scientific methods for optimization and management of manufacturing processes. Striving for efficient processes, leads to faster production, less resources, less waste, faster time-to-market and ultimately to competitive advantage and revenues.

1.1 Context

The digital era and today's worldwide economy has led to a wide range of new (business) opportunities. Business processes are at the core of medium and large-scale organizations, and IT supported infrastructures of these organizations are highly process driven [1]. Hammer and Champy [16] define a business process as: "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer." Examples of business processes are: purchase orders, flight booking, stock control and financial transactions.

Various languages exist for designing and executing business processes, such as the Business Process Model and Notation (BPMN) [15], Event-driven Process Chains (EPCs) [19, 38] and the Business Process Execution Language (BPEL) [24]. BPMN is one of the prominent languages for design of process models in the business domain [29]. However, most process languages such as BPMN, EPCs and BPEL lack support for modeling parts of a process that are subject to change [38, 42]. Whenever a change is required, process models need to be changed manually. Consequently, either all variations are modeled in one monolithic process model, or many coexisting but similar process models need to be maintained simultaneously [42]. Since process models represent the operations of an organization, a (subtle) change can have significant impact on the entire organization and

beyond [40]. Fast response to change is difficult when business processes are pervasively implemented. Preferably, business processes need to stay synchronized with the organization and adapt to changing requirements, such as new business opportunities, customer demand, legislation and organizational restructuring. Process models become unmanageable without explicit support of variability.

According to Santos, et al. [32], "variability in business process models, consists of defining alternative paths of execution", and occurs in two dimensions [42]. First, variability occurs over time, which is usually caused by changing requirements, resulting in process evolution. Second, variability occurs within a domain space, which is usually caused when offering customized products and services, resulting in a process family. The ability to express explicit variability in process models, requires that a process language has syntax for specifying alternative paths of execution.

National governments and their connected municipalities, face the challenge of reusing process models as much as possible while allowing a degree of customization, such that different municipalities can offer the same service while adhering to a national legislation. Each municipality may differ in size, resources and IT systems. In addition, there are (subtle) differences of how legislation is allowed to be implemented. Since each municipality is different, no manageable process model suits all municipalities. While the overall structure of a process model must be preserved, variations are required such that a process model can be adapted to the needs of any municipality. Preserving the overall structure of a process model while allowing a degree of variation, requires separation of process modeling in to, design and customization of process models. Furthermore, other domains such as health care and the automotive industry, have similar scenario's [1, 42].

1.2 Problem Statement

Most process languages such as BPMN and EPCs were not designed with variability modeling in mind. To account for variability, either a new process language has to be developed from scratch, or an existing process language needs to be extended with syntax for variability modeling. Various techniques to address process variability have been proposed in literature, such as feature diagrams and variation points (Feature-EPC [42], VxBPEL [4]), inheritance (C-EPCs [37]), Petri nets [46] and declarative specifications (ConDec [27], DECLARE [28]). However, none of the proposed techniques have yet established into an industry standard. In addition, process languages exist in design, execution and interchange languages [20]. Obviously, the need for process variability affects business process information systems, as these systems were developed without support for (standardized) explicit

variability management.

A common problem of modeling variability in business process models is: a) to manage large sets of different versions of a process model, and b) to deal with unforeseen situations and changing requirements. A method for business process variability is the use of a reference process such as C-EPCs [31, 40]. A reference process is a generic model, allowing customization of specific process models. Many implementations such as C-EPCs, Feature-EPC [42] and VxBPEL [4] use imperative methods to specify variability. Imperative methods focus on how tasks of a process are performed [1, 14]. In practice, imperative variability requires to specify all possible variations in advance [14]. Imperative variability is far from flexible, since theoretically, a reference process needs to hold an infinite amount of possible process variants. Flexibility of process models is required for the government and municipalities scenario (see Section 1.1). In such scenario's, many different versions of a process model exist, since each municipality is different. Using an imperative method for modeling variability is practically infeasible when a process designer has to specify large sets of differences.

1.3 Research Question

Software As Service for the varying needs of Local e-Governments (SAS-LeG) [25], is a project that aims to improve (digital) service offering of the Dutch municipalities, by proposing software services and business process technology. Within the SAS-LeG project, Aiello, et al. [1] compiled a list of requirements for explicit process variability. The requirements are categorized by: expressive power, variability techniques, service requirements, consistency and fault handling, and evolution requirements. Moreover, in [1] different tools and frameworks such as VxBPEL [4], ADEPT [7] and DECLARE [28] are evaluated and compared with respect to the requirements for explicit process variability. In conclusion, there are no frameworks nor tools available that support all listed requirements of variability management [1]. Groefsema, et al. [14] proposed the Process Variability - Declarative'n'Imperative (PVDI) framework, which focuses on a subset of the listed requirements from [1], that relate to process modeling. Similar to imperative methods (see Section 1.2) for process variability, the PVDI framework also works with a reference model. However, in contrast to an imperative method, the PVDI framework offers a higher degree of flexibility, by combining imperative and declarative approaches as a specification for modeling of process variability. Instead of specifying all alternative tasks in a process model using an imperative method, a declarative specification defines rules for what must be satisfied, limiting the boundaries of a process model [1]. A tool for demonstrating and testing the PVDI framework, however, is still

missing. Furthermore, variability management requires change to the life cycle of business process management [1].

The main research question of this thesis is:
Would a software prototype, demonstrate the PVDI framework as a candidate solution for modeling explicit variability in business process models?

From this research question, the following sub questions are derived:

1. *How would process variability using the PVDI framework change the standard life cycle of business process management?*
2. *What is needed to achieve compatibility with other process tools, such that the business process life cycle incorporating the PVDI method can be simulated?*
3. *Does an automated model checker confirms formal verification of the PVDI framework on customized process models?*

The objective of this thesis is to introduce a software prototype of a process modeling tool that implements the PVDI framework.

1.4 Thesis Organization

This thesis is organized as follows. First, in Chapter 2 the SAS-LeG project is introduced, followed by a scenario. The scenario illustrates the problem statement, and is used as an example throughout this thesis. In Chapter 3, background information is given about: business processes, Web services, business process variability and Computation Tree Logic$^+$. The state of the art is provided in Chapter 4, which explores the PVDI framework. First, a formal definition of a process is given, followed by an early specification of the PVDI framework. Chapter 5 addresses the realization of the solution and a software prototype is showcased in Chapter 6. Finally, in Chapter 7 the research questions are answered and challenges for future work are presented.

Chapter 2

Context

Because this thesis contributes to the SAS-LeG project, the objective of the SAS-LeG project needs a brief introduction. The problem faced by the Dutch government and its connected municipalities, is then illustrated with a scenario.

2.1 The SAS-LeG Project

Software As Service for the varying needs of Local e-Governments (SAS-LeG) [25] is a project that aims to improve (digital) service offering of the Dutch municipalities. The project is funded by the Netherlands Organization for Scientific Research, and is a collaboration between the University of Groningen, Cordys and several municipalities of the north of the Netherlands.

Municipalities have to implement national laws, to service their citizens. Currently, municipalities are responsible for their own activities to support these national laws. Each municipality uses a lot of resources to implement and maintain the national laws. Some municipalities may have processes that coordinate their activities, whereas others still deliver their services with ad hoc performed activities. Because each municipality implements the same national law, the activities and processes that support the law, share a lot of commonalities. Whenever a law is changed, or a new law is prescribed by the Dutch government, each municipality implements it individually. Since the Netherlands has more than 400 different municipalities, a lot of work is done redundantly. This inefficiency calls for a solution, such that national laws are uniform and correctly implemented by all decentralized municipalities. Section 2.2 illustrates a scenario of this problem.

The SAS-LeG project proposes Software as a Service (SaaS) to improve (digital) service offering of the Dutch municipalities [25]. SaaS is an approach

to stimulate reuse of (distributed) IT components and legacy systems. The objective is to implement the law once and offer it as a customizable service [25]. A customizable service can then be aligned to the organizational structure of each municipality. Section 3.2 explains (digital) services in more detail.

2.2 Scenario: Subsidized Wheelchairs

The Netherlands counts more than 400 different municipalities. These municipalities all vary in size, (human) resources and IT systems. In addition, municipalities have their own local rules. Yet, each municipality has to implement the same law prescribed by the Dutch government. One such law governs provision of subsidized wheelchairs. With this law, handicapped people can request a wheelchair from their respective municipality. To qualify for a subsidized wheelchair, a request has to meet certain criteria. When a municipality receives a request, a process is initiated and handles the request.

Figure 2.1 depicts a graphical representation of a process for granting a wheelchair. The circle at the left side represents the start of the process, and the circle at the right side is where the process terminates. The rectangle and diamond shapes represent activities, and the arrows represent flow of the process. The process coordinates activities that are executed in a certain order. Activities can be either automated tasks or human activities. For example, an activity could be a "home visit", where an employee visits the person who requested a wheelchair. Eventually, the process terminates and an outcome for granting a wheelchair is based on how the process is executed.

Municipalities greatly differ from each other, and so do their activities. Figure 2.2 depicts a similar, but slightly different process model. The overall structure bears resemblance with the structure of the process model in Figure 2.1, but has a slightly different composition of activities. For example, some municipalities prefer to execute a home visit, whereas others do not. In Figure 2.2 a municipality executes a home visit, whereas another municipality following the process model of Figure 2.1, omits this activity. Another activity where municipalities may differ, is the "refer to CIZ" activity, where an audit on the type and amount of health care is done by an external organization. The last difference between both processes, is a check on the age of a person when a wheelchair is granted for free. As the borderline for age is not prescribed by the law, the condition for it, may differ from municipality to municipality. In the process of Figure 2.1, a person has to be at least 70 years old, whereas in Figure 2.2 a particular municipality had chosen to soften the condition for age, to 65.

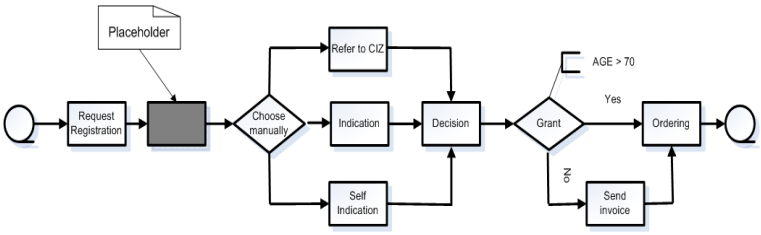

Figure 2.1: Generic process model[1].

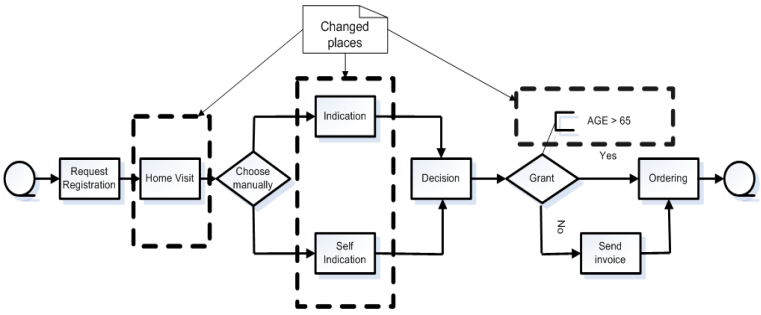

Figure 2.2: Process variant [1].

Chapter 3

Background

This chapter provides background information about: business processes, Web services, and business process variability. Lastly, a temporal logic used for formal model checking called Computation Tree Logic$^+$ is explained, which is the foundation of the PVDI framework (see Chapter 4).

3.1 Business Processes

The general topic of business processes is explored in this section. The following subsections introduce business processes, Business Process Model and Notation, XML Process Definition Language and business process management.

3.1.1 Business Processes in Context

Business processes are at the core of today's medium and large-scale organizations [1]. The outcome of a business process can be a product, service or supporting process. An important factor of managing business activities requires that an organization is conscious about the processes within and across the organizational. A process can be monitored to find bottlenecks for improvement. Management of processes becomes more important as an organization increases in size as a mechanism for strategic and operations management. A lot of work is done redundant without a process to be followed. In literature various definitions of a business process are stated [13, 20]. Besides the definition stated by Hammer and Champy [16] (see Section 1.1), the following two definitions, among more, are used in literature.

According to Davenport [8], "a business process is a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within

an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action."

Smith and Fingar [34] defined a business process as: "a complete and dynamically coordinated set of collaborative and transactional activities that deliver value to customers."

Let's consider a manufacturing process. The execution of a manufacturing process is fixed and known in advance. The steps taken for assembly of a product are exactly followed as the procedure described by the process. Each product of the same type is manufactured the exact same way. By monitoring the production process, the process can be optimized to raise performance, resulting in faster and cheaper production.

Similar to optimization of manufacturing processes, optimizing a business process may lead to efficient use of resources. In contrast to traditional manufacturing processes, however, a business process is of less rigid nature. Business processes have a less predictable character, and require more flexibility. Business processes have to support various forms of communication such as machine to machine communication, human-machine communication and human to human communication. Management of business processes (see Section 3.1.4) requires commitment to capture knowledge of the activities and communication flows within an organization. In contrary to traditional manufacturing processes, a human-centric system controlled by a business process management system, requires a degree of flexibility to accommodate for human work.

Business processes can be either short or long lived. A short-lived business process is entirely composed of automated activities such that it responds within the interaction of a Web form. The activities of a business process are often implemented as Web services (see Section 3.2), and a business process is often invoked as a Web service itself. An example of a short-lived business process is a flight booking service. A flight booking service invokes various other Web services to arrange a flight without human intervention, yet providing the necessary details. Obviously, the execution of a business process for a flight booking service should respond shortly after a customer filled in a Web form with information. A long-lived business process doesn't have to terminate shortly after it is initiated, and may take intervention of human activity. Examples of long-lived business processes are manufacturing processes, product delivery and provision of subsidized wheelchairs.

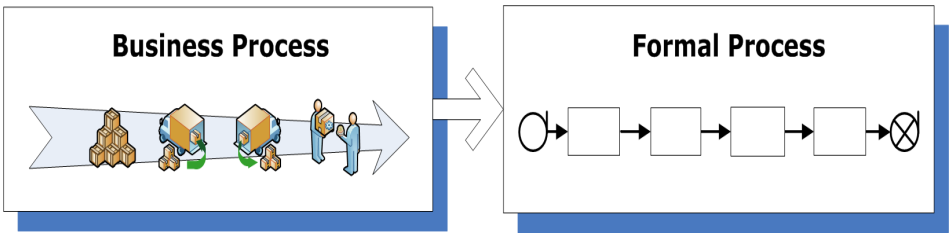

Figure 3.1: A formal structure of a business process [22].

Product delivery, e.g., involves activities such as transportation.

Furthermore, the terms "process" and "business process" are used interchangeable in the field of business process management. Ryan K. and L. Ko [20] state that, "a process can be any human endeavor. A business process is a process within a business context." In this thesis the term "process" is generally used and the term "business process" is used when specifically referring to a process in a business context. For the PVDI framework (see Chapter 4) a formal definition of a business process is required. Figure 3.1, depicts an abstraction from a specific business process model. The abstraction has a more general structure that can be formalized by a data structure. Section 4.1 introduces a formal definition of a (business) process.

3.1.2 Business Process Model and Notation

Traditionally, business and IT staff have different viewpoints of business processes and did not share the same vocabulary, which often causes a mismatch in communication [20]. The Business Process Model and Notation (BPMN) [15] is a standard specified by the Object Management Group (OMG) for designing Business Process Diagrams (BPD's). With BPMN, different professionals such as business analysts, managers and IT staff [20, 21, 33] have a shared vocabulary. BPMN is one of the most popular languages for modeling BPD's [29] and is supported by many process modeling tools [21, 29] such as Cordys BOP [6], Bizagi Process Modeler [3], Intalio BPMS [17] and Tibco Business Studio [36].

Figure 3.2 depicts the core set of BPMN elements where flow objects (nodes) and connecting objects (edges) are the most used modeling constructs of the language. Nodes represent the tasks of a business process, which can be activities, gateways or events. Tasks are the atoms of a process model, and are interconnected by connectors, which specify flow. Furthermore, BPMN has swim lanes and grouping elements used to organize flow objects into organizational units and categories respectively. However, BPMN lacks support of business execution rules [29], nor is a machine readable format (originally) prescribed. A standardized machine readable format is required for interop-

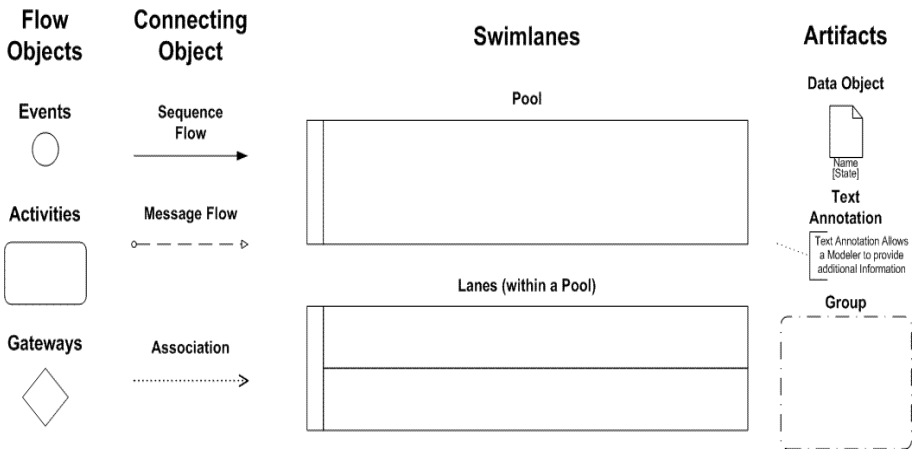

Figure 3.2: Core set of BPMN elements [15].

erability of process modeling tools from different vendors. The absence of a format for storing BPD's has led to XML-based languages such as XPDL [5] (see Section 3.1.3).

