Final-year thesis

A framework for modeling and maintaining business process models with variability

August 30, 2012

Author:
Wilco Wijbrandi

Supervisor:
Dr. Alexander Lazovik
Second supervisor:
Prof. Hans Wortmann

Abstract

Business Process Modeling is a popular technique for orchestrating Service-Centric information systems. There are cases in which several very similar business process models are in use. Making copies of business process models and adjusting only certain aspects of the models introduces redundancy and makes making general changes to the models difficult and error prone.

Business Process Variability techniques solve these issues. The PVDI framework is a framework which does this by combining declarative and imperative techniques. The framework allows users to create abstract models (*templates*) and concrete derived models (*variants*).

In this thesis we propose a framework which makes the PVDI framework usable in practice. Important aspects of the framework are a graphical editor and a version control system for the business process models. Using this version control system the relation between templates and variants can be tracked and variants can easily be updated to a newer version of the template. We have implemented the framework as a cloud-based web application.

Contents

Preface

This research project would not have been possible without the support of many people. I wish to tank my supervisor, Dr. Alexander Lazovik, for his guidance, comments and advice throughout the process. I also wish to thank my second supervisor, Prof. Hans Wortmann, for his comments and feedback. I also wish to thank Pavel Bulanov and Heerko Groefsema for all their advice and help.

Chapter 1

Introduction

Business process modeling is a technique for representing processes in enterprises. When a process is formally described, it can be used to perform tasks in an information system. A business process model consists of activities, flow-control elements and events. Activities in a business process model can be seen as single calls to a certain interface. The flow of the process depends on the output of these activities and events. This way, a complex tasks can be described at a high level. Multiple systems can be combined to perform a certain task. Business process models are easier to understand for non-technical users and allow information systems to be more flexible.

There are cases in which there are multiple variants of the same business process model in use. Imagine for example a factory building different versions of the same car. The process for all the cars is mostly the same, but the details differ. The current practice of creating multiple variants is *copy and paste* a business process model several times, resulting in multiple very similar but not linked business process models. This makes making general changes to the models cumbersome and error prone.

These issues can be solved by introducing variability in business process models. There are several techniques available to accomplish variability. In most approaches there is a *master process* or *template* which describes the common part between the business process models. Variants describe the elements that make the business process model suitable for a specific situation.

The PVDI-framework is a framework developed at the University of Groningen which combines declarative and imperative techniques to accomplish business process variability. Templates can be created which pose constraints on possible variants. The authors of the framework have developed a prototype for a graphical editor.

The work described in this thesis is based on the PVDI-framework [23].

Although the PVDI-framework gives templates powerful control over possible variants, there are still obstacles for practical usages. Currently there is only a prototype for a graphical editor. Business process models are stored in an XML-based file, which makes keeping track of changes, collaborating and maintaining a relation between templates and variants difficult.

In this thesis we propose a framework for modeling business process variability. The framework consists of a graphical editor. Business process models are stored in a central version control system. This way it is easy to track changes and manage several versions of templates and variants. When a new version of a template is created, a variant based on an old version can be updated as well.

We have implemented the proposed framework in the form of a cloud-based web application. Web applications offer many advantages compared to desktop applications. The application can be accessed from anywhere where a modern web browser and an Internet connection is available. Files are stored on the servers, so there is no need save them or carry them with you.

In Chapter 2 we will discuss existing work on the field of business process variability. In Chapter 3 we will discuss the SAS-LeG project and the case study of the WMO law in the Netherlands. This cases study describes the problem and the potential advantages of a business process variability framework. In Chapter 4 the proposed framework is described. Section 4.2 discusses the graphical editor and Section 4.3 discusses the version control system. Chapter 5 discusses the implementation of the framework. In Chapter 6 we evaluate the work and in Chapter 7 we will come to our conclusions.

Chapter 2

Related work

2.1 Web Services

Information systems nowadays are often not single databases which store and verify information internally. They are composed of many systems which communicate with each other over local networks and the Internet. For example, when booking your holiday on-line, the systems of several airlines are queried for ticket prices and availability, as well as hotels and car rental companies. When purchasing and paying products on-line, the payment processor automatically notifies the store whether the payment got through.

There is a wide range of databases and software platforms. The organisations whose computer systems have to communicate with each other probably use different, incompatible applications. In order for all these systems to cooperate, a standard, platform independent protocol needs to be used to communicate. Web services provide a standardised way for discovering and communicating with services. They use standard communication protocols and open XML standards [17]. By modeling software actions as interchangeable Web services, information systems become more flexible and are easier to adapt to changes. Information systems built with these techniques are known as Service-Oriented Architectures or Service-Centric systems.

Although Web Services is a widely used standard for standardised communication between different platforms, there are alternatives. For example, interfaces based on *REpresentational State Transfer* (REST) have gained popularity over the past few years [50]. Communication using REST is considered less strict and easier to implement.

2.2 Business Process Modeling

There are two ways to design a Service-Centric information system: *orchestration* and *choreography* [36]. With orchestration there is usually one entity trying to achieve a goal by using other Web services, where with choreography there is a collaboration between several Web services. Orchestration and choreography can also be combined. A choreography could for example be realized by creating an orchestration for every peer involved in the process.

Orchestration allows us to design a complete business process up front. When the process is executed, the machine on which it runs can invoke Web services. A process can for example be an order for an air plane ticket. The process can have different branches at some point. For example, for different payment methods, another sub-process can be executed.

Because Web services always use the same XML-based interface, we can model business processes at an abstract level. The *Business Process Execution Language* (BPEL) [32] is an open XML standard for describing business processes. These processes can be executed on an execution engine supporting BPEL, such as the *Apache Orchestration Director Engine* (Apache ODE) [21]. BPEL supports typed variables, scoping, event-handlers, transactions and some programming constructs as conditional execution and loops. While BPEL is a popular execution language for business processes, there are alternatives such as YAWL [25].

While BPEL allows us to create an abstract definition of a business process, it is merely an XML standard without a graphical representation. Because the lack of a graphical representation it is hard to understand for people without a technical background. Some software vendors have tried to overcome this problem by introducing their own graphical representation.

The *Business Process Model and Notation* (BPMN) [34] is an open XML standard for both describing the execution of a process as well as the graphical representation. The graphical representation is very similar to flow diagrams and the Activity Diagram from the Unified Modeling Language (UML) [49]. The goal of BPMN is to allow both technical users as business users to easily understand the described processes, while at the same time being able to express complex structures. The BPMN standard includes a partial mapping between BPMN and BPEL [48].

2.3 Model Driven Development

In other engineering disciplines than software engineering it is very common to develop a variety of specialised system models before actually building

anything. In software engineering, this is not quite common yet. Although programming languages have evolved greatly in the past years, the level of abstraction of general-purpose programming language never really changed much. Modern general-purpose programming language still operate on the level of conditional statements and loops.