3.1.3 XML Process Definition Language

BPMN (see Section 3.1.2) originally did not come with a machine readable format. Using graphical constructs, the BPMN specification specifies syntax and semantics of how diagrams can be composed. However the specification is independent of the underlying technology used for supporting BPMN diagrams. Specified by the Workflow Management Coalition (WfMC), the XML Process Definition Language (XPDL) [5] is a standardized XML-based format for marshaling business process diagrams with an emphasis on BPMN diagrams. XPDL has support for storing both graphical and execution information of BPMN graphs. With the ability of storing graphical information in XPDL, different BPMN tools can interchange process models. With a standardized format for storing process models, vendors can make their tools compatible with tools and business process platforms from other vendors. Because XPDL is XML based, XPDL can be extended with new features such as constructs of the PVDI framework (see Section 1.3). In addition, XPDL provides functionality for storage of tool specific properties.

3.1.4 Business Process Management

Ad hoc workflows might be sufficient or even an advantage for small organizations. However, process management becomes a necessity for large organizations, to have strategic control over the operations performed. Management of business processes is called business process management (BPM).

Figure 3.3: The BPM life cycle [35].

Van der Aalst, et al. [41] defined BPM as "supporting business processes using methods, techniques and software to design, enact, control and analyze operational processes involving humans, organizations, applications, documents and other sources of information."

The idea behind BPM is to acquire high-level control of the operations within an organization at a management level. A diagrammatic process model provides a bird's-eye view of the activities within an organization. When monitoring a process, bottlenecks can be found, which could indicate opportunities for optimization. Figure 3.3 depicts the BPM life cycle [41]. This figure shows four phases in a closed-loop life cycle that are applied by many BPM adopters. The next subsections briefly introduce each phase of the life cycle that business processes undergo in BPM.

Design and Model

The first step in BPM is to get a diagrammatic overview of the processes running in an organization. A business process diagram is a visual representation of a process model. Before a process model can be designed, the current situation is analyzed first. A business process model can be designed from specified requirements and data collected during observations. In the first iteration of the life cycle, the process model is drawn from scratch. Specific details may remain unknown during this phase. A business architect first draws an initial design on a paper sheet or using a process modeling tool. In subsequent iterations of the life cycle, the process model is revised for optimization.

Process models are designed in a (standardized) modeling language such as BPMN and EPC, which let different experts work on the same process models. A wide variety of graphical modeling tools exist for design of business process models [21, 29], such as Cordys BOP [6], Bizagi Process Modeler [3], Intalio BPMS [17] and Tibco Business Studio [36].

Develop, Test and Deploy

A business process diagram is a mere visual representation of a process model. Therefore, a process model needs to be coupled with executable tasks, such that a process can interact with real world activities. Executable tasks can be either human work or automated activities. During this phase, coupling of executable activities with the respective process model takes place.

A unit of work can be reused when there is clear definition that has: a) an interface or contract representing a unit of work, and b) an implementation of the respective unit of work that conforms its interface or contract. The implementation of a unit of work can be an automated or a human activity. For example, at some point during execution of a business process, an employee performs some activity and has to fill in a (digital) form with the data gathered during that activity. The digital form acts as an interface between the human activity and the business process in execution. Another example is a Web service [45] (see Section 3.2) that is invoked via an interface such as WSDL [44].

Furthermore, most BPM platforms such as Cordys BOP [6], offer simulation tools such that process models can be tested. The flow of communication must be accurate between different executable tasks, as the output from one activity of the process is input to another activity. Finally, when the test cases are successfully passed, a process model can be deployed for actual usage.

Execute

A deployed business process model coordinates the various activities of an organization. When a business process is started, an instance of the business process model is created. The instance is the actual execution of a process. Multiple instances of the same business process can coexist and are handled in parallel. For example, consider a business process for a subsidized wheelchair (see Section 2.2). When a citizen requests a wheelchair, an instance of the respective business process model is created. When another citizen requests a wheelchair, a second instance of the same business process model is started for that request. Each request is handled by a separate instance of a process model. The amount of process instances depends on the amount of requests to be processed.

Analyze and Optimize

When instances of a process model are running, the execution of these processes can be closely monitored. Also, the state of a process instance can

be analyzed. Quantitative data can be collected when multiple instances are running. With quantitative data, statistics can be applied to analyze the efficiency and effectiveness of the respective process. When analyzing a process, bottlenecks can be found, which might indicate opportunities for optimization. Moreover, as requirements change and new technologies emerge, these changes might affect the process model. The analyzed data acquired during this phase can then be used as input for the next iteration of the process life cycle as the closed-loop in Figure 3.3 suggests.

3.2 Web Services

The SAS-LeG project (see Section 2.1) proposes SaaS as a software technology model, to implement the law once and offer it as a customizable service to the municipalities of the Netherlands [25]. SaaS is a software delivery model, which is often implemented using Web service technology. Web services [45] are loosely coupled software components distributed over a network such as the Internet. "Loosely coupled", means that software systems using Web services, are not dependent on the implementation of a Web service. Instead, systems that use Web services, only know the interface of a Web service. The interface describes the functionality of a Web service. The advantage of loose coupling, is that the implementation of a Web service can be replaced by another implementation, without any modification required on dependent software systems that utilize Web services.

The interface of a Web service is often accomplished trough the Web Service Description Language (WSDL) [44]. WSDL is an XML-based language for describing the location and functionality of a Web service. Similar to interfaces used in programming languages, a WSDL file contains declarations of functions that are implemented by the respective Web service. A standardized format for accessing Web services is the Simple Object Access Protocol (SOAP) [43]. With SOAP, data can be interchanged between different information systems without ambiguity, since the type of data is specified in the SOAP standard, implemented by both client and server software.

Web services can be seen as autonomous, distributed building blocks that are hosted on a Web server from which they can be accessed by other software clients simultaneously. An example of a Web service is a service for retrieving weather information. A weather station might wish to make their weather information digitally available to other applications. Web services can be either atomic or a composition of other Web services. For example, an atomic Web service retrieves weather information from a database of the weather station, and then forwards this information to requesting software clients. A decomposed Web service invokes other Web services to create rich/ new content. For example, a Web service for travel information needs

weather information in addition to other information such as destination and date of arrival. The travel information service is a composite Web service composed from other atomic and composite Web services. The travel Web service on its turn can be used by different travel agencies.

3.3 Business Process Variability

Current BPM standards and technologies offer little support for variability of process models [40]. Business activities within (large) organizations are difficult to maintain and change. As described in Section 3.1.4, the BPM life cycle starts with design of process models. However, most process languages such as BPMN and EPC offer little to no support for design of change and variation of process models [38, 42]. As a result, process models are limited in their configuration [40] abilities. Process languages without support of explicit variability, are called non-configurable process languages [42]. Without functionality for explicit design of variable activities, the final process model has a rigid structure. This rigid process model is passed on to the next phase in a process life cycle. Responding to change becomes problematic when rigid process models are propagated through the process life cycle. When a process model finally reaches an operating environment, execution of a business process can be predicted in advance [1, 30, 39]. Prediction of process execution is a desirable requirement for BPM, but when processes are pervasively implemented, alignment with a constant changing environment becomes difficult.

With non-configurable process languages such as BPMN and EPC, variability is implicitly designed within the process model via the use of choice constructs such as OR gates. Alternatively, slightly different but similar process models are designed [42] to cope with variability. The first method leads to an incomprehensible process model, whereas the latter method results in maintenance problems. A process increases in size and complexity when all variations are designed in one large process. When a process has to incorporate all different variations, a process model increases complexity, which affects readability. The latter case leads to maintenance problems when having several, but similar process models.

Santos, et al. [32], state that: "variability in business process models, consists of defining alternative paths of execution" and occurs in two dimensions [42]. First, variation occurs over time resulting in process evolution. Process evolution happens when a process model needs to be changed while already being in operation. For example to improve a bottleneck in the process or a change in legislation requires a change in the process. The second dimension is variability within the process model resulting in a process family. Management of variation in these two dimensions, enables fast response to:

product/ service demand, innovation and change in regulations.

Another distinction of variability can be made between design and run-time process variability. Design-time variability requires expressive power and ease of use for designing process families. The designer of a process model can foresee certain variations of the process and models the variable parts accordingly. Run-time variability concerns process execution models, instead of process diagrams. On the execution level, variation exposes that cannot be foreseen by the process designer. The execution aspects concern how activities should be performed, whereas variation of process diagrams concern what should be done. Process modelers are more interested in what should be done than the details of how activities should be done. Service replacement is an example of run-time variability. When a Web service is unavailable, the process should be able to make a request to the same type of Web service from another service provider. Run-time variability covers technical issues such as service availability, whereas the modeling part covers the aspect of what one wants to see the process should be doing. This thesis focuses on design-time process variability.

3.4 Computation Tree Logic(+)

A useful component for verification of process models that have to obey national laws, is a model checker. A model checker verifies whether a model satisfies a given specification. Figure 3.4 illustrates the basic approach of formal model checkers. Model checkers are often used for systems that have safety-critical requirements, where certain properties of a system such deadlocks and race conditions are undesired.

Temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [10], are often at the core of advanced model checkers. CTL is a branching-time logic in which statements over branching paths of time can be expressed. A process model can be formalized as a directed graph [14] where activities can be represented by nodes and flows can be represented by directed edges. An example of a formalized process model is depicted in Figure 3.1. More specifically, a process model resembles a state machine where activities correspond to states and flows to (labeled) transitions. This similarity makes CTL a candidate logic for checking branching time conditions of process models. CTL is introduced here, since a variant of CTL called CTL$^+$ [11], is used by the PVDI framework (see Chapter 4) by Groefsema et al. [14]. The PVDI framework is the core of the tool presented in this thesis (see Chapter 6).

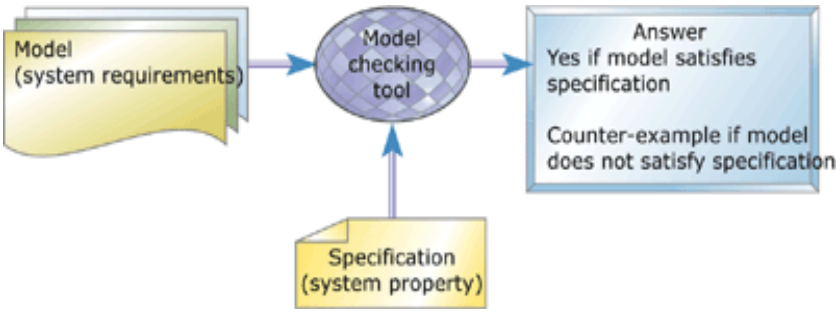

Figure 3.4: The model checking approach [26].

Syntax

CTL formulas are created over models that have the form of a state transition system such as a Kripke structure. A Kripke structure is a state transition system with a labeling function. A Kripke structure is described by a triple $\mathcal{M} = (S, \rightarrow, L)$ where S is the set of states and $\rightarrow$ is the set of transitions specified as $\rightarrow \subseteq S \times S$ and where $L : S \rightarrow 2^P$ is a labeling function with P the set of atomic propositions. A directed graph such as a business process is represented by a pair $(S, \rightarrow)$. The following BNF grammar specifies the syntax for creating CTL formulas:

$$\begin{aligned} \phi \quad ::= \quad & p, q \mid \bot \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \\ & AX\,\phi \mid EX\,\phi \mid AF\,\phi \mid EF\,\phi \mid AG\,\phi \mid EG\,\phi \mid \\ & A\,[\phi\,U\,\phi] \mid E\,[\phi\,U\,\phi] \end{aligned}$$

Formal verification is accomplished via a relation between model $\mathcal{M}$ and specification ϕ. The relation between a model and CTL formulas is defined by the satisfaction relation: $\mathcal{M}, s \models \phi$ with $s \in S$ and $\phi \in \Phi$ where Φ is a set of well formed CTL formulas, and p and q are atomic propositions. The satisfaction relation, couples model and specification (given as a set CTL formulas).

Semantics

The letters A and E represent state quantifiers and correspond to the predicate logic quantifiers: "All" and "Exists", which are often denoted by the symbols $\forall$ and $\exists$. Bound to these quantifiers are the unary operators: F, X, and G and binary operator U. These operators are path operators and respectively mean "Finally", "neXt", "Globally" and "Until". The unary path operators: F, X and G respectively correspond to the modal operators: $\Diamond$ (possibly), $\Box$ (necessarily) and $\bigcirc$ (obligatory) from modal logic. In CTL the path operators are interpreted as modalities of time.

Figure 3.5 depicts the semantics of the temporal quantifiers of CTL in a

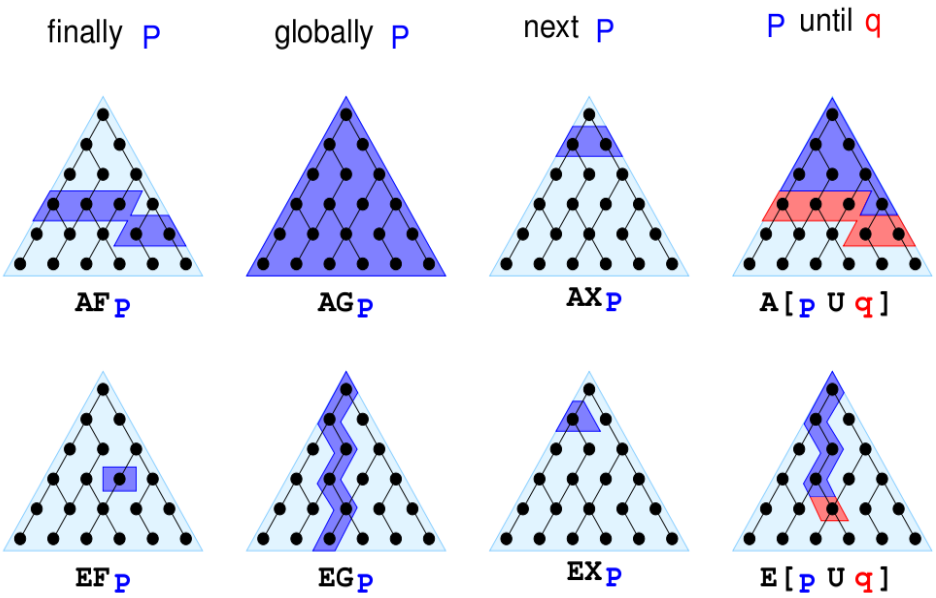

Figure 3.5: Semantics of CTL visualized [2].

visual manner. In this figure, the concept of each branching-time formula from the grammar is visualized. An example is given for each basic CTL formula applied on a model visualized as a tree structure. State quantifiers are visualized in the width of a tree structure whereas path quantifiers are expressed in the length. The A quantifier states that its argument is true for all paths, whereas the E quantifier states that its argument is true in some path. The path quantifiers F, X, G and U are statements over paths. For example, formula EFp states that proposition p should eventually hold somewhere in a path along any of the paths. In contrast, formula AFp states that proposition p should eventually hold somewhere in all paths.

Computation Tree Logic$^+$

A variant of CTL called Computation Tree Logic$^+$ (CTL$^+$) [11] is used as specification logic for the model checker proposed by the PVDI framework (see Chapter 4). CTL$^+$ has exactly the same expressive power as CTL but CTL$^+$ formulas can be written more compact and are easier to understand than their equivalent CTL counterparts [18]. Emerson and Halpern [11] define the language of CTL$^+$ as follows:

1. Each primitive formula is a state formula.
2. If p, q are state formulas, then so are $(p \wedge q)$ and $\neg p$.
3. If p is a state formula, then Xp, Fp and Gp are path formulas (which intuitively say that at some state (resp. the next state) on the path p

holds).

4. If p is a path formula then Ep is a state formula (which says some path satisfies p).
5. If p is a path formula then Ap is a state formula (which says all path satisfies p).
6. If p, q are state formulas then $(p\,U\,q)$ is a path formula (which says there is some state on the path which satisfies q, and all states before it satisfy p, i.e., p holds until q).
7. If p, q are path formulas, then so are $p \wedge q$ and $\neg p$.

In BNF grammar form, the syntax of CTL$^+$ is as follows, where p and q are atomic propositions and φ and ψ are state and path formulas respectively:

$$\begin{aligned} \varphi &::= p, q \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid A\,\psi \mid E\,\psi \\ \psi &::= p, q \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid X\,\varphi \mid F\,\varphi \mid G\,\varphi \mid \varphi\,U\,\varphi \end{aligned}$$

A difference of CTL$^+$ compared to the grammar of CTL, is that state and path formulas are separated in CTL$^+$ whereas in CTL path quantifiers are bound to state quantifiers. For example, CTL$^+$ allows statements like: $E(Xp \wedge Fq)$, whereas CTL would only allow statements like $EXp \wedge EFq$ due to the bound path quantifiers.

Chapter 4

The PVDI Framework

Chapter 2 and Section 3.3 issued the need for expressing variability in process models. To apply formal model checking methods on process models, a formal definition of a business process is required. Section 3.1.1 provides two definitions of a business process. These definitions, however, are not formal and are slightly different. A formal definition of a business process needs to have a form that matches a specification language for formal model checking. Section 3.4 describes CTL$^+$ as a language for specification and verification in formal model checking systems. This chapter describes the Process Variability - Declarative'n'Imperative (PVDI) framework proposed by Groefsema, et al. [14]. The PVDI framework uses CTL$^+$ as a language for formal model checking. For each constraint feature of the PVDI framework, a formal definition and a CTL$^+$ formula generation procedure is given in the next sections.

4.1 Formal Process Definition

Groefsema, et al. [14] use the following formal definition of a process as a formal model and use CTL$^+$ for specification and verification acting on a formal process model. A formal process model is a directed graph defined as [14]:

Definition 1 *A process P is a tuple (A, G, T) where:*

- *A is a finite set of activities with a start $\odot$ and end $\otimes$ activity;*
- *$G = G_a \cup G_o \cup G_x$ is a set of gateways consisting of and, or, xor types respectively;*
- *$S = A \cup G$ is a set of states;*
- *$T = T_a \cup T_g$, where:*
- *$T_a : (A \setminus \{\otimes\}) \rightarrow S$ is a finite set of transitions, which assigns a next state for each activity;*

– $T_g : G \rightarrow 2^S$ *is a finite set of transitions, which assigns a non-empty set of next states for each gateway.*

To prepare a process model for CTL$^+$ model checking, a process P needs to follow the form of an abstract model $\mathcal{M} = (S, T, L)$. Process P has the same form as an unlabeled state transition system (S, T) where S is a set of states and $T \subseteq S \times S$. If the arguments A and G of process P are replaced by the set of states $S = A \cup G$ then tuple (S, T) remains, which corresponds exactly to the description of an unlabeled state transition system.

To make a process compatible with model $\mathcal{M}$, a labeling function is required. Groefsema, et al. [14] use the natural valuation function as a labeling function where a dedicated variable is used for each state (an activity or gate in a business process model). A variable is valuated to TRUE when its corresponding state is reached in that state only. Adding the natural valuation function as a labeling function L to process P gives tuple (S, T, L), which makes a process model compatible with model $\mathcal{M}$ for CTL$^+$ (see Section 3.4).

Furthermore, Definition 1 abstracts the textual definitions of a business process (see Section 1.1 and 3.1.1) to a generic process description. Only the structural formalism of a process model is required, such that a subset of BPMN fits this definition. Definition 1 can function as a meta-model for any process language. In essence, any process language that follows, or can be transformed to, the structure of Definition 1, can be used.

4.2 PVDI Templates

A template defines a family of similar, but different process models. Using Definition 1 as a formal process description, a process template that encompasses a process model can now be formalized. A process template consists of (partial) process model and a set of constraints. Using constraints, a process template determines the outcome space for the members of a process family. A process family is the set of all process variants satisfying the constraints from the process template. Constraints are the boundaries of a process template, which restrict the possible process variants that can be derived from the respective template [1]. Constraints are translated to a set of CTL$^+$ formulas (see Section 3.4), that ensure the correctness of customized process variants. Each constraint has a graphical representation, which hides the CTL$^+$ formulas from the template designer. The semantics of constraints relate to expressing variability in process models. Groefsema, et al. [14] define a constraint as:

Definition 2 *A **constraint** over process P is a Computation Tree Logic$^+$ (CTL$^+$) formula. A constraint is valid for a process P iff it is valuated*

to TRUE in each state of the process under the natural valuation. More formally, let ϕ be a constraint, $\mathcal{M}$ be a model built on the process P using the natural valuation, and S be the set of states of process P. Then $\mathcal{M}, x \models \phi$ $\forall x \in S$.

Definition 2 is an abstraction of the constraints that are defined in the following subsections. A template is a (partial) process description extended with constraints. A template is formally specified as [14]:

Definition 3 *A template R is a tuple (A, G, T, Φ) where:*

- *A, G, S and T are from Definition 1;*
- *$T_a : (A \setminus \{\otimes\}) \rightarrow S$ is a finite set of transitions which assigns a next state for **some** activities;*
- *$T_g : G \rightarrow 2^S$ is a finite set of transitions which assigns a non-empty set of next states for **some** gateways;*
- *Φ is a finite set of constraints.*

Template R from Definition 3 is an extension to a process description with a set of constraints Φ added to the tuple of process P from Definition 1. In addition, the rules for the transition sets T_a and T_g from process P are softened in template R. In process P an activity needs a transition to another next state, and a gateway must be connected to a non-empty set of next states. This requirement is no longer necessary for process templates, because a template does not require a fully specified process model [14]. A process model embodied in a template, however, can serve as the recommended process model. Because of the softened rules of the transition sets T_a and T_g, a process model in a template can vary from a sparse structure requiring a lot of customization, to a fully specified process model with little to no customization at all. The flexibility offered by a template allows for many ways of expressing variability such as declarative and imperative variability techniques.

A model and specification are required input for a formal model checker as Figure 3.4 illustrates. A template captures both input information: a (partial) process model and the specification in form of constraints. CTL$^+$ formulas are generated from the constraints, to form the specification. After customization of a process model, the model checker verifies whether the customized process model satisfies the specification from the CTL$^+$ formulas.

4.3 Flow Constraint

As explained in Section 4.2, a template holds a (partial) process model P and a set of constraints Φ. A constraint ϕ is an abstraction of specific constraints

such as a flow constraint. A constraint restricts the boundaries of a process model [1]. A flow constraint is a relation between sets of source and target elements. A visual representation of the flow constraint is depicted in Figure 4.1, which shows six different forms. A flow constraint is a connecting object, similar to connecting objects from BPMN such as sequence and message flows. The difference with BPMN connectors, is that flow constraints are part of a template whereas BPMN connectors are part of a process model. A flow constraint, constrains the customizations of the flow in a process model. For example, a flow constraint allows a template to either mandate or exclude certain set of target activities to be followed from a source activity or set of source activities. Definition 4 formally describes a flow constraint [14]:

Definition 4 *A flow constraint is a tuple* $F = (S, T, \Omega, \Pi, N)$ *where:*

- S *is a set of source elements;*
- T *is a set of target elements;*
- $S \bigcap T = \emptyset$*;*
- $\Omega \in \{A, E\}$ *is a state quantifier from CTL*$^+$*;*
- $\Pi \in \{X, F, G\}$ *is a path quantifier from CTL*$^+$*;*
- $N \in \{TRUE, FALSE\}$ *is a negation.*

A flow constraint is a relation between a source and target sets of elements, two single elements or a combination of a single element and a set of elements. Elements in both sets can be of any node object from a process such as the flow objects (activities, gates and events) from BPMN. A set of elements is graphically represented as either a BPMN group or a PVDI group constraint (introduced in Section 4.5). A group allows grouping of BPMN elements.

A flow constraint has six different forms (see Figure 4.1) and the type is described by the arguments Ω, Π and N. Arguments Ω and Π represent the state and path quantifiers from CTL$^+$ respectively. Figure 3.5 depicts a visual interpretation of the CTL$^+$ semantics with the combinations of arguments Ω and Π. Argument N specifies whether a target set T should be either followed or prohibited to be followed from a source set S. Depending on the form of a flow constraint (described by its arguments) a corresponding CTL$^+$ formula can be generated as follows [14]:

1. Let $s_1, \ldots s_n$ be elements of S and $t_1, \ldots t_m$ be elements of T.
2. If $N = TRUE$ then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega\Pi(t_1 \vee t_2 \vee \ldots t_m)$.
3. If $N = FALSE$ then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega\Pi\neg(t_1 \vee t_2 \vee \ldots t_m)$.

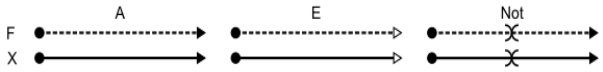

Figure 4.1: Flow constraint [14].

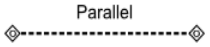

Figure 4.2: Parallel constraint [14].

4. If $N = FALSE$ and $\Pi = F$, then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega G\neg(t_1 \vee t_2 \vee \ldots t_m)$.

4.4 Parallel Constraint

Shown in Figure 4.2 is the parallel constraint. Like flow constraints, a parallel constraint connects two sets of elements. A parallel constraint enforces both sets of elements mutually exclude each other from being followed in the same path. A parallel constraint can be useful in situations where certain activities are prohibited to be followed in the same flow of execution. The parallel constraint [14] is formalized as:

Definition 5 *A parallel constraint is a pair* $P = (S, T)$ *where:*

- S *is one set of states;*
- T *is another set of states;*
- $S \bigcap T = \emptyset$

The CTL$^+$ formulas of a parallel constraint are generated as follows [14]:

1. let $s_1, \ldots s_n$ be elements of S and $t_1, \ldots t_m$ be elements of T.
2. Create a formula: $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow AG\neg(t_1 \vee t_2 \vee \ldots t_m)$
3. Create a formula: $(t_1 \vee t_2 \vee \ldots t_m) \Rightarrow AG\neg(s_1 \vee s_2 \vee \ldots s_n)$

The formula generation procedure for a parallel constraint requires creation of two CTL$^+$ formulas. The formulas are similar to the formula of a flow constraint where $N = FALSE$. The two formulas together mutually exclude the sets to be followed from each other. One formula has a set of elements S on the left hand side and a set of elements T on the right hand side, which excludes any element of set T is followed in a path after set S. The second formula has set T on the left hand side and set S on the right hand side, which excludes elements of set S followed in a path after set T.

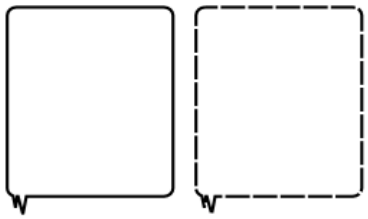

Figure 4.3: Frozen group (left) and semi-frozen group (right) [14].

4.5 Group Constraint

Besides relation based constraints such as the flow and parallel constraint, the PVDI framework offers group constraints, which are area based. A group constraint (Figure 4.3) constrains all elements inside its area. Adding restrictions or allowing modifications on a template are generally accomplished in either two ways. Either, a template allows everything by default and a designer specifies what cannot be changed, or a template disallows all modifications by default and a designer specifies the exceptions that are allowed to be changed. The PVDI framework offers both approaches for designing process templates. By default, a process template allows everything. However, via the use of a group constraint, the opposite can be accomplished, as a group constraint by default disallows all modifications. Since a group constraint is area based, it can embody an entire process model. There are two type of group constraints, frozen groups and semi-frozen groups. The next subsections introduce both type of group constraints.

4.5.1 Frozen Group

Whenever a part of a process model is not allowed to be changed, a frozen group may be used. A frozen group is a type of group constraint that freezes all elements inside its region. All elements inside a frozen group, are finalized and cannot be customized. A frozen group can be compared with the access control modifier "final" from object oriented programming languages. A frozen group can be useful when a designer of a template wants to force certain activities to be untouched in the final process model. The definition of a frozen group is formulated in [14] as:

Definition 6 *A frozen group is a pair $G = (P, M)$, where P is a process (see Definition 1), P_a a set of process activities and $M \subseteq P_a$ a set of mandatory activities.*

Definition 6 introduces another template feature: "mandatory activities". Mandatory activities are elements that cannot be removed from the process model. Since a frozen group cannot be modified, all elements inside

its area, are mandatory. The authors of the PVDI framework refer to a group constraint as a sub-process with its own start and end events. CTL$^+$ formulas for group constraints are complex, as branching paths are involved in the formula generation procedure. The following steps are required for generating CTL$^+$ formulas for a frozen group [14]:

1. For each state a let the chain of states $P = (a_1, a_2 \dots a_k)$ be a path between a and $\otimes$.
2. If path P is empty (i.e. the next state is the final one) then create a CTL$^+$ formula $a \Rightarrow AX\otimes$ where A and X are CTL$^+$ quantifiers. If path P is not empty then advance to the following steps:
3. If path P is the only path from a to $\otimes$, then create a CTL$^+$ formula of the type $a \Rightarrow A((a_1 \vee a_2 \vee \dots a_k)U\otimes)$ where A and U are CTL$^+$ quantifiers.
4. If there are several paths $P_1, P_2, \dots P_t$ which lead from a to $\otimes$, then create the formula $a \Rightarrow A[P_1^T \vee P_2^T \vee \dots P_t^T]$ where $P_i^T = (a_1^i \vee a_2^i \vee \dots a_k^i)U\otimes$ with $a_1^i, a_2^i, \dots a_k^i$ is a chain of path P_i^T.

The generation procedure shows that the formulas make use of branching time logic where both state and path quantifiers are used. Furthermore, the formulas make use of the syntactic separation of state and path quantifiers offered by CTL$^+$ (see Section 3.4).

4.5.2 Semi-frozen Group

By default, a group constraint is frozen and, therefore, doesn't allow any modification to its enclosed sub process. However, a frozen group can be made semi-frozen, which does allow specific customizations to its sub process. The semi-frozen group offers three variability features: optional activities, weak links and floating activities.

Optional Activity

In addition to mandatory activities (see Section 4.5.1), a group constraint can have optional activities. A group constraint is frozen by default with all activities inside, of the type mandatory. Mandatory activities however, can be made optional. Changing an activity from mandatory to optional, makes a group constraint, semi-frozen. The presence or absence of optional activities, does not affect the consistency of a semi-frozen group [14]. Optional activities are allowed to be removed from a semi-frozen group. No changes are required for the formula generation procedure. For example, activity a is made optional, then formulas of the form $a \Rightarrow \dots$ remain valid and formulas like $b \Rightarrow a \vee \dots U\otimes$ remain valid too [14].

Weak Link

Another semi-frozen feature, is the ability to weaken links between two activities. Weak links are specialized flows that allow insertion of activities between its source and target nodes. A weak link is useful in situations where a chain of activities cannot be foreseen by a template designer but the target node must follow the source node. If no activities are inserted, a weak link acts as a normal flow. Since weak links concern paths, step three from the generation procedure of a frozen group needs a slight modification [14].

1. Let the chain of states $P = (a_1, \ldots a_k)$ be a path between a and $\otimes$. For example the link between a_i and a_{i+1} is weak, then the appropriate CTL$^+$ formula is $a \Rightarrow A[(a_1 \vee \ldots a_i)UAFA[(a_{i+1} \vee \ldots a_k)U\otimes]]$.
2. If there are several weak links in a path, for example link $a_i \rightarrow a_{i+1}$ and link $a_j \rightarrow a_{j+1}$, $i < j$ are weak, then create a formula $a_j \Rightarrow \phi$. The final formula is $a \Rightarrow A[(a_1 \vee \ldots a_i)UA[(a_{i+1} \vee \ldots a_j)U\phi]]$, where ϕ is retrieved in the previous step.
3. The same recursive rule applies with three or more weak links.

Floating Activity

A third semi-frozen feature is the floating activity, which fulfills the move and swap usability patterns for process variability. A floating activity can be moved freely over the region of a semi-frozen group, and thus can be moved and reconnected to another position in the sub process. In addition, two floating activities that are in the same semi-frozen group, can be swapped. However, to either move or swap floating activities, a weak link in a semi-frozen group is required. A floating activity can only be inserted to a position with a weak link. Floating activities do not add complexity to the formula generation procedure of for a semi-frozen group. In the CTL$^+$ generation procedure, floating activities are threated as follows [14]:

1. Create the set formulas as described by the algorithms for the frozen/ semi-frozen group;
2. For a floating activity a, omit the creation of the formulas of type $a \Rightarrow \phi$.

When a floating activity is moved, the move is split in two operations [14]: (1) the activity is removed from its previous position, (2) the activity is placed to its new position and reconnected between the two activities that are connected by a weak link. The removal of an activity at the original position, does not conflict during verification, because the formulas of type $a \Rightarrow \phi$ (where a is a floating activity), are omitted during formula generation.

Chapter 5

Realization

Since this thesis involves the presentation of a mid-sized graphical tool, development of a prototype is briefly described in this chapter. The tool implements the features of the PVDI framework described in Chapter 4. Section 5.1 describes a proposal of an architecture envisioned in which the graphical tool takes part of. The functional requirements of the tool are listed in Section 5.2. The challenges of developing a mid-sized graphical tool, are summarized in Section 5.3 and choices of the implementation, are briefly mentioned in Section 5.4.

5.1 Architecture within the SAS-LeG Context

The standard BPM life cycle (see Section 3.1.4), does not incorporate explicit variability. By extending BPM with explicit variability, additional layers of management are required for both design and execution of process models. These additional layers introduce artifacts for process variability with respect to the BPM life cycle. Applying the PVDI framework to BPM, means that process templates are added artifacts in the design phase of the BPM life cycle. The addition of variability management to BPM introduces the role of template designer. The following subsections describe the process pipeline and a system overview that supports the process pipeline.

5.1.1 The Process Pipeline

Figure 5.1 represents the process pipeline, which depicts the order of transformations of process artifacts from design to execution. This figure restricts to design-time variability. The symbols in this figure are borrowed from the Unified Modeling Language (UML). The process artifacts are represented by UML note symbols. Process artifacts can be digital or paper documents. The UML actors either represent human roles or software systems that have a particular role in the transformation or use of a process artifact. In the

context of the SAS-LeG project, the diagram maps a process pipeline to e-Government. The diagram conceptually illustrates how legislation can be digitally propagated from government to municipality, and is divided in three areas: law construction, law modeling and law execution. Those areas correspond to three phases of the BPM life cycle (see Section 3.1.4), which are: analyze, process modeling and process execution.

The flow of the diagram starts with the law documents. Law documents constitute a set of rules that are created by law implementers of the government. The first step is analyzing the law documents, which forms the input of the templates. Templates are created by a template designer who has knowledge of BPMN and constructing process templates using the PVDI framework.

When a process template is given to a municipality, customizations can be made for the specific needs of a municipality. The restrictions of a template, enforce that some parts of the process cannot be changed, as the final process should obey the law. Other parts are allowed to be changed to customize the process model to the municipality needs, resulting in "late modeling" and customization. The only work for a municipality is customizing the process model. Without templates, municipalities have to create their business processes from scratch, which is a point of failure because the implicit restrictions of a process prescribed by the law, might not be implemented to the government's intention. The constraints of a template, enforce that customized process models do not violate the law.

After approval, derived process models are ready for deployment. However, most process execution engines cannot directly execute graphical process models. The graphical information is not needed anymore and the process is transformed to execution rules, which can be used by a process execution engine. The back-end software transforms the BPD to an execution format. However, executable models can be traced back to graphical models, monitoring running processes. The execution format is the last model in the BPM life cycle. Whenever a process is started, an instance of the process model is created for actual execution.