With *Model Driven Development* or a *Model Driven Architecture*, the primary focus is on models and products, instead of computer programs [33]. The advantage is that models are expressed using concepts that are not bound to the implementation, but rather to the problem domain. However, many modelling techniques, such as UML class diagrams, hardly ever grow beyond a tool for the design phase and for documentation. Since there is no formal connection between the model and the implementation, the models are often not in sync with the implementation, which can lead to integration and maintenance problems. It is possible to generate code from class diagram models, but this only results in skeleton code. Model generation from code is also possible, but this usually does not result in the right amount of abstraction.

In order for a modeling language to be successful as a development tool, it must have some characteristics: it must have the right level of abstraction, it must be understandable, it must provide a true-to-live representation and must be predictable [39]. Also, constructing the model should not be more difficult than constructing an implementation. Such a language could take over the role of the implementation language, just like modern high-level programming languages displace assembly languages. Executable Business Process Modeling languages could be of use in a Model Driven Architecture, since they provide abstraction and provide a true-to-live representation. However, they still require a low-level implementation for some activities, so a gap between the model and the implementation remains [26].

2.4 Business Process Variability

While business process models offer a great deal of abstraction, the process models themselves are not flexible at all. With the widespread adoption of business process models, problems and needs emerge [12]. When trying to adapt an existing process to another situation, there are many potential issues [45]. The current practice, when adapting a process to a specific situation, is to create a copy of the original process and modify it [8]. This results in many very similar instances of the process. This introduces redundancy and makes managing these processes cumbersome. When a common element of the processes needs to be changed, all processes need to be adjusted manually.

It is important to make a distinction between *flexibility* and *variability*. With flexibility we mean the ease of adopting a certain process to a specific situation, where with variability we mean the ability to create several similar processes. With variability, some parts of business processes are variable or not yet fully defined. Flexibility and variability are two techniques that are of course closely related.

Both flexibility and variability are active research topics. Weber et al. have discussed patterns for process flexibility [47]. Schonenberg et al. have surveyed a number of approaches to create flexibility in process models [38]. There are two approaches to achieve flexibility: *imperative* and *declarative* methods. An imperative method focuses on how a task should be performed, for example by defining the order in which subtasks should be performed. A declarative method focuses on what should be done. This method uses constraints in order to restrict possible options.

With variability, all the processes share certain parts. These common parts are defined in a single, abstract process which is generally known as a *template*, a *reference process* or a *generic process*. The derived processes which are adapted to a specific situation are known as *variants*. The variation in the template is usually included through *variation points*, places in the process where changes may occur [10].

Aiello et al. describe two categories of variability techniques in the context of business process management [8]. *Design-time variability* deals with the variability of variants of a process at design-time. These techniques deal with defining common parts of processes and defining the parts that differ. The goal of these techniques could be avoiding redundancy in the design of several processes and restricting the variants. Techniques related to design-time variability are [11]:

Patching is defining a new process by describing all changes compared to another process. It allows changes that were not foreseen at design-time.

Blueprinting or design from template allows you to define a template and to extend it using special variation expressions.

Inheritance is a special form of using a template in which extension, specification or substitution can be used.

Late modeling uses templates with placeholders, which can be specified in a variant.

Run-time variability deals with variation of running processes. Changes in requirements, responding to errors or moving to another variant may be

reasons to use run-time variability. It could also improve the Quality of Service, for example by replacing an unavailable or inadequately preforming service with a good performing service. Both categories are important for the evolution of the processes. Changing requirements might affect the generic process as well as the variants.

Gottschalk et al. propose configurable workflow models based on YAWL [22]. Instead of creating explicit variants, a process can be configured to be used in different contexts. Configuration happens before run time. Actions in the workflow can either be *blocked*, *hidden* or *enabled*. If an action is blocked, it cannot be executed and the process cannot continue after this action. If an action is hidden it is simply skipped and if an action is enabled it is executed as normal. The authors have developed a configurable variant of YAWL called C-YAWL. Together with a configuration, a C-YAWL file can be converted into a regular YAWL file which can be executed using an execution engine. Although the implementation is build upon YAWL, the concepts are applicable in other execution languages.

COVAMOF is a variability management framework, developed to handle issues in variability management [40, 41, 43]. The framework is designed to model variability in software product families: software products which are quite similar, but are adapted to a certain context. It offer facilities to model variability over multiple layers of abstraction. COVAMOF works with variant points and variants. There are special elements to keep track of variation points at different levels of abstraction.

One way to implement variation points in service-centric systems is by using proxies: Web Services which choose, statically or dynamically, between possible variants and forward the request to those Web Services [19]. Koning et al. have proposed VxBPEL, an extension to the BPEL specification which introduces variability inside the BPEL process [42, 29]. The VxBPEL extension allows the specification of variation points (places in the process in which variability may occur) and variants (concrete alternative implementation) inside a BPEL process. A variation point can be placed anywhere inside VxBPEL code where an activity or container can be placed. The VxBPMN specification was designed to allow new variants to be added to the process model at run-time. The choice between variants is determined using the configuration of the process. In order to configure the processes and describe higher-lever variation points, it is possible to define *configurable variation points*. The authors have built a prototype based on the open source package ActiveBPEL. The configuration can remotely be changed using JMX. In order to keep the implementation simple, it is not possible to reconfigure a running instance.

Business process variability is an active research topic, and there are

many approaches to accomplish business process flexibility and variability. Which technique is the most suitable depends of course on the context of the problem. Imperative methods focus on how tasks should be performed, while declarative methods focuses more on what should be done. Variability can be applied for both design-time and run-time. Although the two types of variability are related, they often require other techniques.

2.5 The PVDI framework

Groefsema et al. propose the *Process Variability - Declarative'n'Imperative* (PVDI) framework [23] as a way to introduce variability in business process models. The aim of the framework is to allow a higher degree of process variability while preserving the main business goals of a process. The framework allows the use of both declarative techniques as imperative techniques to accomplish variability.

PVDI uses graphical elements that are quite similar to those of BPMN: Activities, events, gateways and arrows to indicate the flow. In addition to these normal BPMN elements, there are elements and properties that allow declarative and imperative variability. It is also possible to create groups of elements, in order to apply the variation properties on multiple elements.

Declarative variability techniques focus on what tasks are performed. In principle every variation is allowed, except for the variants that are explicitly disallowed. To accomplish declarative variability, elements can have the properties *Mandatory Selection* and *Mandatory Execution*. When an element has the Mandatory Selection property, it must be present in a variant. When an element has the Mandatory Execution property, the element must be present in every possible flow of the process. It is also possible to create edges that force *Ordered Execution*. It is possible to state that when a certain element is executed in the process, another element must or must not be executed. It is possible to indicate that these elements must follow immediately, or eventually. It is also possible to state that there should at least exist a path in which this happens, or that it should happen in every possible path. There is also an edge to indicate *Parallel Execution*. This makes sure that two elements are never in the same path.

Imperative techniques differ from declarative techniques by allowing no changes in variants unless this is explicitly specified. This makes the process less flexible, but makes the design process more straight forward. There should be designated areas, or groups of elements, in which imperative techniques could be used. There are two types of such areas in PVDI. There is a *Closed Area* in which no new elements may be added, and a *Frozen Group*

which may not be altered at all. In a Closed Area or Frozen Area however, it is possible to allow the removal of certain nodes. In a Frozen Area it is also possible to allow the moving, replacing and swapping of nodes.

It is also possible to define *constraints relations*: relation which indicate that if one element is included in a variant, another element may not be included. This relation can also be used with groups. The available relations are *Prerequisite*, *Exclusion*, *Substitution*, *Corequisite* and *Exclusive choice*.

Since a process is basically a directed graph, it can be used as a framework for a model logic of processes. In the PVDI framework, all constraints are defined as a *Computational Tree Logic*$^+$ (CTL$^+$) formulas [18]. This way, all constraints in the process model can easily be evaluated to see if all the constraints are satisfied, both in the template as well as in the variants. All these complex formulas are hidden from the end user by using easy to understand graphical elements.

A designer can, based on the requirements, create a template for a process. While designing, the designer can generate the CTL$^+$ constraints and evaluate the template to see if it is consistent. If the template is finished, it can be published and variants can be created. In the process of creating variants, constraints cannot be removed from the model. The variant can also be evaluated. Only if the variant is evaluated correctly, the process is a valid variant of the template. When requirements change, the template as well as the variants can be adapted. When the template changes, new CTL$^+$ constraints have to be generated and the template has to be republished. Variants can be re-evaluated. They can then either be evaluated correct, which means the variant is also a correct variant of the new version of the template, or they can be validated not correct, which means they have to be adjusted in order to become a correct variant of the new version of the template.

The authors have created a prototype of an editor which supports the creation of templates and variants. The editor includes code for the generation and valuation of CTL$^+$ constraints.

Chapter 3

Case study: The SAS-LeG project

This thesis is written in the context of the *Software As Service for the varying needs of Local eGovernments* (SAS-LeG) project [7]. SAS-LeG is a joint project of the University of Groningen, Cordys and local municipalities in the Netherlands. The scenario presented in this chapter is the typical case study for this project. However, it is presented here solely for demonstrating purposes. Business process variability management is a more generic problem which can be applied in more contexts, for example product families. The framework proposed in this thesis is applicable in a wide range of domains.

3.1 The WMO law

In 2007 the WMO law (*Wet maatschappelijke ondersteuning*, a social support law) was approved in the Netherlands. This law mandates, among many other things, rules for which needing citizens receive a publicly subsidised wheelchair. All municipalities in the Netherlands are required to have a desk where citizens can make requests for facilities offered by this law.

There are 418 municipalities in the Netherlands. All these municipalities have to implement this national law. However, they all have their own IT infrastructure. Also, they differ very much in size, methods and business models. The current practice of implementing such a law would be getting every municipality to get their IT staff to interpret the law. They then each have to model, implement and test this system[1]. Interpreting a law 418 times

[1]In reality there is a limited number of software vendors serving this market, so there probably is some reuse.

will probably introduce mistakes and redundancy, but more importantly, it will cost an enormous amount of effort and money.

But it gets even worse: national laws are changing all the time. When a law changes, it is possible that the IT systems need to be adjusted to accommodate these changes. This means that the IT staff of all the municipalities individually have to interpret the law again and adjust the IT systems. Since IT systems may differ between municipalities, the effects of the changes in the law can also be very different.

3.2 Implementing national laws using business process variability

When business process variability techniques are used by municipalities, much effort can be saved.

Not all municipalities have the same IT infrastructure. They have different software products from different vendors. Luckily, by using Web Services or similar techniques these packages can communicate and cooperate with each other, and can be used to execute business processes.

When a new law like the WMO law is introduced, all municipalities have to implement the law. This can be done by letting every municipality create their business process models for all the processes required by the law. Because every municipality works in it's own way, the implementations will differ a lot. But because all municipalities implement the same law, the commonalities between the processes should be large. When we have a lot of commonalities, we can probably benefit from variability techniques.

Instead of letting every municipality interpret the law and create business process models individually, we could let one person or team interpret the law and translate it into a business process model template. All the constraints in the law should be translated to constraints in the template, while parts that are unspecified by the law are left unspecified in the template or are free to be changed. To be short: the template must reflect the law.

When the business process model templates have been created and published, all the municipalities have to do is to create variants that can be used with their own IT infrastructure, methods and business models. When developing these variants, they can be evaluated with the constraints in templates. If the variant under development does not evaluate and thus is not a correct variant of the template, it probably also does not correctly implement the law. When the variant does evaluate, it must also be in line with the law.

When the IT infrastructure, methods or business model of a municipality

changes, the variant can be easily adjusted. As long as it is a correct variant of the template. When however the law changes, the template needs to be adjusted. The new version of the template needs to be published. The changes in the template also have to be applied in the variants. This can lead to two results: the new version of the variant evaluates and thus is still compatible with the new template, or the variant does not evaluate and has to be adjusted in order to be compatible with the new template. In the first case, it is possible to propagate the change automatically to the production environment, although in the context of national laws some user interaction is probably desirable. In the second case, someone of the technical staff will have to change the variant. It might be a minor change, but it could for example also require the municipality to change their business model.

In Section 4.4 we will discuss a concrete example of what the business process model of the WMO law might look like.

Chapter 4

A framework for modeling Business Process Variability

In Chapter 3 we have seen that the potential work and cost reduction of business process variability is enormous. Aiello et al. have listed requirements for an explicit variation handling framework which is desired in the case study [8]. The PVDI-framework (see Section 2.5) aims to satisfy these requirements in terms of variability constructs. However, when we look at the software tools, the gap between current Business Process Modeling tools and the PVDI-framework is large.

Many Business Process Modeling tools are build as a desktop application. Examples include the Microsoft Vision editor [30] and JBoss jBPM editor [28]. Many of these tools include a business process repository, in which processes can be stored and from which processes can be deployed. There are also tools built as a web application, such as JBoss jBPM. The software includes a web server which users can run on their own machine. The software is still distributed as a product. The editor can be accessed using a web browser, which makes it platform independent and accessible from many machines. There are however also web based tools which are hosted on a cloud platform and are delivered as a service, such as IBM Blueworks Live [27] and the Cordys Business Process Management Suite [16]. This way the user does not have to care about hosting the tools or creating back-ups. All processes are stored in a central repository, making collaborating easier.

Our goal is to make Business Process Variability easy to use for the end user. In order to achieve this goal we have formulated five new high-level requirements compared to the existing PVDI-framework prototype editor. One of the research goals of this framework is to provide a version control system for business process models with variability. This has resulted in two functional requirements. In order to bring the framework up to par with

existing solution, we have formulated three non-functional requirements.

FR-1 The relation between templates and derived variants should be tracked. When a template is updated, it should be easy to update a variant.

FR-2 Changes in the models should be tracked. It should be possible to view a previous version of the model.

NFR-1 Scalability: The solution should be able to accommodate a large amount of users. Thus, the software must perform well and must be scalable.