5.1.2 System Overview

The architecture is given as a (loosely) UML deployment diagram in Figure 5.2. The diagram depicts how the process pipeline (see Section 5.1.1) is carried out, with respect to the hardware and software components and the users. The rectangle shapes with two smaller rectangles anchored on them, represent software components. The software components are placed inside nodes (box shapes). The nodes represent hardware systems or platforms

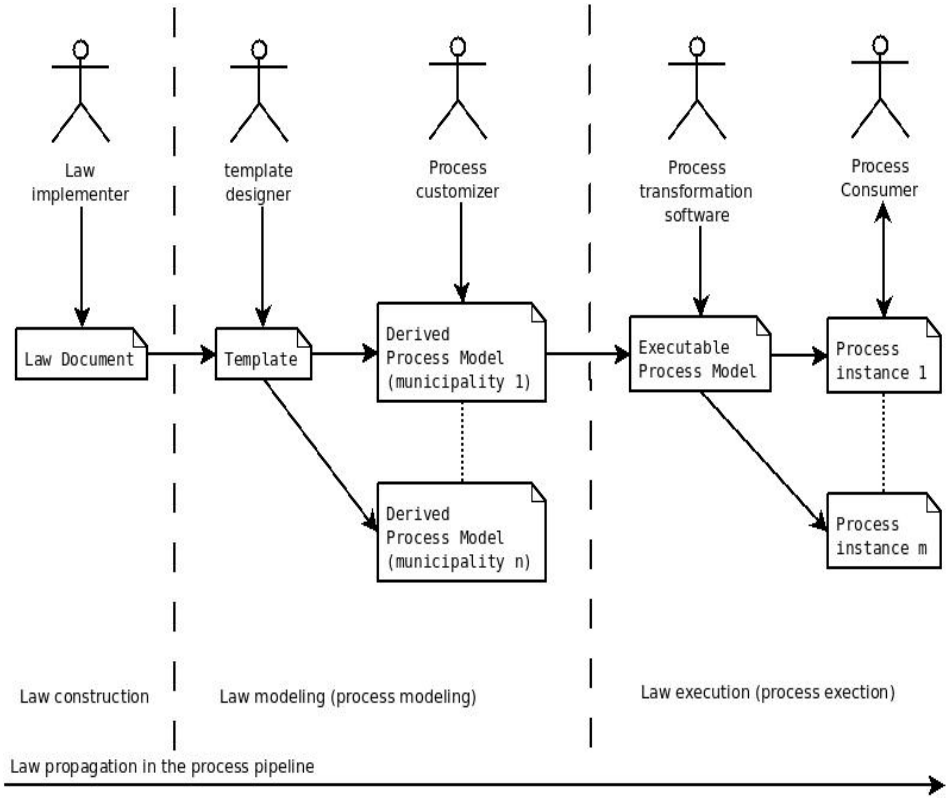

Figure 5.1: The process pipeline in e-Government.

on which software components are installed. A connection between components is represented by a line. Lines ending with an arrowhead, express flow of information in a certain direction. The architecture is divided in two sections, existing of components needed for the government and components needed for municipalities. The dashed line represents this division. The left area represents the government, whereas the area on the right represents a SAS-LeG enabled municipality. A single municipality is labeled with "municipality X", which means there is a one-to-many relation between government and municipalities.

A variability supported process life cycle starts with the design of a process template. Using a modeling tool, a template designer constructs a process template from the corresponding law documents. After approval, templates are stored in a template repository. Since templates need to be offered as reusable services to the municipalities, a catalog system is needed that has access to the template repository. The template catalog has a publish/ subscribe interface that can be accessed by municipalities. For simplicity, both the catalog interface and the template repository are drawn as components hosted on a Web server. In reality, these components might be distributed differently.

Municipalities need process templates that are offered by the government. Publish/ subscribe software is required to let municipalities subscribe to the process templates of their needs. The publish/ subscribe client keeps track on the internal version control of process models. After subscription to a particular process, a municipality can receive a template from the template catalog when authorized. Once a process template has been modified during its life cycle, the template catalog notifies each subscribed municipality of a new version. The publish/ subscribe client software running at the municipality side forwards a notification to the version controller. The version controller keeps track of the internal deployed processes and can check the internal process repository of a municipality.

A process designer uses a process variant modeling tool in which templates can be customized. The process variant modeling tool receives a list of updates from the "process version controller". A process designer makes the necessary customizations and derives a process variant from the process template. The variant modeling tool is equipped with a model checker, which verifies a customized process model against the specification of the respective process template. After successful verification of a variant, the process variant is ready to be sent to the process repository. The process repository of a municipality stores deployed process models that derived from process templates, and is within the back-end of a Business Process Management

System (BPMS).

A BPMS is a platform that includes a wide variety of tools for BPM such as a process modeling tool, a process execution engine, process simulation tools and process monitoring dashboards. A process execution engine creates and maintains instances of a process model. Process instances are the actual running processes that are consumed by either human or machine communication. Typically, processes are human centered where humans perform part of the work required within a particular process. Therefore, a rich BPMS platform does have means for interaction with processes. Notifications such as e-mail are sent to users when a certain state within the process is reached. A process might wait in a state for the user to reply and fill in the data required via (Web) forms. The software for interaction is pictured in the diagram as the workflow front-end component.

A process consumer can be any type of user such as an employee, a citizen or a machine. Cordys has made their BPMS platform, Cordys BOP, available for the SAS-LeG project. Cordys BOP does have development tools integrated into the platform, including a process modeling tool. However, since Cordys BOP and its integrated process modeling tool do not support variability, an external tool is needed that does support variability. The node labeled with business process management platform is kept simple for the sake of understanding the diagram, though in reality the architecture has more detail and different components are distributed over multiple dedicated systems and may be inside a cloud.

5.2 Functional Requirements

Section 5.1 describes a simplified architecture of the SAS-LeG approach. Now that the rationale for the need of a tool is given in the context of e-Government, the tool can be discussed. This section starts with a list of functional requirements for the prototype of the tool. The next two subsections explain the functional requirements along two use case diagrams. The functional requirements of the tool are:

1. Designing a business process in a standardized form;
2. Support for the following constraints:
   (a) Mandatory/ optional activities;
   (b) Flow constraint;
   (c) Parallel constraint;
   (d) Frozen and semi-frozen group.
3. Saving templates;
4. Open templates;
5. A mode for switching between template and process design;

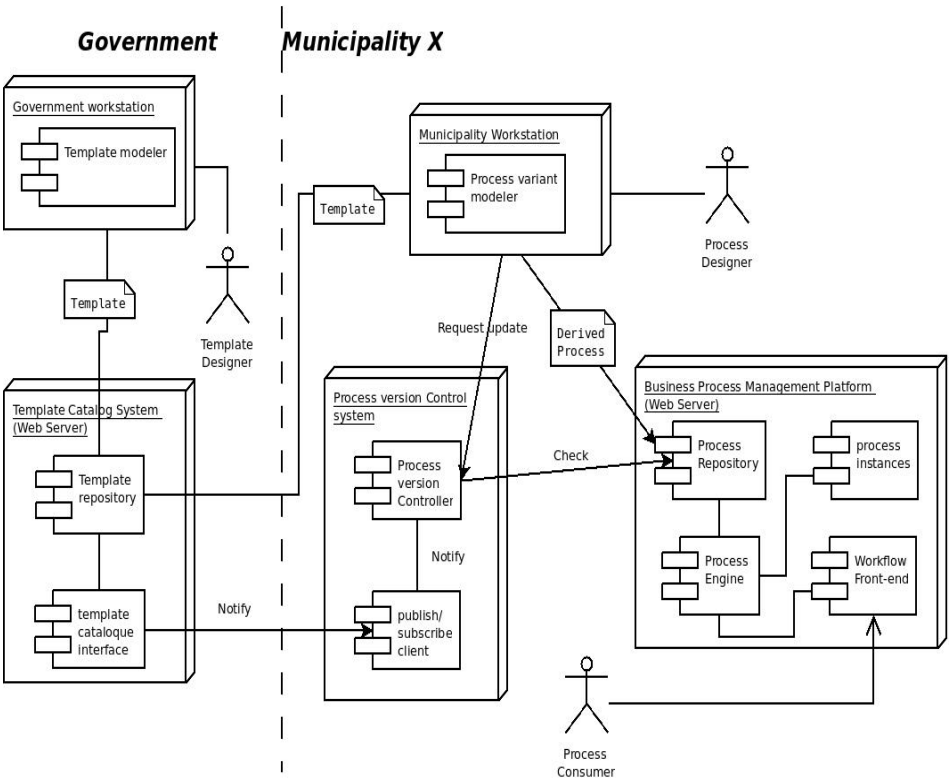

Figure 5.2: Deployment diagram of the SAS-LeG architecture.

6. Verification of process models;
7. Comparing process models.

5.2.1 Use Case Diagram: System

The use case diagram in Figure 5.3 shows the desired behavior of the tool as being an integrated part of the SAS-LeG system. Only the use cases that relate to the tool, are shown in the diagram.

A template designer models a reference process as a BPMN diagram, which relates to the first requirement "designing a business process in a standardized format". The second requirement is "support of constraints", which corresponds to the use case "model template". BPMN diagrams can be constrained by the constraint features of the PVDI framework. Approved templates are submitted to a template repository. A process designer is able to receive a template from the repository and can then customize the process. When a process is customized, a process designer can give approval for deployment. Before deployment, consistency of the process is checked against the template constraints (requirement 6). Whenever a customized process model is inconsistent with a template, the tool should give an error and flag the inconsistencies.

5.2.2 Use Case Diagram: Process Modeling Tool

The use case diagram depicted in Figure 5.4 shows the required behavioral aspects of the tool. This diagram makes no separation between template designer and process designer. Both roles are now represented by the actor "process designer". Separation of roles is left out, because both BPMN elements and constraint elements can be represented by the same abstract concept of "graphical element". The same is true for connectors between elements.

The process designer creates a template on a canvas where the graphical elements can be placed. A graphical element can be either a BPMN element or a template element. After an element has been placed on the canvas, it can be selected. A selected element can be removed or moved position over the canvas. Furthermore the properties can be changed. The elements can be connected by either a BPMN connector such as a flow or a constraint connector such as a parallel constraint. Connectors also have properties that can be set by the user.

Some template constructs can only be applied on a selected element or connector (use case "assign constraint on selection"). Depending on the role of template designer or process designer, the tool should have a mode where both roles can be carried out. Of course a process model can be

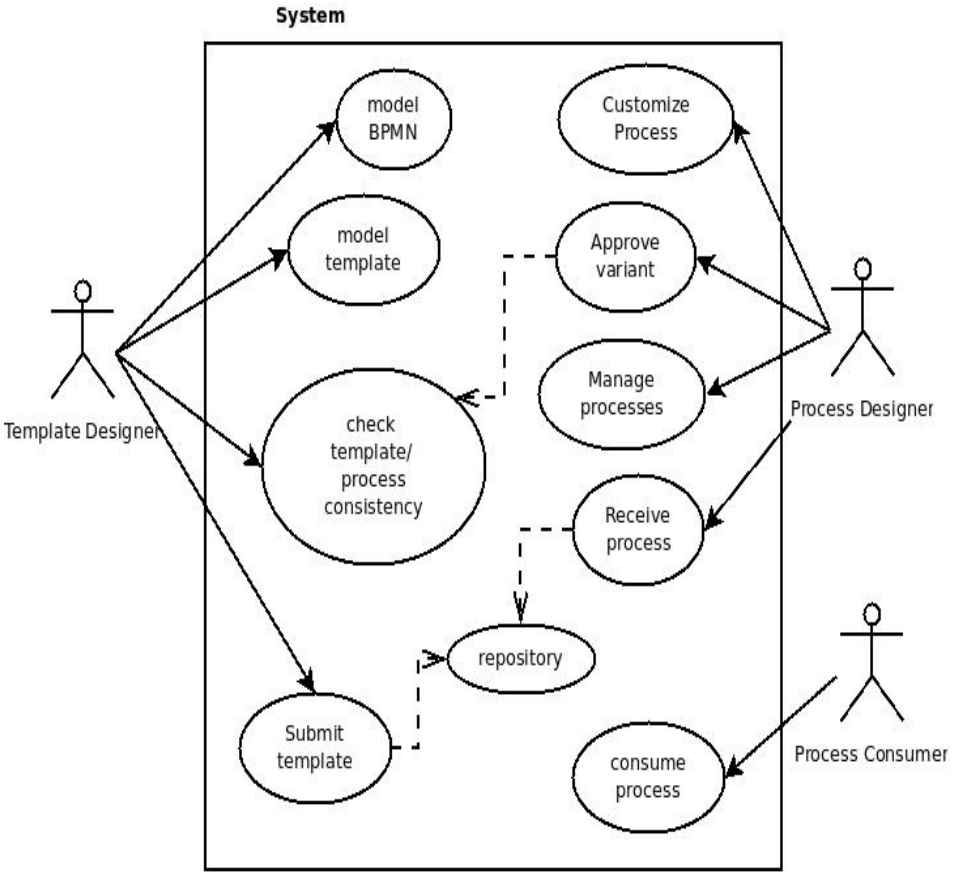

Figure 5.3: Use case diagram: system.

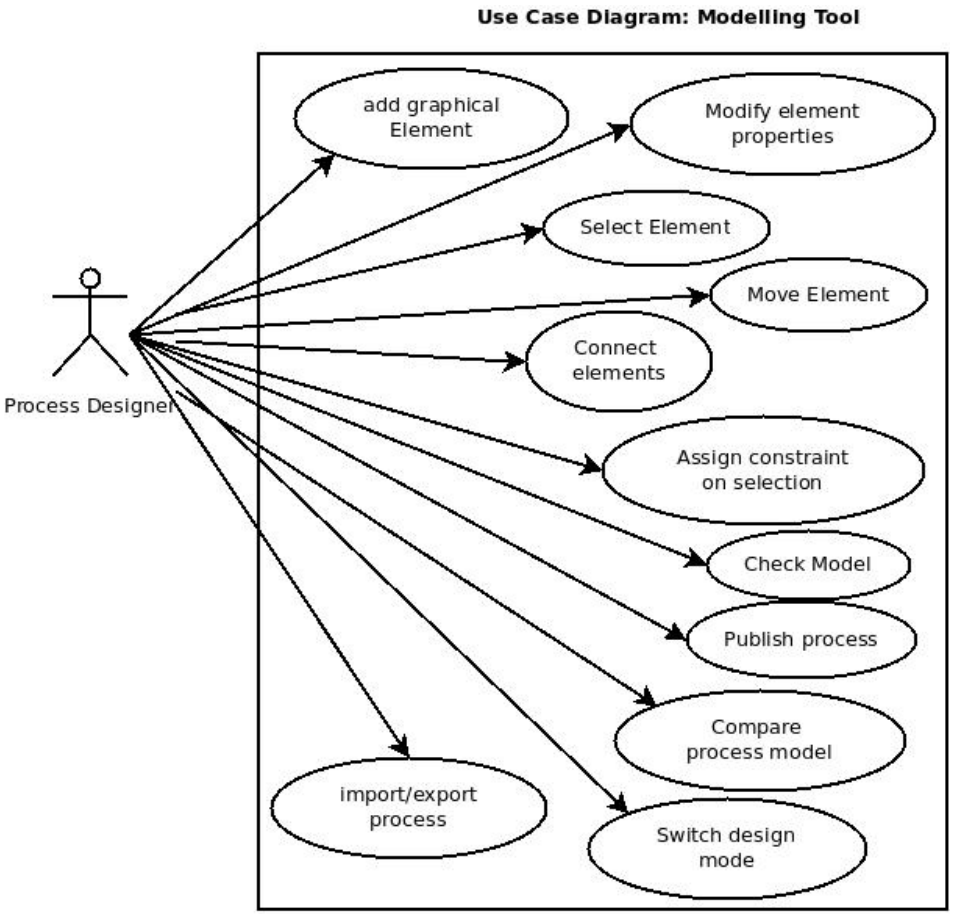

Figure 5.4: Use case diagram: process modeling tool.

imported and exported by the user. Finally, a customized process model can be checked on correctness.

5.3 Development Challenges

Because demonstrating the PVDI framework with a software tool is central to this thesis, development of this tool comes with a few challenges. The next paragraphs describe the challenges that arose during development of the software prototype that is introduced in Chapter 6. These challenges are typical for development of a graphical modeling tool.

Software Design and Development

The first challenges are software design and development. Even though the tool is a prototype, software design is important because a graphical

modeling tool is a complex piece of software. As more features are required, complexity increases significantly. A simple design serves as a road map during development. However, with increasing complexity, maintaining comprehensiveness of the software design is difficult. The implementation pushes complexity further. During programming, many unforeseen implementation specific issues arise. Those details add up to the challenges of a software project. Additionally, software design has challenges on its own and requires application of varying software engineering disciplines such as software patterns, architecture and software design.

2D Graphics and Interaction

Since BPMN models are 2-dimensional diagrams, 2D graphics is essential for a graphical modeling tool. A wide variety of 2D graphics libraries are available for most programming languages. Each graphics library has its pros and cons. Some libraries are easy to use, but do not support advanced graphical operations. Other libraries are low-level and require more geometry and programming skills from the programmer. Basic knowledge of 2D graphics is required in any case. When a particular graphical feature missing in a graphics library, geometry is needed for a self-made implementation.

Marshaling

A graphical editor needs the ability to import and export process models, which is usually done using the Extensible Markup Language (XML). When process models are stored and later retrieved, the visualization should exactly be as before. Preferably, the format should be compatible with other existing business process modeling tools. The graphical editor needs a marshaling component for storing and retrieving business process diagrams in an XML based language that is able to capture graphical and executable aspects. Furthermore, the XML format to be used requires a modification, such that template functionality can be stored.

Model Checking

Customized process models need to be checked whether they are valid derivations of a process template. A template can virtually have an infinite amount of customized process variants. Permutation of different process graphs, depends entirely on the constraints in a process model, and because of the property of endless ways in configuring graphs, the PVDI framework proposes formulas for each constraint. Thus, the tool requires to have a model checker, that implements the PVDI framework. Model checking involves graph theory and temporal logic.

5.4 Implementation

The tool presented in this thesis (see Chapter 6), is a prototype for demonstrating and testing practicability of the PVDI framework. Groefsema, et al. [14] developed a Java package for offering tool support of the PVDI framework. The core of the package is a CTL^+ model checker in which a programmer can create CTL^+ formulas. The tool is developed from scratch and is kept simple to demonstrate the purpose of the SAS-LeG approach to business process variability. For software development a top-down approach with iterative development is followed. The tool uses the PVDI package to generate and verify CTL^+ formulas from a given process template.