NFR-2 Usability: There should be a user-friendly environment for modeling templates and variants. This editor should be attractive and easy to use.

NFR-3 Reliability: Business process models should be stored safely and always be accessible. A single failed machine should not be the cause of lost data or interrupted services.

In this thesis we propose a framework for modeling business process variability which complies with these requirements. In this chapter the principles of the proposed framework will be discussed. In order to demonstrate these principles, we have implemented a web application. In Chapter 5 the details of the implementation will be discussed.

4.1 Web applications

Traditionally, applications run on the computer the user is working on. The application is distributed using for example CD's, DVD's or can be downloaded using the Internet. However, in the last couple of years *web applications* are becoming more and more popular. Web applications are not applications that run on the local machine, but are essentially dynamical web pages displayed by a web browser which behave like a normal application.

Web pages are documents that can be downloaded on demand from a server based on a URL. Web pages can, among other techniques, be made dynamic using JavaScript. JavaScript code is executed locally by the web browser. In the last couple of years, the performance of JavaScript execution by browser has been dramatically improved, making it possible to create more complex web applications. With the introduction of techniques such as *Scalable Vector Graphics* (SVG) [46] and the *Web Graphics Library* (WebGL) [24], it is even possible to create rich visualizations inside the web browser.

Web applications have two major advantages compared to normal applications. First of all, Web applications can be accessed from any computer with a modern browser and an Internet connection. It is not necessary to install the application and it does not matter which hardware or operating system is used. Secondly, Web applications usually don't work with files that have to be stored on the local hard drive. Instead, all data is stored in a database on the server. Users can log on to their account from anywhere to access all files. This means that the user does not have to worry about storing files, taking them with him or her or creating backups. The files are always available. Another advantage is that the software is always up to date. The code that runs inside the browser is always downloaded on-demand, so only the machines of the vendor need to be updated.

Of course, there are also a few disadvantages. First of all, the Web applications are normally not available when there is no Internet connection available. Also, the user always depends on the provider of the application. When the provider has problems, for example a power outage, the application is not available to the user. Finally, the data stored in the application could be confidential. Not everyone appreciates that the data is stored by a third party.

One famous example is Google Docs, which recently has become a part of Google Drive (`http://drive.google.com`). Google Docs is a complete office suite, featuring a word processor, a spreadsheet editor, a presentation program and a drawing application. It does not run on the local machine, but is a collection of dynamic web pages which offer all features that are expected from a modern office suite. Documents are stored in the users Google account. It is possible to share documents with other users, making it easy to collaborate.

4.2 Graphical editor

The main part of the framework is the graphical editor. The editor allows the user to model business processes. A graphical representation makes business process models easier to comprehend. As explained in the previous section, the editor is created in the form of a web application.

4.2.1 User interface

Figure 4.1 shows an overview of the graphical editor. On the top the menu, the title of the process and the *Check model*, *Undo*, *Redo* and *Create new tag* buttons can be seen. In the top on the right side the status of the document

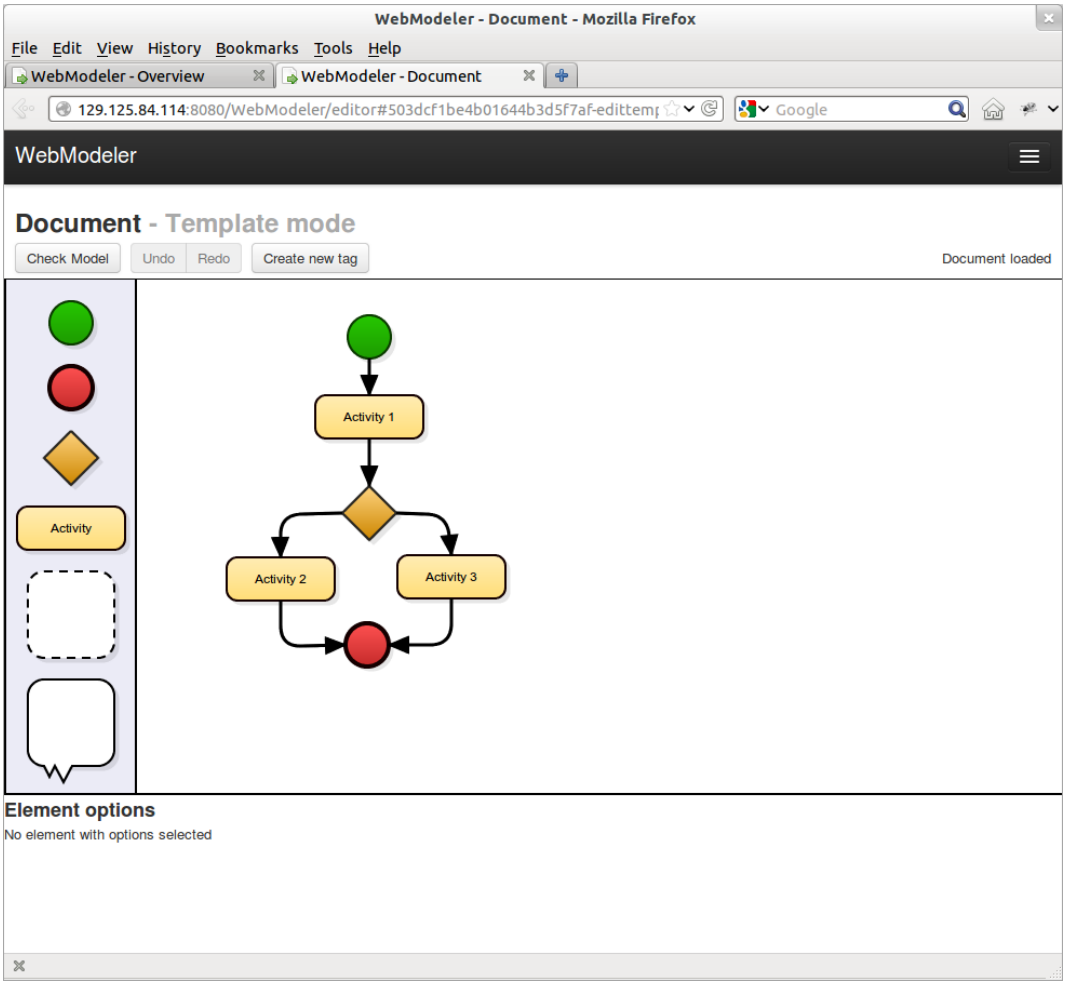


Figure 4.1: The main screen of the graphical editor in the web browser Mozilla Firefox.

can be seen. This can either be *Document loaded* (the document is loaded from the server, no changes are made yet), *Saving...* (changes are currently being submitted to the server) and *Saved* (changes have been successfully saved).

In the centre the current business process model can be seen. On the left are prototypes for new shapes. When the user wants to add new shapes to the model, he or she can just drag these prototypes to the model.

On the bottom the *options menu* can be seen. Most elements have options. For example, an activity has a name. Each type of element has it's own options menu. The options of the element which was last clicked on is shown.

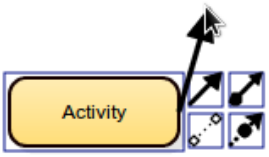


Figure 4.2: Selected shape.

The user can select a shape by clicking on it. It is possible to select multiple items by holding down the Control-button and clicking multiple elements. As can be seen in Figure 4.3, it is also possible to select multiple elements by dragging. When a shape is selected, a box is drawn around it to indicate the shape is selected. The *new edge menu* also appears (Figure 4.2). There are buttons visible for the different type of edges. The user can create a new edge by dragging one of those buttons to another shape. Once the user starts dragging, a new edge appears between the selected shape and the mouse cursor. If the user releases the mouse button above another element the new edge is created. If there is no shape underneath the cursor the edge disappears.

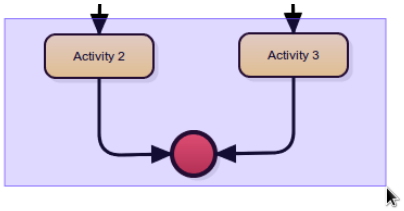


Figure 4.3: Selecting multiple elements by dragging.

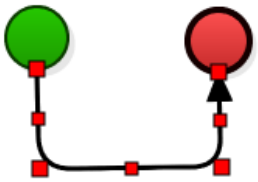


Figure 4.4: A selected edge which has multiple segments.

When an edge is selected red squares appear on both ends. By dragging these squares, the head or tail of the edge can be moved. This way the source shape and the target shape can be changed. When the user stops

the dragging when there is no new shape underneath the cursor is, the edge snaps back to it's previous position.

There is also a smaller red box at the center of the edge. By dragging this box, a new *segment* is created (see Figure 4.4). This way the position of the edge can be changed, which can be very useful for keeping business process models clear. When creating new segments, the small red box become larger and new small red boxes are created in order to create more segments. Bézier curves are used to create smooth corners.

4.2.2 PVDI elements

Figure 4.5: Normal BPMN elements which can be used in the editor: a start event, an end event, a gateway and an activity.

The editor uses the PVDI framework as basis. As explained in Section 2.5, the PVDI framework uses elements that originate from the BPMN language. A business process model represents the possible flows of a process. This flow always start with a single start event. Then it can go to an activity, where an action can be performed. It could also go through a gateway. At a gateway the flow splits in multiple flows, or multiple flows come together. All flows end at the end event. In the PVDI framework, every process has exactly one start event and one end event. The graphical representation of these elements can be seen in Figure 4.5.

Figure 4.6: Four different states of an activity. The activities on the bottom are mandatory, the activities on the right execution is mandatory.

In the PVDI framework, certain shapes can have states. They can be mandatory, which means that the element must be present in a variant of the process. The shape can also have mandatory execution, which means that the activity must be executed in every possible flow. The graphical representation of these states can be seen in Figure 4.6.

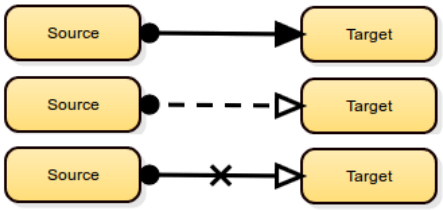


Figure 4.7: Three examples of an Ordered Execution Relation.

It is also possible to create Ordered Execution Relations in order to control the flow (see Figure 4.7). There are several combinations possible. A dashed line indicates that the source element must be followed by the target element eventually. A solid line indicates that the source element must be followed by the target element directly. A white arrowhead indicates that there must be at least one flow in which the source element is followed by the target element, where a black arrowhead indicates that this should happen at every possible flow. Finally, when there is a cross at the middle of the edge, the constraint is negated, which means that the source element may not be followed by the target element.

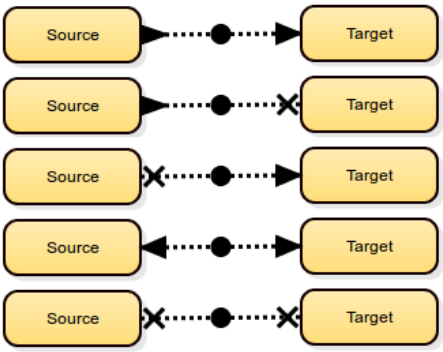


Figure 4.8: The different types of Constraint Relations. From top to bottom: *Prerequisite*, *Exclusion*, *Substitution*, *Corequisite* and *Exclusive choice*.

Constraint relations can enforce the presence of elements based on the presence of another element (see Figure 4.8). There are five types of Constraint Relations:

Prerequisite When the source element is included, the target element must be included as well

Exclusion If the source element is included, the target element must not be included

Substitution If the source element is not included, the target element must be included instead

Corequisite If the source element is included, the target element must also be included and vice versa

Exclusive choice If the source element is included, the target element must not be included and vice versa

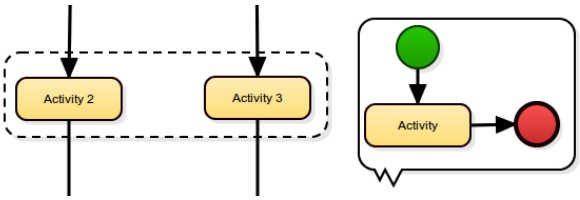


Figure 4.9: A normal group (left) and a frozen group (right).

Finally, Figure 4.9 shows the two types of groups: a normal group and a frozen area. Using these groups, constraints can be applied to multiple elements at once. A frozen group is a sub process which cannot be changed, unless specifically specified.

4.2.3 Valuating constraints in business process models

On top of the editor there is a *Check model* button. When the user clicks this button, the model is evaluated: all constraints are checked. All valuation errors are reported to the user. There are two types of errors: errors that apply to the whole model (for example if the model has a dead end) and errors that apply to a certain element (for example a constraint that is not satisfied). Errors that apply to the whole model are shown directly when the user evaluates the model. Elements with errors are displayed in red. Clicking them shows the error message.

Figure 4.10 shows an example of a business process model with several errors. The edge between Activity 1 and Activity 2 indicates that they should be executed parallel. They are not; they are executed after each other. Activity 2 and Activity 3 are however executed in parallel. The edge between Activity 1 and Activity 3 indicates that there should be an execution path in which Activity 1 is eventually followed by Activity 3. This is also the case. Activity 2 is mandatory. Since Activity 2 is present in the model, that is fine too. Activity 3 should be executed in every possible execution path. This is not true, Activity 2 could be executed instead of Activity 3. Finally,

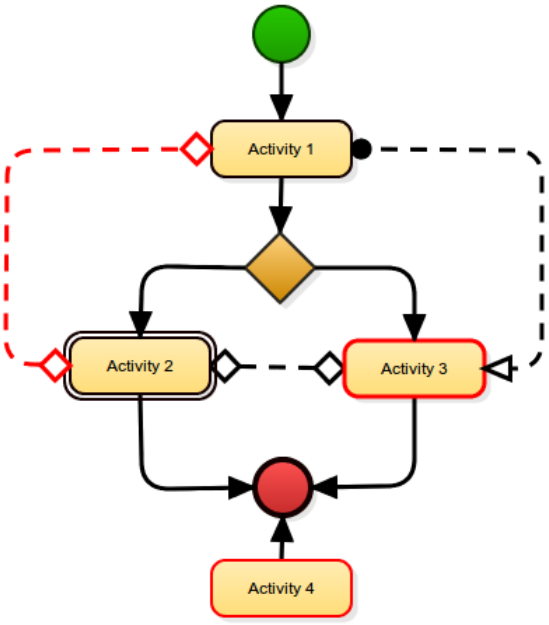


Figure 4.10: A business process model with three valuation errors.