Since only a Java Runtime Environment is required, the tool can be easily distributed for testing purposes. A testing user is not burdened with installation of any proprietary software, or software development environments such as the Eclipse platform. In addition, carrying out new ideas is easier, and does not put restrictions on development. A major drawback of this approach is that a lot of effort in development is required, which burdened this thesis with a lot of implementation specific details.

Because most tool vendors support XPDL (see Section 3.1.3) as format for interchanging process models, selecting XPDL was an easy choice. Moreover, Cordys BOP process modeler [6] has XPDL support for importing/ exporting process models. Because Cordys is a technology partner of the SAS-LeG project and the ability to extend XPDL with new constructs for template constraints, using XPDL for storing process templates was a convenient choice. The Cordys BOP platform contains a plethora of BPM tools including a process execution engine. The software prototype can be used in conjunction with Cordys BOP, to simulate a process life cycle.

Chapter 6

Software Prototype

The PVDI framework (see Chapter 4) proposes process templates, in which a combination of declarative and imperative modeling techniques are used for describing variability in process models. Process templates are constructed from a reference process and a set of constraints. The constraints describe variability in business process models. The PVDI framework provides a formalization of how constraints on a process model can be checked. BPMN (see Section 3.1.2) is the process language used for modeling process diagrams. The formalization of constraints is given in CTL$^+$, which describes how model checking can be done for template compatible modeling tools. This chapter introduces the Variability extended BPMN (VxBPMN) designer, which demonstrates the PVDI framework in practice. Each constraint from the PVDI framework specified in [14] is showcased in Section 6.2.

6.1 VxBPMN Designer

VxBPMN designer (see Figure 6.1) is a graphical modeling tool, running as a Java stand-alone application. VxBPMN designer runs on any computer equipped with a Java Runtime Environment and can be easily distributed. The Graphical User Interface (GUI) has the standard Java Swing look and feel.

The canvas located at the center, is where business process diagrams are drawn. The toolbox located on the left, is where the constructs for a template/ process can be selected from. A selected element or connector can be put on the canvas. VxBPMN designer can be in either two modes, "template design" or "process design". These modes reflect the role of the user. In "template design", the toolbox is equipped with the constraints features in addition to the BPMN constructs. The buttons for adding constraints are not available in "process design". In "template design", the user has full

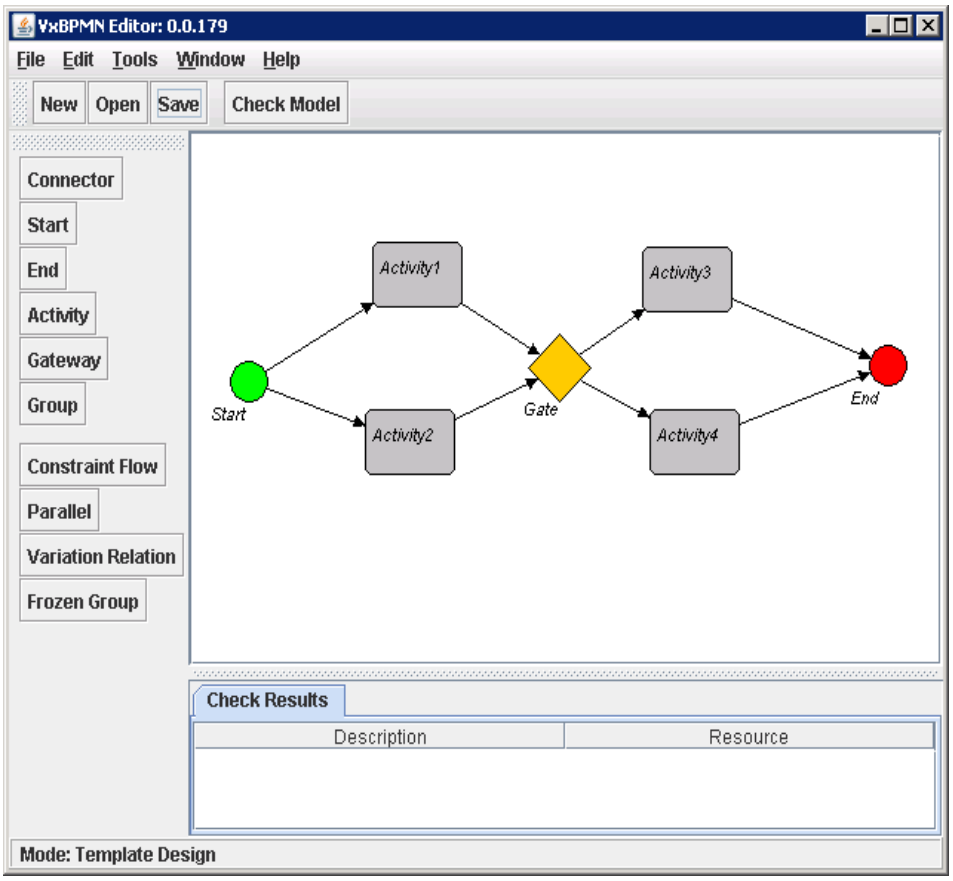

Figure 6.1: VxBPMN designer.

control over editing the BPD, whereas in "process mode" the user can only change what is allowed according to the template. The mode, the user is working in, is showed in the status bar at the bottom of the GUI.

Figure 6.1 also shows an example of a simple process model with four activities, one gateway and a start and end event of the process. Elements are labeled and connected by flows (visualized as arrows). Process models can be checked whether they are valid according to their templates. When a process model contains errors, the graphical elements on the canvas are highlighted. The highlighted errors are explained with textual descriptions, summarized in the "check results" tab located at the bottom region of the GUI. A description and the corresponding resource responsible for the error, are given. Furthermore, VxBPMN designer comes with tool and menu bars located at the top of the GUI, where most functionality is found.

6.2 Demonstrating PVDI's Template Features

This section demonstrates the constraints from the PVDI framework formulated and explained in Chapter 4. These constraints are: flow constraint, parallel constraint, frozen group, optional activity, weak link and floating activity. Each constraint is presented with a basic example and a set of CTL^+ formulas that are generated from the respective template. In each example, a process model is customized. After customization, the model checker of VxBPMN designer uses the CTL^+ formulas to verify a process model, which either accepts or rejects a customized model as Figure 3.4 illustrates. For clarity of the formulas, end events are represented by the symbol $\otimes$ and activity names are lowercased to distinguish them from the letters used by the CTL^+ syntax.

6.2.1 Flow Constraint

With flow constraints, certain restrictions can be put on paths inside a graph. For example, a certain activity should be followed in a path starting from another arbitrary activity. Another example would be a certain activity should never exists in a path. As formulated in Section 4.3, a flow constraint has six forms of constraining paths. Figure 6.2 depicts a process template with a chain of four BPMN activities, labeled a, c, d and b respectively. In addition the model has a flow constraint with activity a as source node and activity b as target node. The following CTL^+ formula is generated from the flow constraint:

$$(a) \Rightarrow EF(b)$$

The formula shows that, the flow constraint is of type "exists finally" (EF), which means that in the graph, activity a should eventually be followed by activity b in a path. As explained in Section 4.3 a flow constraint has two dimensions, path and distance. The path dimension is set to "exists", which means that activity b should be followed in any path that originates from activity a. With more paths branching from activity a, at least one path should lead to activity b. In this example the process model has only one path where activity a is followed by activity b. Alternatively, the path dimension could have been set to "All". In such a case, every path originating from a should lead to activity b. The distance dimension could be set to either finally or next. If the distance dimension is set to next, a normal flow going from a to b directly, is required. However, the distance dimension is set to finally, which requires a to be followed by b eventually.

When the process template of Figure 6.2 is passed to a process designer, the process model can be freely changed as long as the flow constraint is satisfied. For example the process designer decides to remove the flow between

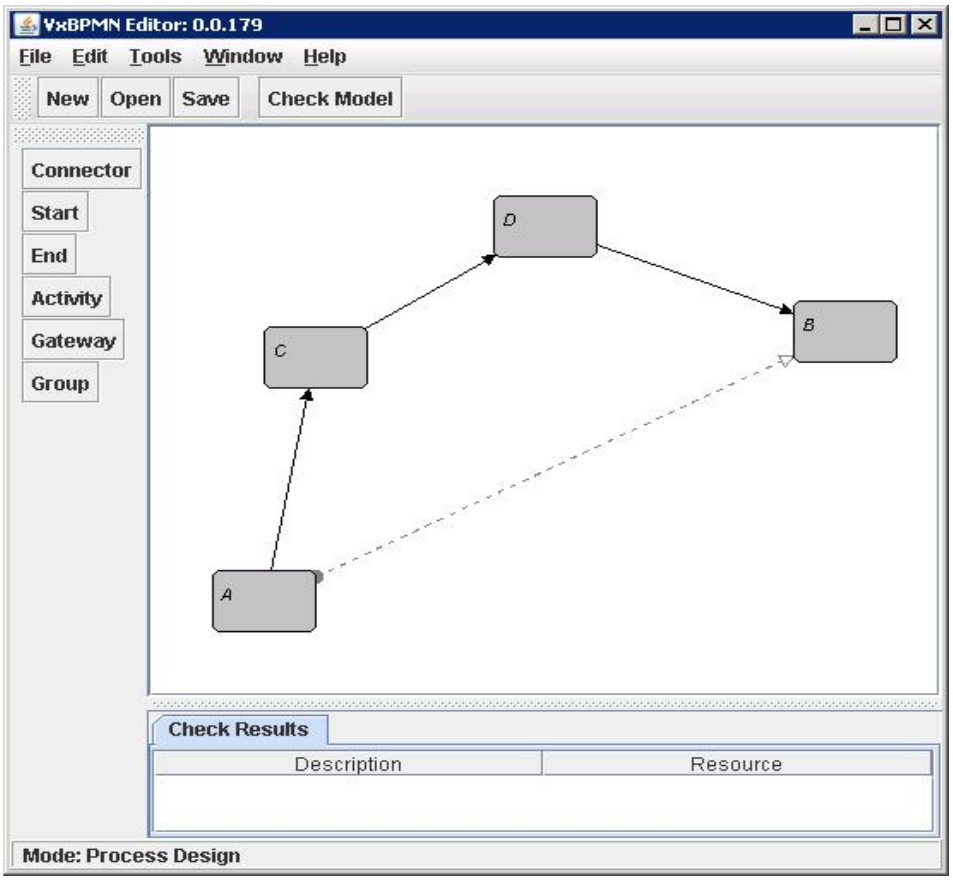

Figure 6.2: Flow constraint between activities a and b.

activity d and b. The modified process model is shown in Figure 6.3. Since the process model has no path leading to activity b, the model checker gives an error. The process designer should satisfy the flow constraint, else the model checker gives an errors. The flow constraint guarantees that activity b will always be followed somewhere in any path originating from activity a. In a simple graph, the advantage might not look obvious. However, for large process models with multiple branching paths having chains consisting of large number of activities, the flow constraint would be very useful. VxBPMN designer helps the user in not forgetting to satisfy this requirement. Theoretically, flow constraints can handle connected structures with an infinite number of elements. Eventually, regardless the length of a chain, a should be followed by b.

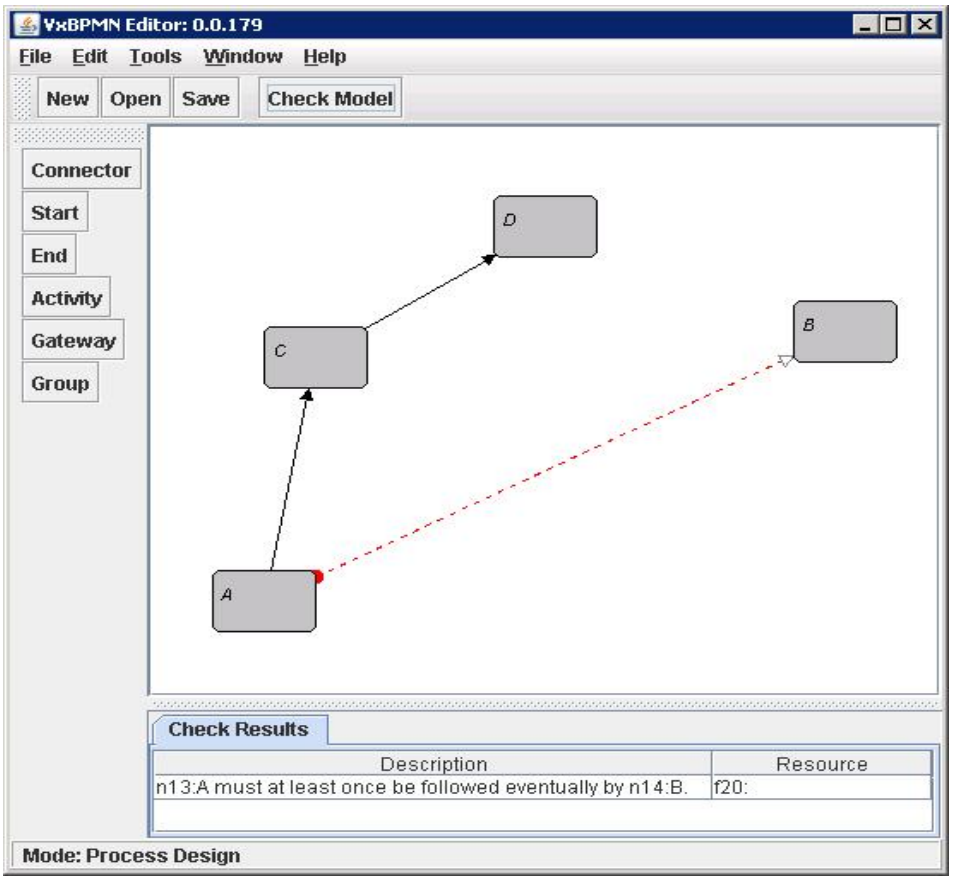

Figure 6.3: Flow constraint not satisfied.

6.2.2 Parallel Constraint

As explained in Section 4.4, a parallel constraint between two activities, enforces that any flow coming out of any of the two shall never finally lead into the other activity. Either activity always exclude their outgoing flow sequence leading into the other one. Figure 6.4 and 6.5 demonstrate the parallel constraint. The shape for a parallel constraint is a dashed line with a diamond shape on both ends of the line as illustrated in Figure 4.2. In Figure 6.4, a parallel constraint is drawn between activity a and activity b. The following CTL$^+$ formulas are generated for the parallel constraint:

$$(a) \Rightarrow AG\neg(b)$$
$$(b) \Rightarrow AG\neg(a)$$

As the figure illustrates, no sequence of flows starts from activity a and ends in activity b, nor has the graph a sequence starting from b ending in a. Both activities mutually exclude each other from passing information to, or receiving information from each other in any path. During verification of the process model, VxBPMN designer does not report an error, because both activities are parallel from each other.

In Figure 6.5, an error is flagged when the model is checked. The corresponding parallel constraint is now colored red, which means that the parallel constraint is not satisfied. Additionally, an error message is given in the "check results" table. The error message: "a may never be in the same path as b" indicates the presence of a flow sequence starting from activity a that ends in activity b eventually. The constraint is not satisfied because of a flow sequence leading from activity c to activity b labeled "flow 1" that violates the parallel constraint. Because activity c can be reached from activity a, the process model has a sequence flow from activity a to activity b, making the model invalid. The model checker makes sure that an activity is never in the same path as the other activity, regardless of the process' complexity.

A parallel constraint is useful when one wants to enforce that both activities are always parallel to each other. In this example, the violation can easily be seen, but when process models increase in complexity, manually checking errors becomes difficult. A model checker is responsible for verifying that a constraint is never violated, regardless a graph's complexity. The CTL$^+$ formulas behind the scenes, make sure that a constraint should always be satisfied. Furthermore, like the flow constraint, a parallel constraint works on a set of activities.

6.2.3 Frozen Group

Section 4.5.1 presented the frozen group along with the CTL$^+$ formulas needed for model checking. This section demonstrates the frozen group

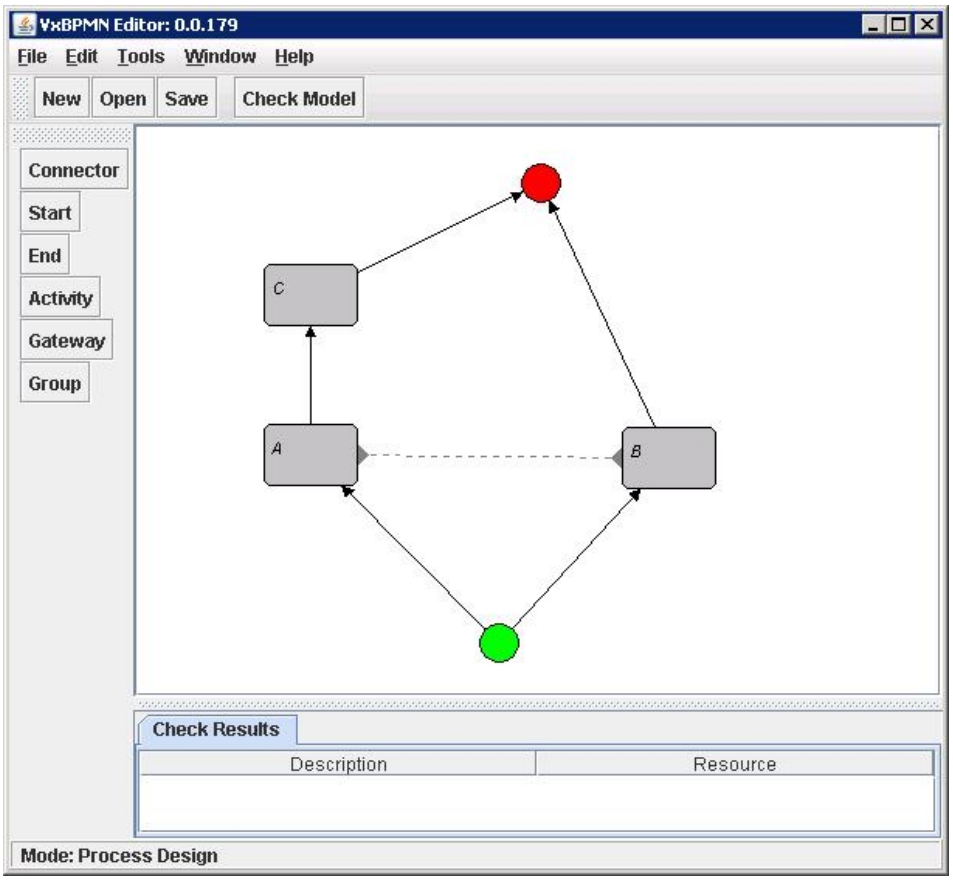

Figure 6.4: Parallel constraint between activities a and b.

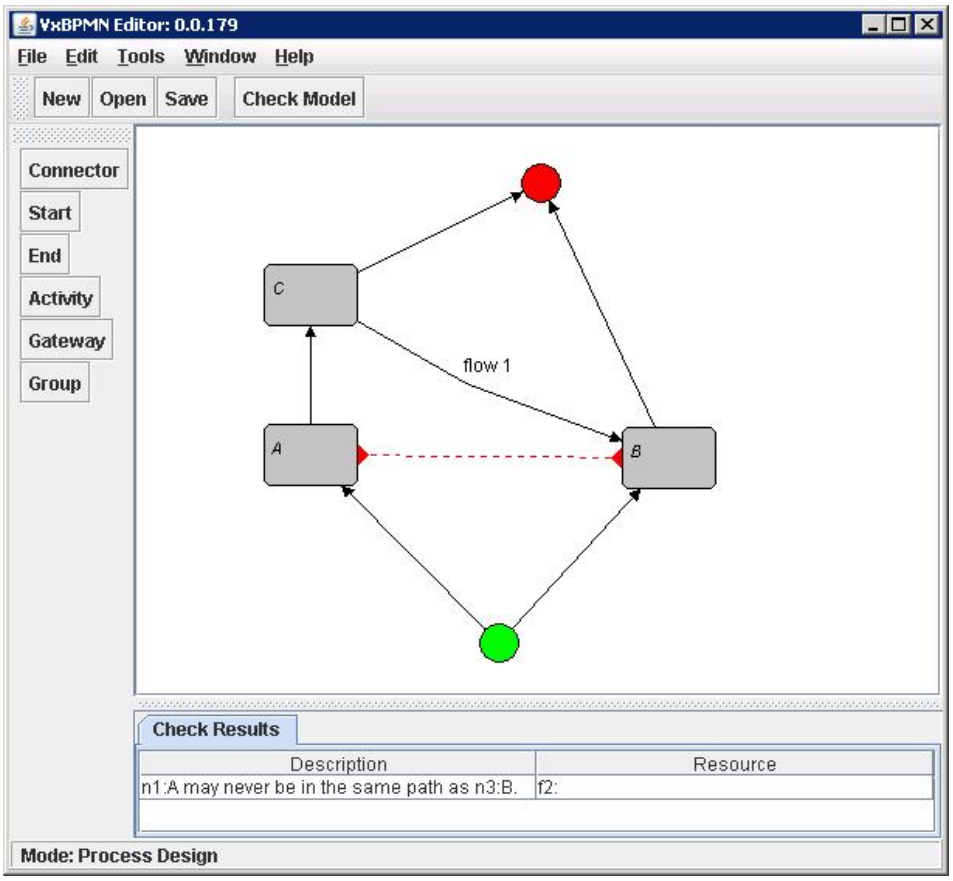

Figure 6.5: Parallel constraint violated.

constraint as a template feature of VxBPMN designer. Figure 6.6 depicts a simple process model that is entirely enclosed by a frozen group. The frozen group is visualized as a dashed rounded rectangle. From the perspective of the user interface, the frozen group works like a compartment, which is a region in which other elements can be placed. All elements inside the region of a frozen group, belong to it. The user can either put a frozen group around a region of elements, or elements can be put inside an already existing frozen group. The formula generator of the tool creates the following formulas from the template.

$$
\begin{aligned}
&a \Rightarrow A[((b \vee c \vee d)U\otimes) \vee ((b \vee e \vee f)U\otimes)] \\
&b \Rightarrow A[((c \vee d)U\otimes) \vee ((e \vee f)U\otimes)] \\
&c \Rightarrow A(dU\otimes) \\
&d \Rightarrow AX \otimes \\
&e \Rightarrow A(fU\otimes) \\
&f \Rightarrow AX\otimes
\end{aligned}
$$

As explained in Section 4.5.1, a process designer is not allowed to make any modifications to a graph that is inside a frozen group. When a process designer tries to modify a frozen group, the model checker raises an error during verification. In Figure 6.7, activity g is added to the frozen group. The model checker flags an error by highlighting the structure where the error occurs, and provides an error message in the "check results" table. Activities a and b and the start event, are also highlighted because activity g is reached via those three elements. Preferable, activity g should have been highlighted as the element that causes the error (not implemented in this prototype).

A frozen group is threated as a sub process with an implicit start and end event. For simplicity, groups cannot be nested in the version of the tool presented in this thesis, and only supports groups with one start and one end event. The implicit start and end events are necessary because the model checker needs to create formulas for each path between start and end nodes. The inclusion of start and end events are used to simplify path searching. With an implicit inclusion of a start and end event, the formula generator only needs to find all paths in between, whereas if they are not present, the search for start and end events would have been required.

6.2.4 Optional Activities

Since generating CTL$^+$ formulas and model checking are computational intensive, generating CTL$^+$ formulas for a frozen group might seem unnecessary, because the user could as well be restricted from editing a frozen group. However, a frozen group can be made semi-frozen. A group constraint leaves

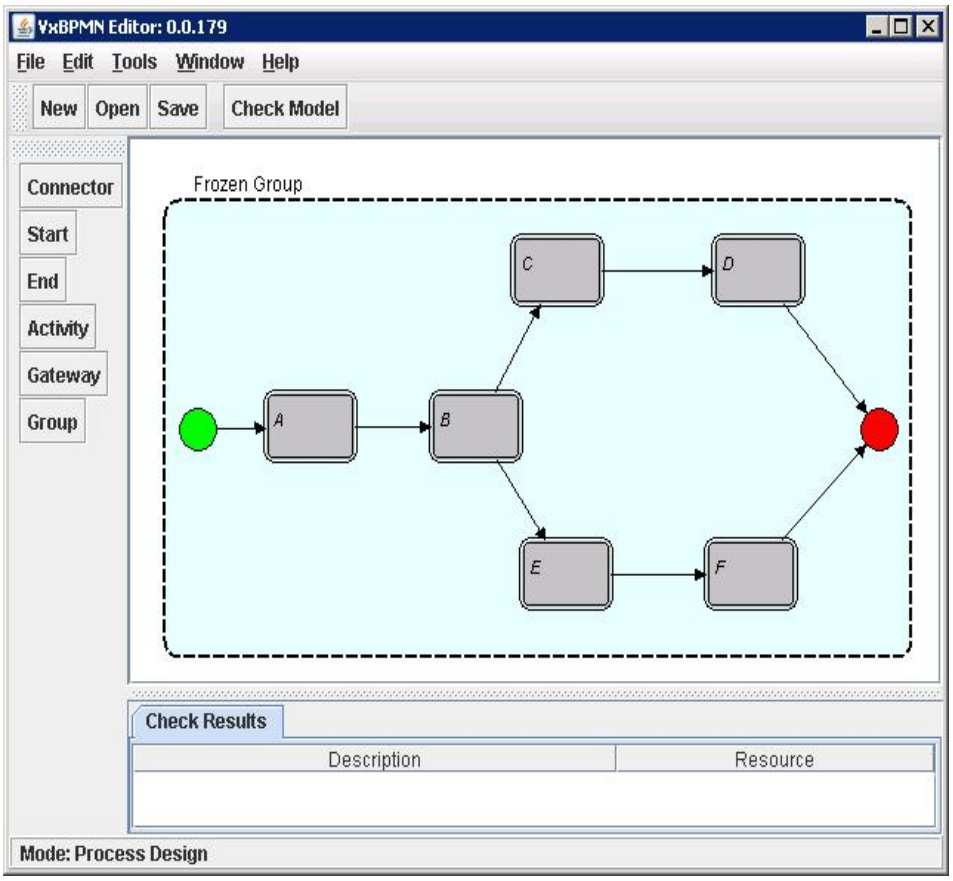

Figure 6.6: Frozen group.

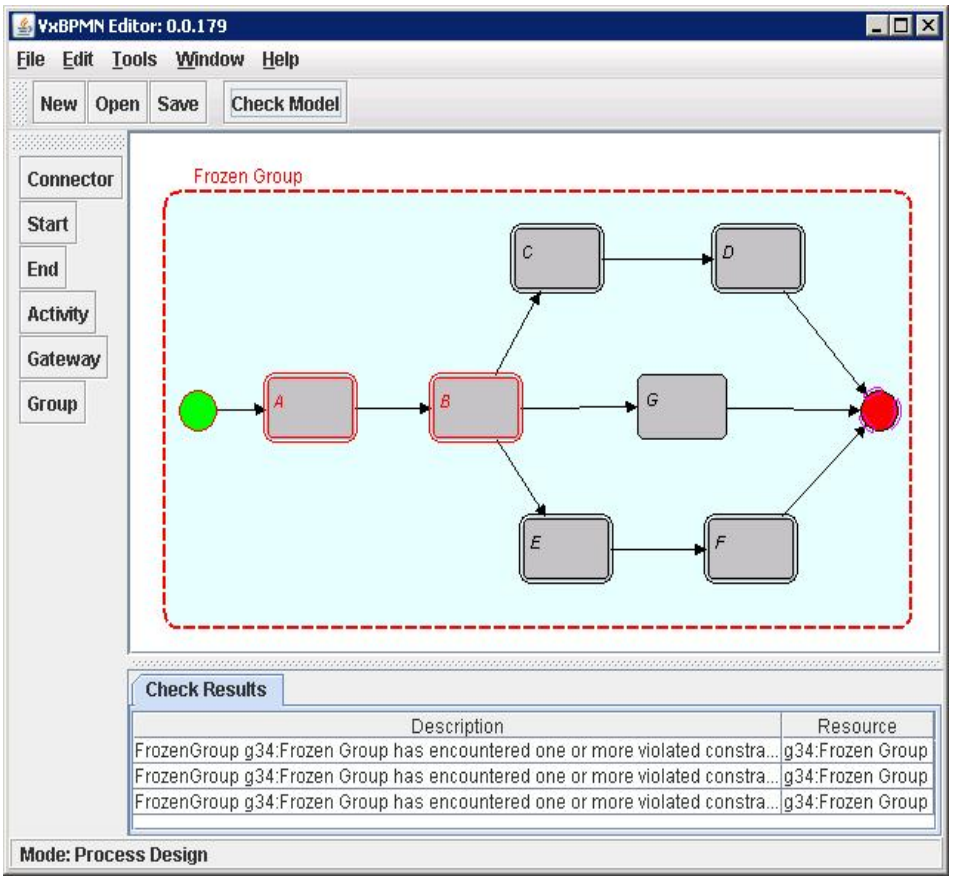

Figure 6.7: Frozen group violated by activity g.

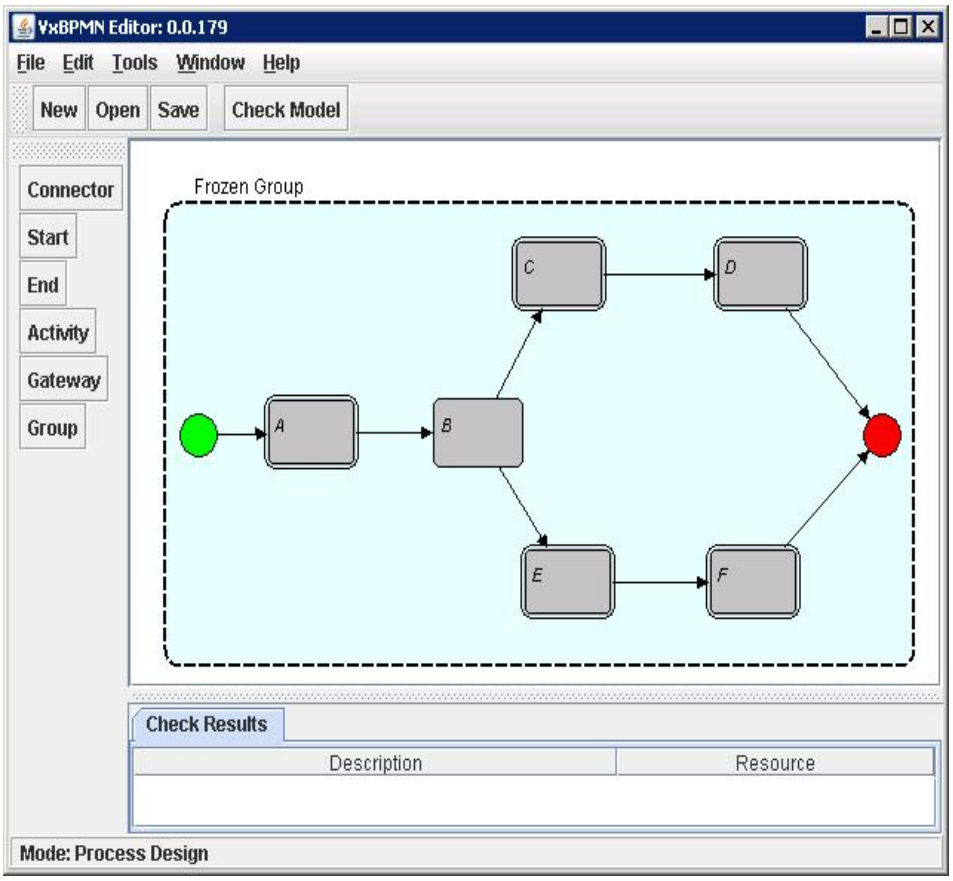

Figure 6.8: Activity b is optional.

room for modifications, which needs to be explicitly specified, as the frozen group does not allow any modifications by default. The first semi-frozen feature is the "optional activity". As explained in Section 4.5.2, optional activities are allowed to be removed from the frozen group. In contrast to optional activities, "mandatory activities" are not allowed to be removed. Figure 6.8 depicts a semi-frozen group with activity b made optional. Optional activities are visualized with a single border; mandatory activities have a double border. When activity b is removed from the process model, the graph looks like Figure 6.9. Because activity b is optional, the model checker does not raise an error. Hence, the resulting process model in Figure 6.9 is a valid derivation from Figure 6.8.

6.2.5 Weak Link

Another feature of the semi-frozen group is the weak link. A weak link allows for arbitrary insertion of multiple paths of nodes between the source

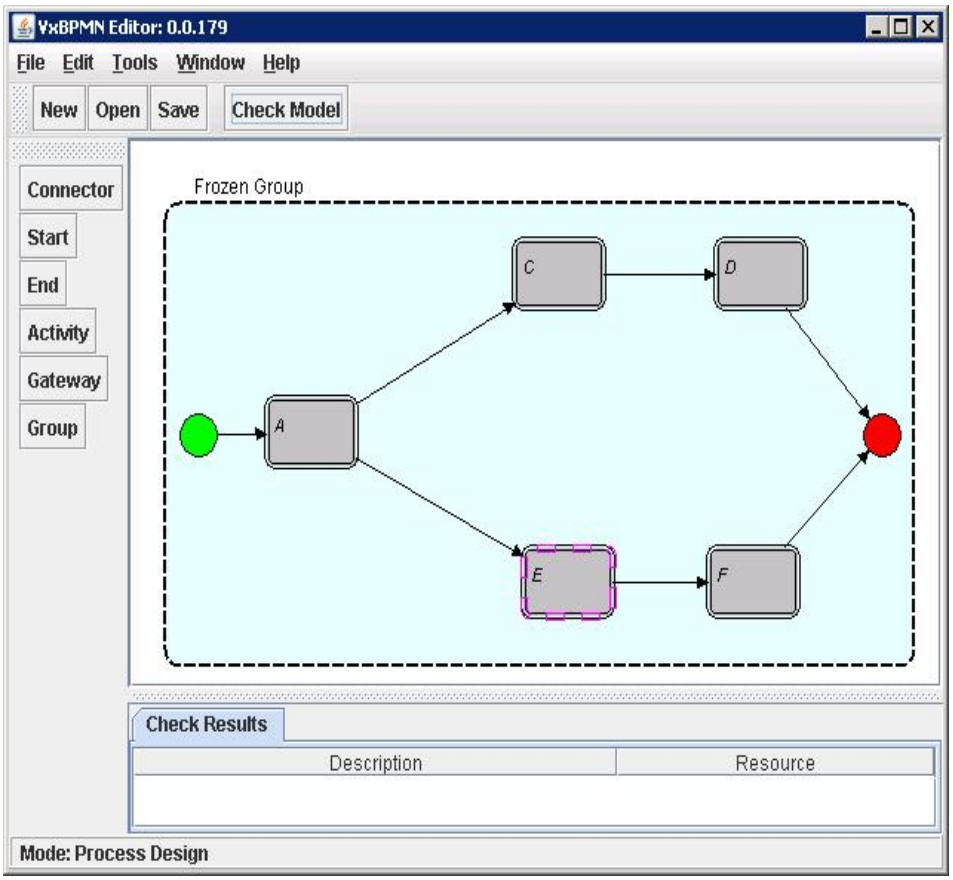

Figure 6.9: Optional activity b removed.

and target nodes of a weak link. A weak link is similar to a flow constraint with a "exist finally" property. The graph of Figure 6.6, from the frozen group example given in Section 6.2.3, is modified to illustrate the weak link feature. Any BPMN flow between two activities can be weakened. A link can be weakened by right clicking a BPMN flow inside a frozen group and select "weaken" from the context menu.