Activity 4 can never be reached from the start event, which also results in an error.

4.2.4 Modes

The editor can be opened in several *modes*. The mode indicates which actions the user is allowed to take. First of all, there is the *view mode*. In this mode the user is not allowed to do anything; the model can only be viewed. The *template mode* can be used to create a template. When the user is creating a template, the user is allowed to do anything. Finally, there is the *variant mode*. In this mode the user can only make changes which are allowed by the template. For example, if an element is mandatory is a template, it cannot be removed in variant mode.

4.3 Version control

Version control, or *revision control* is the management of changes in collections of information. It is for example used to manage collections of documents, the source code in a software project or content in a content management system. Changes are usually stored as *revisions*, which also includes the date and author of the changes. Revisions can usually be restored

and compared with each other, allowing users to track each other's edits and correct mistakes.

For this framework, we have developed a version control system for storing business process models. One advantage of version control is that it allows the user to keep track of the history of files. When you make a mistake, it is easy to view a previous, working, version. But there is one more advantage: it makes collaborating on documents easier.

Some of the ideas used in this version control system are based on Git, a distributed source code management system.

4.3.1 Git

Git [2] is a open source distributed version control system for managing source code. Git is mainly known for it's distributed nature and the ease of creating and merging branches [13]. The interesting part is how Git stores the history of the files it manages. Git stores the revisions, or *commits*, in a directed graph. When a single user creates a few commits in a row, Git internally creates a chain of commits. Whenever a branch is created, a commit has two children. When two branches are merged, a commit has two parents. A branch is essentially a pointer to a commit in this directed graph, and thus very easy to create or delete.

4.3.2 Work flow

When we look back at the scenario described in Chapter 3, we can image a typical work flow for the framework. Imagine two users of the system: the legislator, maintaining a template, and the municipality, maintaining a variant of the template.

When the new law is approved by the national government, the legislator creates a template of the law. This template should include all the elements and constrains that the law demands. Creating a template should be done very carefully, so this takes some time. When the legislator has finished the template, it creates a *tag*: a named version capturing the state of the template. This could be, for example, `Version 1.0`. The legislator than shares the template with the municipality.

When the template is shared, the municipality can view the template. This way the municipality can determine a way to implement the law in it's own organisation. The municipality can than create a variant, based on a tag from the legislator: `Version 1.0`. The municipality can make changes to the template in order to create the variant: it can, for example, create new elements or move or remove elements.

When the legislator finds out that it did not implement the law entirely correct in the template, it has to make changes and create a new tag: `Version 1.1`. The municipality then has to create a variant based on the `Version 1.1`, but they created a variant based on the `Version 1.0` tag. In order to create a variant that is based on tag `Version 1.1` without the municipality has to start all over again, the existing variant and the new template have to be *merged*: the changes that were done to create the initial variant have to be applied to tag `Version 1.1`. The result is a new variant, based on `Version 1.1`. The merging process is explained further in Section 4.3.4.

By keeping track of which version of the template is used for the variant it is easy to see if the variant is still up-to-date. By being able to merge the changes of the updated template with the changes made in the variant it also becomes easier to stay up-to-date: the process of updating a variant is automated. It is only necessary to manually change the variant if the variant violates new or changed constraints.

4.3.3 Storing models

A business process model is basically a set of elements with properties. An element can for example be an activity. The title and the location of the activity are properties of this activity. When we want to keep track of the history of a model, we don't need to save the state of all the elements in model. Instead, we save all the changes the user made to all elements. This way we can reconstruct the state of the model at any point in time.

There are three types of possible changes: the user can create a new element, the user can change the properties of an existing element and the user can delete an element. When creating a new model, the user starts with an empty model: no changes have been made yet. From there on, the user can add new elements, change their properties or delete them.

But the set of changes does not necessarily has to be a linear data structure. When a template is shared with another user in order to create a variant, we don't have to create a copy of the complete history of the data. Instead, we could store the set of changes as a tree. The changes that were used to create the template can be shared. When a certain version of the template is shared with another user in order to create a variant, we can create a branch: the original user can continue to develop the template, while another user creates a variant.

Let us look back at the scenario described in section 4.3.2. There were two users: the legislator, maintaining a template, and the municipality, maintaining a variant. Figure 4.11 shows a possible tree structure storing the full history of the model created in this scenario. The legislator starts with an

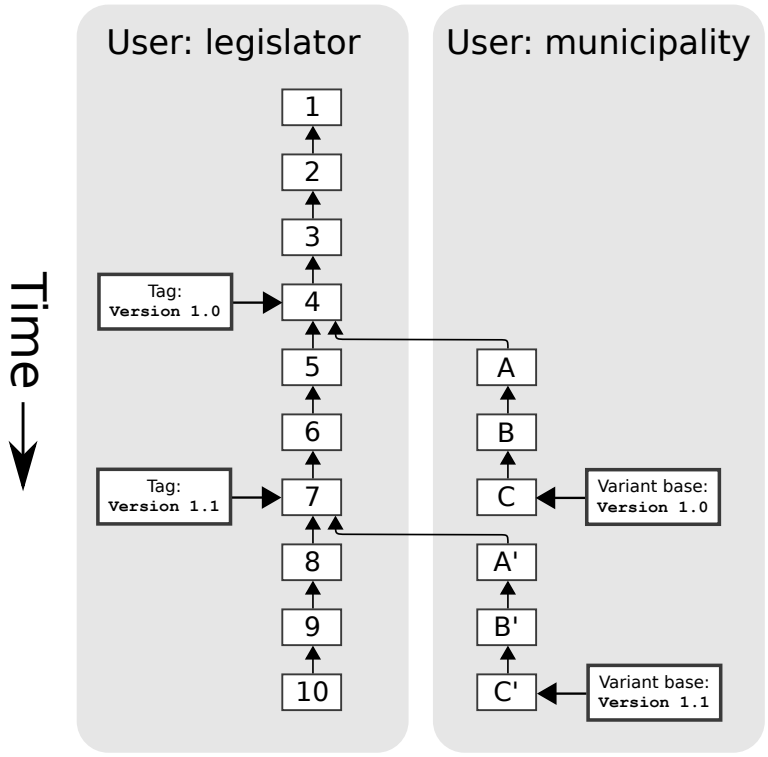


Figure 4.11: Storing changes in a tree structure. The small white boxes represent a modification of the model. The numbers and letter are only for clarification purposes.

empty document. Then the legislator makes some changes. After the fourth change, the legislator decides to create a tag called `Version 1.0` and the document is shared with the municipality. From this point on both users can continue in parallel from the fourth revision: the legislator can continue to develop the template, while the municipality can create a variant based on `Version 1.0`. This means our tree structure has a branch at this point.