When a flow is weakened, the frozen group becomes semi-frozen. A template with a weak link is shown in Figure 6.10. A weak link is represented as a flow with a dashed line. In this example a template designer has weakened the link between activity a and b, labeled "Weaken 1", and has a second weak link between activity e and f, labeled "Weaken 2". The inverse operation strengthens a weak link back to a normal BPMN flow connector. The following CTL$^+$ formulas are generated.

$$a \Rightarrow AFA[((b \vee c \vee d)U\otimes) \vee ((b \vee e)UAF(fU\otimes))]$$
$$b \Rightarrow A[((c \vee d)U\otimes) \vee ((e)UAF(fU\otimes))]$$
$$c \Rightarrow A(dU\otimes)$$
$$d \Rightarrow AX \otimes$$
$$e \Rightarrow AFA(fU\otimes)$$
$$f \Rightarrow AX\otimes$$

When the template of Figure 6.10 is passed to a municipality, a process designer is free to add, an element between activity a and activity b as the flow between those two activities, is a weak link. In Figure 6.11, a process designer has added activity ab between those two activities. The addition of activities would be correctly verified by the model checker as the weak link allows the insertion of activity ab between its source and target nodes.

A weak link also allows a whole chain of connected elements to be inserted between its source and target nodes. Activities g, h, i and j are a chain of connected activities and placed between the source and target nodes of "weaken 2". In this example four elements are placed in between with two paths; However, theoretically an unlimited number of activities can be added. The process model is correctly verified, and the model checker does not give an error. If an activity is placed between the source and target nodes of a normal flow, say an activity between activity c and d, then the model checker gives an error; Hence, the flow between c and d is not weakened.

6.2.6 Floating Activities

A third semi-frozen feature is the floating activity. A floating activity can be moved inside a frozen group, making it possible to change position inside a graph. The premise is, that floating activities can only be moved to positions

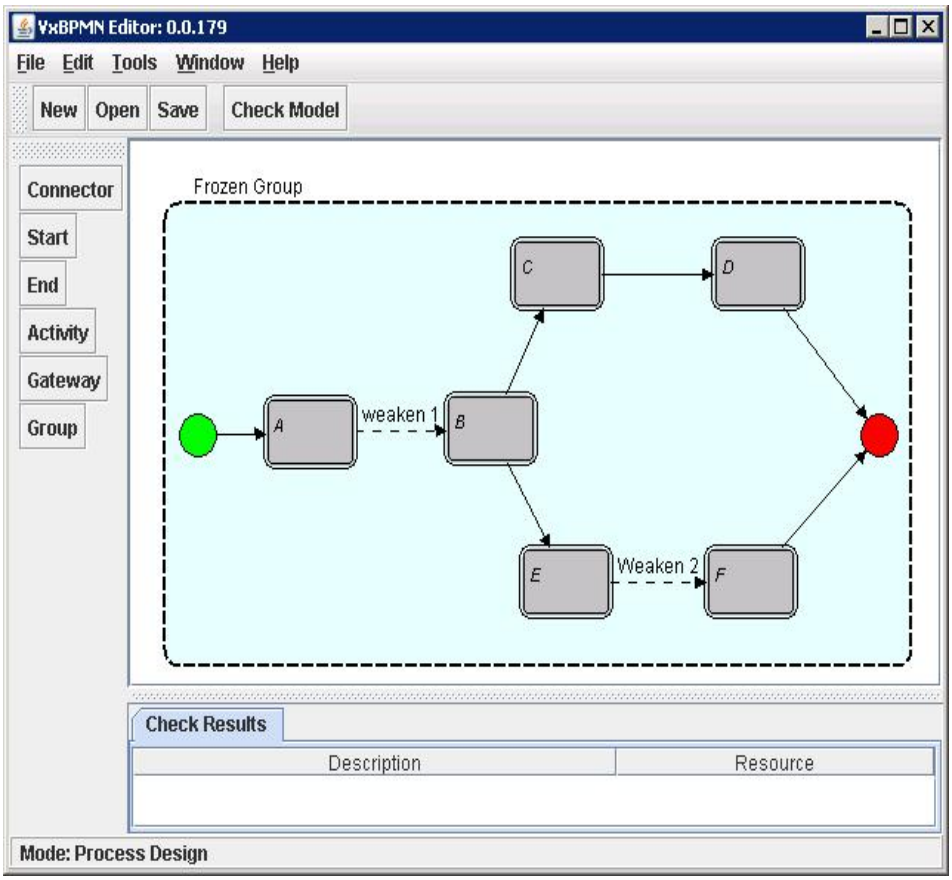

Figure 6.10: Semi-frozen group with two weak links.

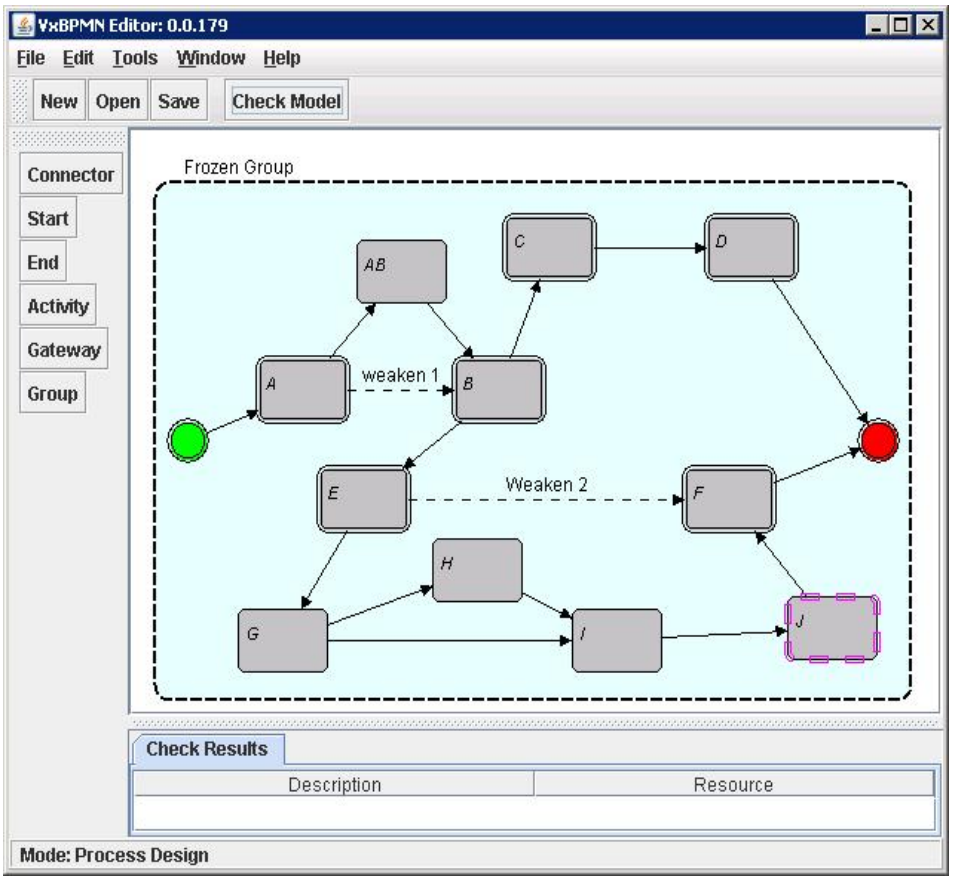

Figure 6.11: Weak link allows insertion of activities.

of the graph with a weak link. Additionally, two floating activities can be swapped. The following CTL$^+$ formulas are generated.

$$a \Rightarrow AFA[((b \vee c \vee d)U\otimes) \vee ((b \vee e)UAF(fU\otimes))]$$
$$c \Rightarrow A(dU\otimes)$$
$$d \Rightarrow AX \otimes$$
$$e \Rightarrow AFA(fU\otimes)$$
$$f \Rightarrow AX\otimes$$

The template in Figure 6.12 has a frozen group, a weak link and activity b is made floating. A floating activity is visualized as an activity with a dashed border. When a template is given to a municipality, the process designer can freely move activity b around the frozen group. The other activities are mandatory, which are fixed inside the frozen group as explained in Section 6.2.3. Furthermore, the frozen group has a weak link, which means that activity b is allowed to change position inside the graph and can be placed between the source and target nodes of the weak link (see Figure 6.13). Activity b is now disconnected from its previous position and connected between activity e and activity f. The model checker does not complain, because the customized process model is a valid derivation of the process template.

6.3 Tool Compatibility

As explained in Section 5.4, VxBPMN designer supports the XPDL format (see Section 3.1.3) for importing and exporting business process diagrams. Compatibility of VxBPMN designer with other tools such as Cordys BOP, is demonstrated by Figure 6.14 and Figure 6.15. Figure 6.14 is a screen shot from the Cordys BOP modeler. The process diagram from Figure 6.10 is imported with the graph having a weak link and being inside a frozen group. The overall structure of the BPMN graphs looks the same in both tools. Figure 6.14 also shows that, both the frozen group and weak link template constructs are not present in the Cordys BOP modeler, and as a result, are not visualized. Because of supporting the XPDL format, VxBPMN designer can be used as a front-end tool for template design while, the execution engine of Cordys BOP can be used for deployment and execution of processes. When a process model is imported in Cordys BOP, the template constructs are ignored by the XPDL parser of Cordys BOP. The result is a standard process model from the template.

XPDL compatibility was also tested with another process modeler called Bizagi process modeler from Bizagi [3] (see Figure 6.15). Similar to Cordys BOP, XPDL formatted process models are smoothly imported and template

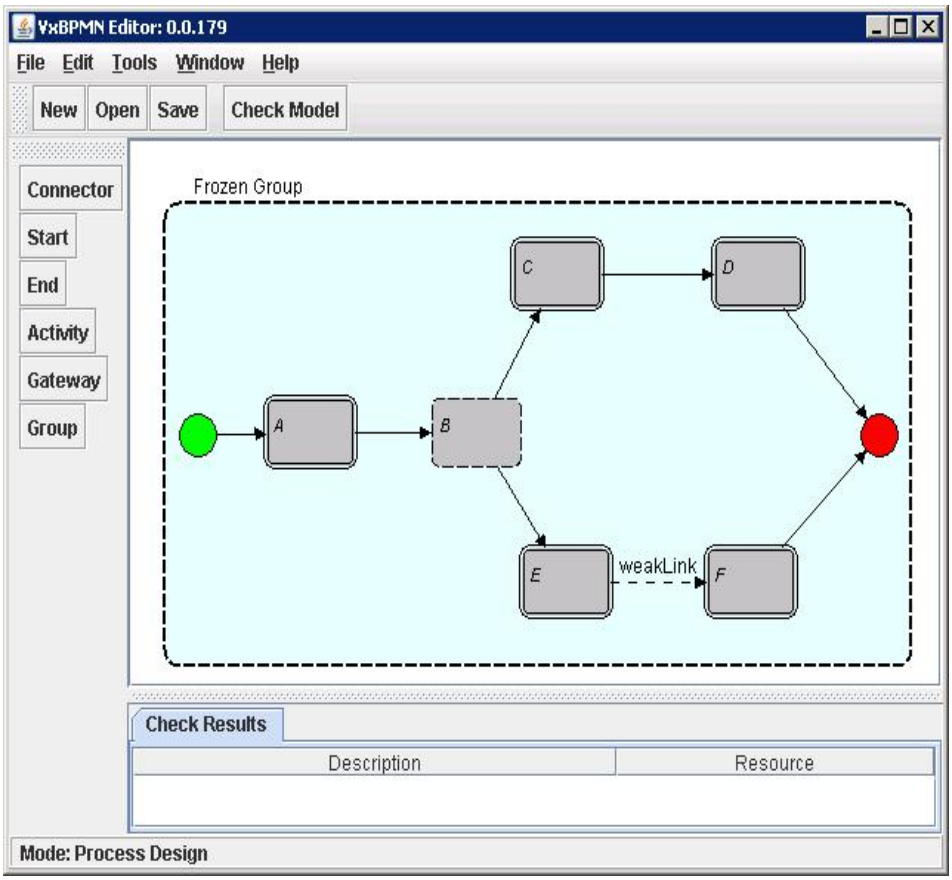

Figure 6.12: Activity b is floating.

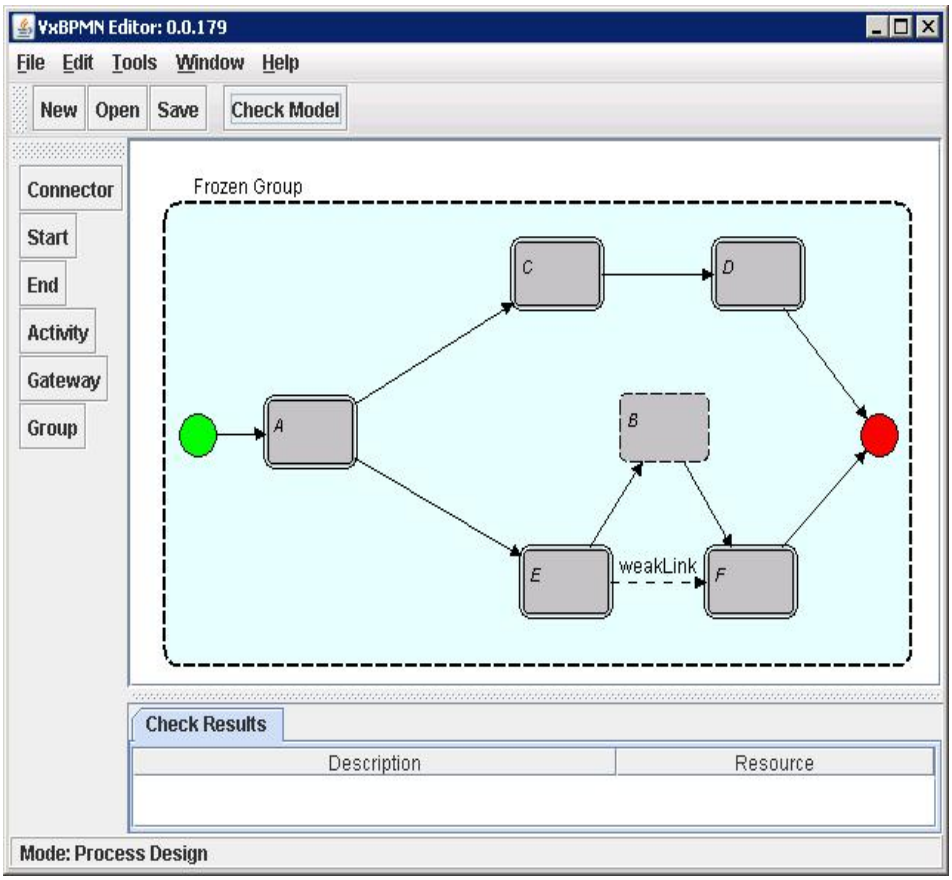

Figure 6.13: Floating activity b can be moved.

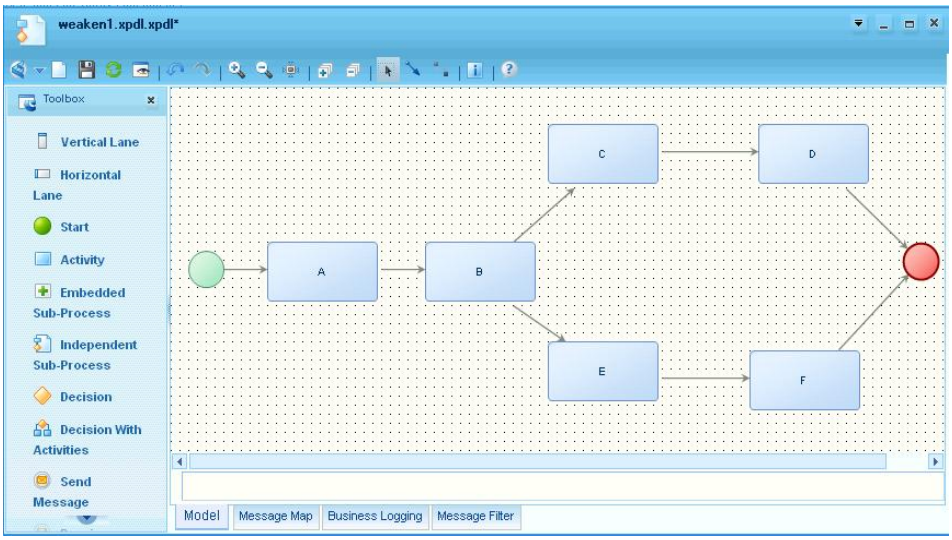

Figure 6.14: Imported process model in Cordys BOP.

constructs are ignored by the Bizagi process modeler. Furthermore, all three modeling tools: VxBPMN designer, Cordys BOP modeler and the Bizagi modeler visualize a process model slightly different (see Figure 6.10, Figure 6.14 and Figure 6.15). The differences are minor in appearance and the BPMN structure remains the same in all three modeling tools. For example, the flows in the Bizagi modeler have rounded edges whereas both VxBPMN designer and Cordys BOP modeler use sharp edges. Different tools have different ways of how the graph should exactly be visualized. These type of graphical preferences can be marshaled in XPDL, which are not forced and any XPDL supporting tool, might choose its own form for the exact layout. These layout differences such as edge visualization and color preferences are tool specific. Nevertheless, the structure of a graph remains the same. Another feature of XPDL, is the option for storing tool and vendor specific properties, allowing different tool developers to have their tool specific preferences stored in the XPDL file. Lastly, process models designed with other tools such as Cordys BOP modeler or Bizagi process modeler, can be imported by VxBPMN designer as well.

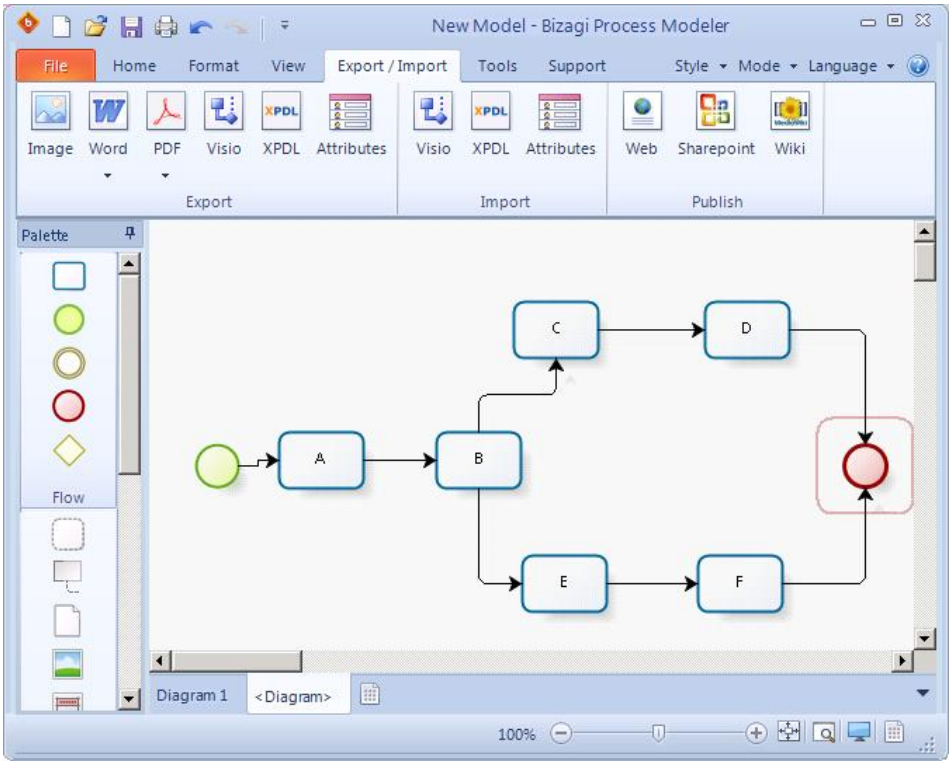

Figure 6.15: Imported process model in Bizagi process modeler.

Chapter 7

Conclusions

From the main research question stated in Section 1.3, three sub questions were derived. The sub questions are:

1. How would process variability using the PVDI framework change the standard life cycle of business process management?

The SAS-LeG project proposes Software as a Service as a model for reusable IT components in e-Government systems. Chapter 2 issues the need for a specific type of business process variability. This specific type of variability separates two organizational roles of authority concerning design of process models. A scenario describing the need for process variability is given in Section 2.2 and Section 3.3 explains business process variability.

The PVDI framework described in Chapter 4 proposes a declarative approach for modeling process variability combined with an imperative process design language such as BPMN (see Section 3.1.2). Declarative approaches constrain process models while providing flexibility for customization features. Business process management is explained in Section 3.1.4 along with its life cycle. However, current BPM practices do not incorporate process variability, and as a result business process platforms such as Cordys BOP are not ready yet for design and management of process variability. Variability management and the PVDI method specifically introduces another phase in the life cycle of BPM. For certain type of organizations such as e-Governments, this phase is required to implement a process as much as possible once, yet providing a degree of flexibility for customization. In Section 5.1.1 the process pipeline conceptually illustrates how the introduction of the method proposed by the PVDI framework would change the BPM life cycle. Furthermore, a simplified system architecture is given in Section 5.1.2 with respect to the addition of the PVDI framework in business pro-

cess management.

2. What is needed to achieve compatibility with other process tools, such that the business process life cycle incorporating the PVDI method can be simulated?

VxBPMN designer supports import and export of process models in XPDL format. Support of XPDL, makes VxBPMN designer compatible with various (BPMN supported) process modeling tools. Section 6.3 shows that the structure and overall graphical layout of process models designed in VxBPMN designer, are preserved when imported in other XPDL supported modeling tools such as Cordys BOP designer and Bizagi process modeler. XPDL's extensions ability was also demonstrated, by adding tags for storing template specific data such as flow and group constraints. Tools with no support for PVDI templates, can still import the reference process from a process template. Both, Cordys BOP designer and Bizagi process modeler ignore tags that yield template information when importing XPDL files. Whenever a process template is imported in a non-PVDI supported process modeler, the process model is visualized such that only the core structure of the reference process model remains. The remaining process model then represents the standard/ advised process model of a template.

Furthermore, VxBPMN designer has a function for switching modes that represent the roles of template design and process customization. In template mode, the user is able to constrain a process model, whereas in process mode the user can only use BPMN elements for customization. The CTL^+ formulas are generated directly after a template is imported in process mode. Any time during customization, process models can be checked. The model checker uses the CTL^+ formulas as a specification to verify customized process models, and checks whether any constraint is violated.

3. Does an automated model checker confirms formal verification of the PVDI framework on customized process models?

The PVDI framework extends BPMN with template semantics for designing process families. The PVDI framework comes with algorithms for generating CTL^+ formulas. These formulas are input for model checking. VxBPMN designer is a software prototype that implements the PVDI framework and is equipped with a formula generator and a model checker. Model checking is required for deriving valid process models from a process template after a model is customized. Whenever verification of a process model fails, the tool provides both textual and graphical feedback. The template features are: flow constraint, parallel constraint, frozen group and semi-frozen group. The constraints are demonstrated in Section 6.2 by providing both correct

and incorrect process models to the model checker of VxBPMN designer. Each demonstrated constraint confirms its specification formulated in [14].

The main research question of this thesis is:

Would a software prototype, demonstrate the PVDI framework as a candidate solution for modeling explicit variability in business process models?

The objective of this thesis is to introduce a process modeling tool that implements and demonstrates the PVDI framework practically. The software prototype is successfully implemented, meeting a minimum set of functional requirements listed in Section 5.2. With this software prototype, basic process diagrams are drawn with a subset of the business process modeling language BPMN.

The process pipeline (see Section 5.1.1) conceptually illustrates how the PVDI framework changes the BPM life cycle. A full simulation of the process pipeline can be achieved by having a process execution engine as back-end tool and a process template modeling tool as a front-end. A business process platform such as Cordys BOP is used as execution engine, which uses BPMN as a design language for process modeling, and XPDL as an interchange format for storing graphical process models. VxBPMN designer is used to demonstrate template design and customization. A demonstration of the software prototype in Section 6.2 confirms practical working of the constraints from the PVDI framework (see Section 4).

In conclusion, VxBPMN designer demonstrates practical feasibility of the PVDI framework as a specification language for constraining process models as templates. The constraints are translated to CTL^+ formulas that form the specification for process models for the automated model checker of VxBPMN designer. A model checker verifies whether customized process models are valid variants of a process family according its specification. A model checker aids during customization of process variants that are valid members of a process family defined by its process template. Although VxBPMN designer is in an incipient state, the tool demonstrates that the PVDI framework makes a good case of balancing preservation of structure while providing flexibility for customization. The amount of flexibility and preservation of structure depend on how a specific set of constraints restrict a template contained process model.

7.1 Future Work

Because this work involved development of a prototype tool, future work is divided in research topics and development effort.

7.1.1 Research Opportunities

The following paragraphs present topics that relate to the tool presented in this thesis.

Usability of the PVDI Language

Because business process management is an interdisciplinary subject [20], there are opportunities in multiple different fields of research. Topics like usability, visual perception, and languages are abundantly studied in various sciences. A visual language such as VxBPMN might still fail when it is hard to learn or when graphical elements are confusing. Thus, a combination of those different aspects influence the success and applicability of the proposed method. These aspects are: expressiveness of a language, usability and possibilities of both the modeling language and tools supporting the language. Interaction and automation possibilities. Various studies such as in [12] apply knowledge from other scientific disciplines for developing visual languages used for domain modeling such as business process modeling. Much of the theory comes from psychology.

Visual and Semantic Discrepancy

The set of graphical elements from the PVDI framework should not violate with the BPMN specification. Furthermore elements from the PVDI framework should not cause confusion with elements from BPMN. Elements should be discriminative and the graphical notation for each feature should reflect its semantics. How visual symbols are perceived by humans goes beyond the realm of computing and is about psychology and linguistics. From other sciences such as psychology, best practices for enhancing the graphical notation of the PVDI framework can be found in literature of those respective sciences and interdisciplinary research fields.

BPMN Specific or Process Modeling Language Generic?

A major question is, whether PVDI should be a separate notation and framework, independent of the process language to use or should it be specific to BPMN? If PVDI is restricted to BPMN only, can PVDI become part of the BPMN specification in a future version of BPMN, or should BPMN and PVDI remain separate notations, where PVDI can be used as an extension.

If PVDI needs to support multiple process modeling languages then the question is: how can PVDI be generalized, to support those languages?

Interoperability of the PVDI Framework

From the previous issue, one can raise the question: how should the PVDI framework be formalized as a machine readable format? The constraints are currently put in a modified XPDL schema. Since the additions to the schema, are committed by other XPDL compatible tools, standardization is required for other tools to adopt the PVDI framework. Does XPDL need an update with template semantics of the PVDI framework? Do other machine readable process formats prove better integrability of the PVDI framework or does the PVDI framework needs to have its own XML schema?

Improving Template Expressiveness

Interesting would be, whether the PVDI framework can be combined with other variability techniques to provide more flexibility and constraint features. One such technique would be the use of questionnaires such as in [23]. Where questions are created by the template designer and the answers can be filled in by the process designer. Such late modeling techniques are useful when a template designer wants to force the process designer to fill in an answer.

Domain Applicability

Furthermore, would BPMN remain a professional tool for the business industry and will the users remain business analysts? Alternatively, can BPMN combined with PVDI become a visual programming language for other domains? Does PVDI have use for other type of processes, e.g., as proposed in [9] where a process of giving a party is used as example for processes in smart homes as services in luxury or aiding disabled people.

Automation

Finally, could the design of process models/ templates be further automated? Ultimately, the vision of the SAS-LeG project is that the law described in natural language, instantly adapts to each municipality without human intervention. When the law changes, the processes used at the municipalities should change accordingly. Future research is needed to see whether natural language processing can be applied, in which law written in natural language is automatically transformed in a process template, in which the law is digitally conserved.

7.1.2 Tool Improvements

The following paragraphs present possible improvements of the tool presented in this thesis.

BPMN Validation

Although VxPBMN designer verifies process templates, the correctness of BPMN models is not checked. Of course both should be checked before a process model should be deployed. However, as demonstrated in Section 6.3, since the tool uses XPDL as a machine readable format for interchanging process models, a process model designed in VxBPMN designer can be imported by other BPMN modelers, which in turn can be used to check whether a process model is a valid BPMN graph. In contrary, a template does not need to be valid BPMN, nor does a constraint need to be valid as a template is a mere blueprint for process models to be derived from the reference model contained by a template. Thus, in template mode, verification of the constraints and checking BPMN are not necessary. However, in process mode both are needed before a process should be deployed.

Basic Tool Operations

A basic feature missing, is to undo actions in the process design mode. Modifications done in process design mode like removing optional activities cannot be undone or put back in a later stage. Other constraints suffer the same problem. The include and exclude constraints also require a way to put in certain elements like including or excluding activities. Thus, the tool needs to store all defined process elements templates are opened in process mode. The user should be able to easily put back template defined structures. The tool needs another toolbox in process mode, consisting of the set of template actions. The defined BPMN elements, created in template mode together with the constraints, determine the outcome space of possible process variants.

Process Attachments

The previous issue gives birth to another usable feature for template design. Certain template defined structures might be allowable in a process variant, but might not be inside the reference process itself. For example the template process has a variation point where only one activity can be positioned, though the template designer wants to restrict the possible choices from a set of predefined fixed activities. In process mode, a mechanism should force the process designer to make a choice from the possible options. This idea can be extended to stimulate reuse, where a template designer, can add

"attachments" to a template. Attachments are predefined structures like activities filled in with a Web service for example. Those attachments are not in the reference processes itself, but are within the template of the process. Attachments are atomic building blocks that are allowable customizations for the realization of a derived process model. Without predefined atomic compositions, a graph remains abstract and can be a structure for any type of process. When activities are filled in, they capture part of the semantics.

Usability Testing

Lastly, thorough testing is needed to evaluate practical use of process templates using the PVDI framework. Different types of data need to be acquired, especially those related to usability. Data acquired from usability testing can be used to improve the PVDI framework and tool support, or even open new opportunities. If users are satisfied using the tool, then PVDI has a chance as a standard for process templates. Large amounts of data about user satisfaction is needed to see whether the PVDI framework adds value to BPM practices.

Bibliography

[1] Marco Aiello, Pavel Bulanov, and Heerko Groefsema. Requirements and Tools for Variability Management. In *COMPSAC Workshops*, pages 245–250. IEEE Computer Society, 2010.

[2] Alessandro Artale. Formal Methods. Computation Tree Logic (Lecture slides: Free University of Bolzano).

[3] Bizagi. Bizagi Process Modeler. Web site. http://www.bizagi.com.

[4] Chang-ai Sun and Marco Aiello. Towards Variable Service Compositions Using VxBPEL. In Hong Mei, editor, *ICSR*, volume 5030 of *Lecture Notes in Computer Science*, pages 257–261. Springer, 2008.

[5] Workflow Management Coalition. XML Process Definition Language (XPDL). Web site. http://www.wfmc.org/xpdl.html.

[6] Cordys. Cordys Business Operations Platform Process Modeler. Web site. http://www.cordys.com.

[7] Peter Dadam and Manfred Reichert. The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - R&D*, 23(2):81–97, 2009.

[8] T.H. Davenport. *Process Innovation: Reengineering work through information technology.* Boston, 1993.

[9] Piet den Dulk. Managing variability in BPEL for adaptable domotic processes. Term project. University of Groningen, 2009.

[10] E.A. Emerson and E.M. Clarke. Using Branching-Time Temporal Logic to Synthesize Synchronization Skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[11] E.A. Emerson and J.Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Science*, 30:1–24, 1985.

[12] Kathrin Figl, Michael Derntl, Manuel Caeiro Rodríguez, and Luca Botturi. Cognitive effectiveness of visual instructional design languages. *J. Vis. Lang. Comput*, 21(6):359–373, 2010.

[13] Gautam Ray and Jay B. Barney and Waleed A. Muhanna. Capabilities, business processes, and competitive advantage: choosing the dependent variable in empirical tests of the resource-based view. *Strategic Management Journal*, 25:23–37, 2004.

[14] Heerko Groefsema, Pavel Bulanov, and Marco Aiello. Declarative Enhancement Framework for Business Processes. In Gerti Kappel, Zakaria Maamar, and Hamid R. Motahari-Nezhad, editors, *ICSOC*, volume 7084 of *Lecture Notes in Computer Science*, pages 495–504. Springer, 2011.

[15] Object Management Group. Business Process Model and Notation (BPMN) specification. 2008.

[16] M. Hammer and J. Champy. *Reengineering the Corporation, A Manifesto for Business Revolution.* New York: HarperBusiness, 1993, 1993.

[17] Intalio. Intalio BPMS. Web site. http://www.intalio.com.

[18] Jan Johannsen and Martin Lange. CTL+ Is Complete for Double Exponential Time. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 767–775. Springer, 2003.

[19] Ekkart Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data Knowl. Eng.*, 56(1):23–40, 2006.

[20] Ryan K. L. Ko. A computer scientist's introductory guide to business process management (BPM). *ACM Crossroads*, 15(4), 2009.

[21] Tobias Küster and Axel Heßler. Towards Transformations from BPMN to Heterogeneous Systems. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 200–211. Springer, 2008.

[22] Bram Leemburg. A declarative meta modeling approach to define process migration constraints. Master's thesis, University of Groningen, 2011.

[23] Vanessa Lopez, Enrico Motta, Victoria Uren, and Marta Sabou. State of the Art on Semantic Question Answering. Technical Report kmi-07-03, Knowledge Media Institute, Open University, Milton Keynes, UK, May 2007.

[24] Oasis. Business Process Execution Language (BPEL) . Web site. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[25] University of Groningen. SAS-LeG project. Web site. http://www.sas-leg.net.

[26] Girish Keshav Palshikar. An introduction to model checking. Web site. http://www.embedded.com, 2004.

[27] M. Pesic. A declarative approach for flexible business processes management. pages 169–180, 2006.

[28] Maja Pesic, M. H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer, 2007.

[29] Jan C. Recker. Opportunities and constraints : the current struggle with BPMN. *Business Process Management Journal*, 16(1):181–201, 2010.

[30] S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.

[31] M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language, 2005.

[32] Emanuel Santos, João Pimentel, Jaelson Castro, Juan Sãnchez, and Oscar Pastor. Configuring the Variability of Business Process Models Using Non-Functional Requirements. In Ilia Bider, Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, and Roland Ukor, editors, *BMMDS/EMMSAD*, volume 50 of *Lecture Notes in Business Information Processing*, pages 274–286. Springer, 2010.

[33] Scott Sehlhorst. Tyner Blain Foundation Series.
Web site. http://tynerblain.com/blog/foundation-series-index/.

[34] Howard Smith and Peter Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2003.

[35] TechRepublic. TechRepublic. Web site. http://www.techrepublic.com.

[36] Tibco. Tibco Business Studio. Web site. http://www.tibco.com.

[37] W. M. P. van der Aalst and T. Basten. Inheritance of Workflows - An approach to tackling problems related to change. *Theoretical Computer Science*, page 2002, 2002.

[38] W. M. P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In N. Markus and F. J. Rump, editors, *EPK*, pages 71–79. GI-Arbeitskreis Geschãftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2002.

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

[40] W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable Process Models as a Basis for Reference Modeling. *Business Process Management Workshops*, pages 512–518, 2006.

[41] W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. pages 1–12, 2003.

[42] Mark Vervuurt. Modeling Business Process Variability. Master's thesis, University of Twente, 2007.

[43] World Wide Web Consortium (W3C). Simple Object Access Protocol (SOAP). Web site. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[44] World Wide Web Consortium (W3C). Web Service Description Language (WSDL). Web site. http://www.w3.org/TR/wsdl.

[45] World Wide Web Consortium (W3C). Web Services Architecture. Web site. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[46] Li Xingyu, Hu Hao, and Lu Jian. Variability model and management for web service: A petri net based approach. In *Computer and Information Technology, 2008. ICCIT 2008. 11th International Conference on*, pages 7 –12, dec. 2008.