After change 7, legislator decides to create another tag: `Version 1.1`. This means that municipality has to create a variant which is based on the new tag. Both set of changes have to be merged. The municipality created changes A, B and C. In order to merge both models, the changes A, B and C have to be applied from change 7. Unfortunately, this can not always happen without conflicts. For example, the legislator could have made an existing activity mandatory somewhere between `Version 1.0` and `Version 1.1`, while the municipality decided to remove the activity in the variant. In order to create a process which is based upon the template, the merging process creates new changes A', B' and C', which are based upon A, B and

C. The merging process can result in a model which does not evaluate or does not fulfil the requirements of the municipality. It is the task of the municipality to address these issues.

The tree structure is stored internally by providing every change with a link to it's predecessor (or parent). This way, when we know the last change, we can easily reconstruct the entire model by following the chain of predecessors until we end up with the initial change.

4.3.4 Merging models

Merging is the process of applying two sets of changes to an existing model. From a common base tag two sets of changes are created: the changes made in the new tag compared to the base tag and the changes made by the variant compared to the base tag.

Unfortunately, merging can sometimes result into conflicts. There are several types of conflicts which can occur. For example, when in the new tag an activity is moved to the left, and in the variant the activity is moved to the right, there is a conflict in the merging process.

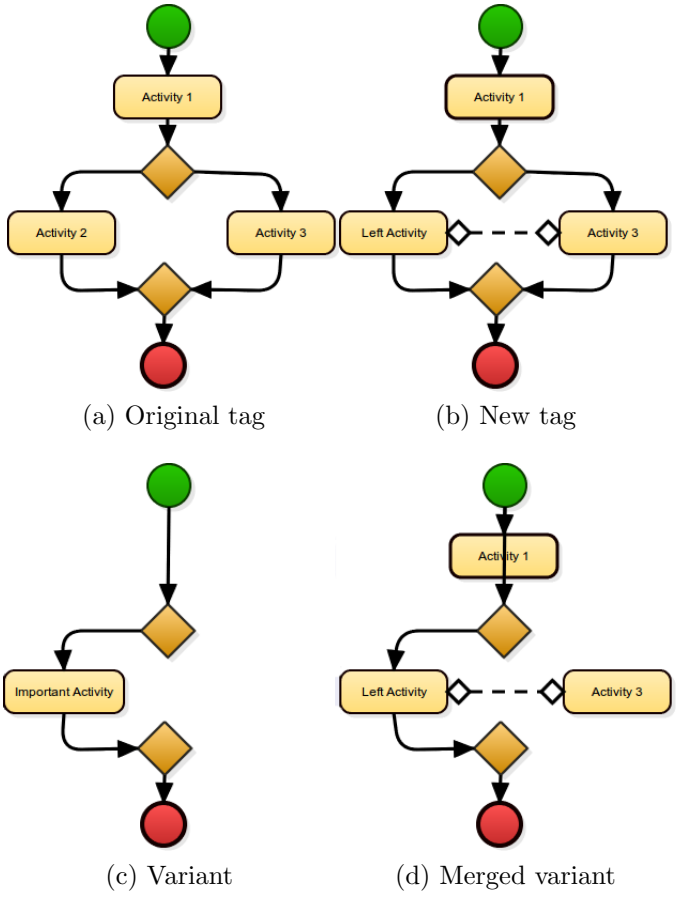
The constraints from the template should always be maintained. When a constraint disappears in the merging process, the user could wrongfully assume that a variant satisfies the constraints from the template. Because of this, when a conflict emerges, we always have to choose the changes that were made by the template.

Figure 4.12 shows an example of the merging process. The original tag is a simple business process model: First an activity is called, then the process branches with in each branch an activity, and then the branches come together and the process ends. There are no special constraints in this process model. After this tag is created, both a new tag and a variant are created in parallel.

In the new tag 'Activity 1' has the mandatory execution constraint. 'Activity 2' has been renamed to 'Left Activity'. Also, 'Left Activity' and 'Activity 3' have the parallel execution constraint.

In the variant other changes are made. 'Activity 1' and 'Activity 3' are removed, together with the incident edges. Since in the original tag 'Activity 1' did not have the mandatory execution constraint, the user was allowed to remove the activity. 'Activity 2' has also been renamed in the variant, but this time it is renamed to 'Important Activity'.

In the merged process model we can see that 'Activity 1' appeared again. Because in the new tag 'Activity 1' has the mandatory execution constraint, it must be present in the merged process model. In both the new tag and the variant 'Activity 2' has been renamed. In case of such conflicts, the value



(a) Original tag (b) New tag

(c) Variant (d) Merged variant

Figure 4.12: Example of a merged variant

of the new tag will be used. We can also see that 'Activity 3' appeared. In order for the parallel execution constraint from the new tag to be present in the merged model, 'Activity 3' is needed. However, the edges connecting 'Activity 3' with the gateways are not present. This is because they were removed in the variant and they are not required for any constraints in the new tag. Because of this, the merged variant does not evaluate, since 'Activity 3' is not reachable. Fortunately, this issue can easily be solved by the user.

The merging process will always maintain the constraints from the template. In case of conflicts, the new version of the template will always be chosen. Merging will always result in a process model with the same constraints as the template, but will not always result in a correct process model. This is however inevitable and these issues must be addressed by the maintainer of the variant.

4.3.5 User interface

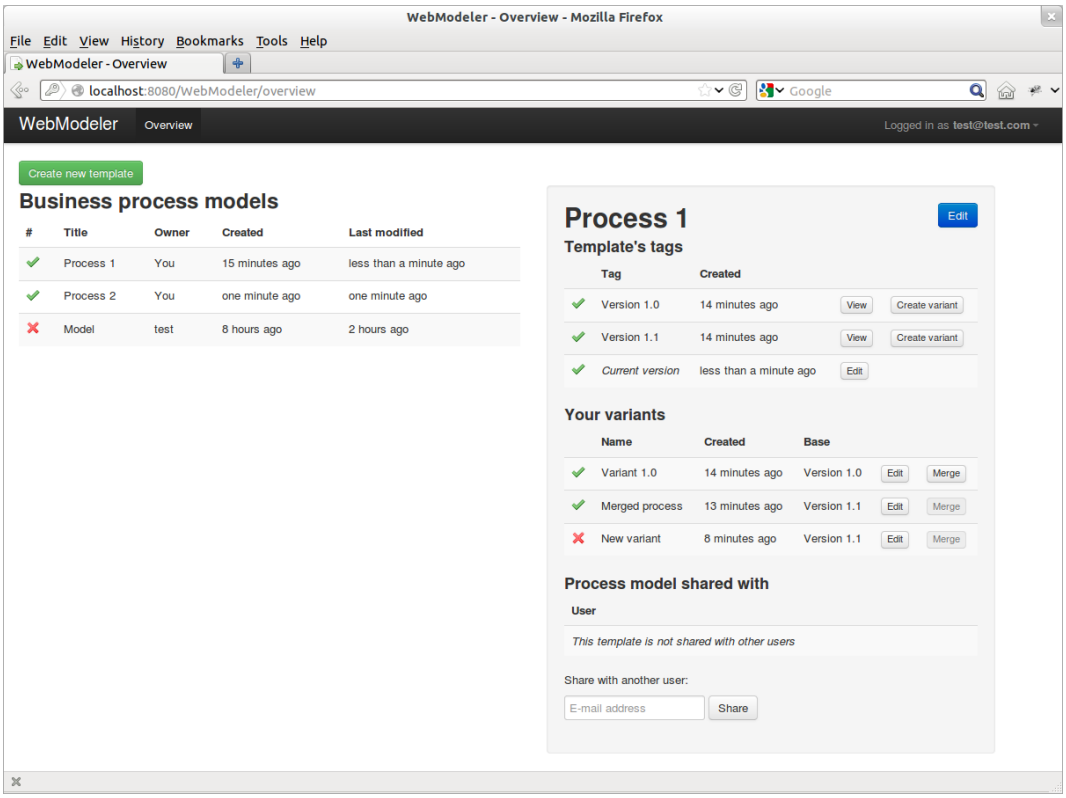


Figure 4.13: The overview page in the web browser Mozilla Firefox.

When the user logs in, the *overview* page is shown (see Figure 4.13). This page is divided in two columns. In the left column all the business process models the user can edit are shown. These are business process models the user created, but also business process models that other users have created and are shared with the current user. A green mark or a red cross indicates whether the latest version of this model evaluates. Also the owner of the model, the date the model is created and the date model was last edited is shown.

When the user clicks on a business process model in the left column, the details are shown in the right column. There are three tables shown. Again, a green mark or red cross indicates whether a model evaluates. In the first table all the tags are shown. Tags can only be created by the owner of the document in template mode. All users with which the business process model has been shared with can view tags and create a new variant based on this tag.

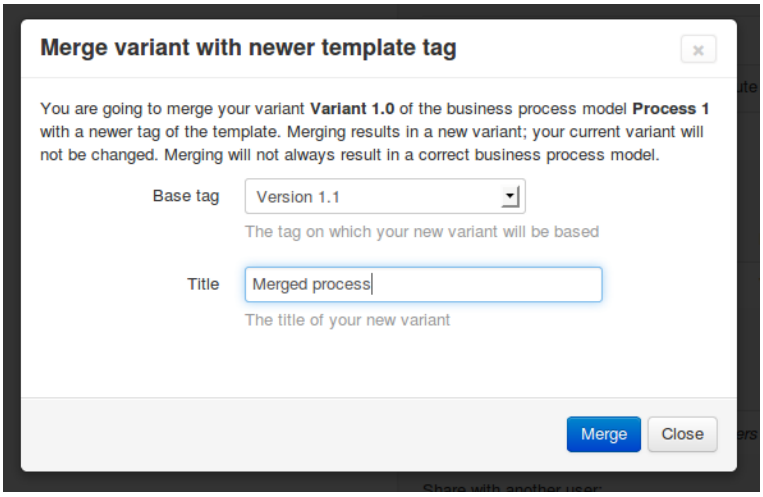


Figure 4.14: The dialog for merging a variant with a newer version of the template.

In the second table the variants are shown. Variants are private to the user. All variants can be edited. If a variant is based upon an old tag, it is possible to merge the variant with a newer tag. Figure 4.14 shows the dialog for merging. The tag on which the new variant is based can be selected from a drop-down menu. After entering a title for the new variant and confirming a new variant appears in the table. The original variant is not changed by the merging process.

In the third table all the users with which the business process model is shared are shown. When the current user is the owner of the model, users can be added and removed.

4.4 Example: The WMO Law

In Chapter 3 we have discussed the case study of the WMO law. In this section we will discuss a concrete example of a possible implementation of this law using the proposed framework.

Figure 4.15 shows a possible template for the WMO law. We can see a frozen area, which may not be altered by a variant, and several Ordered Execution constraints. For example, when an application is received, there should eventually be a path leading to the adding of application to the system. When the application is added to the system, the frozen area should always eventually be executed. When a citizen is notified about a decision, the citizen must always be able to object.

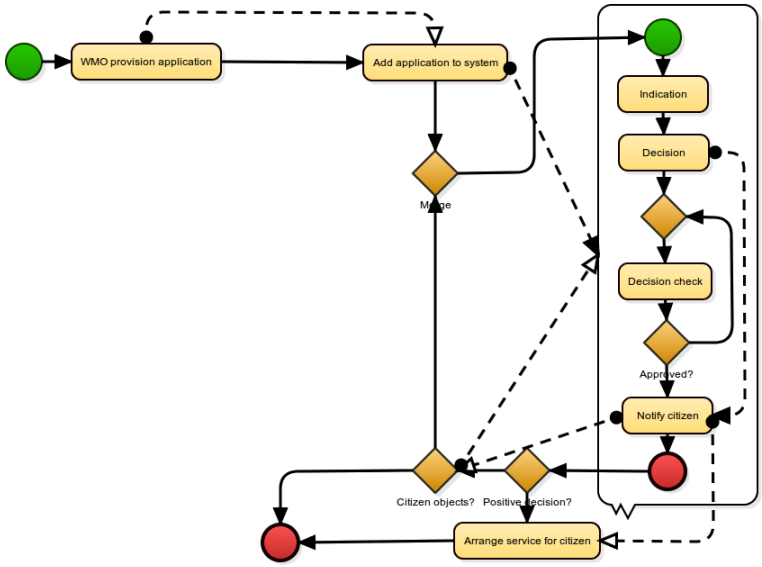


Figure 4.15: A possible template for the WMO law.

Figure 4.16 shows a valid variant of the template. The frozen area has not been modified and all the Ordered Execution constraints are still satisfied. However, some additional steps have been added. After the application is added to the system, it is checked whether the situation of the citizen is known. If not, a home visit is arranged. If then there is any doubt about the handicap, medical advice is sought. After these additional steps, the process continues as described in the template.

Figure 4.17 shows a newer tag of the template. In this tag a new gateway and a new activity have been added in order to process applications which do not qualify. If the application does not qualify, the citizen is advised and the process is ended without adding the application to the system. An extra Ordered Execution constraint and a parallel constraint have been added.

Finally, Figure 4.18 shows the result of merging the variant with the new process model. Since no elements were edited by both the variant as well as the new template, no conflicts have emerged in the merging process and the resulting process model evaluates.

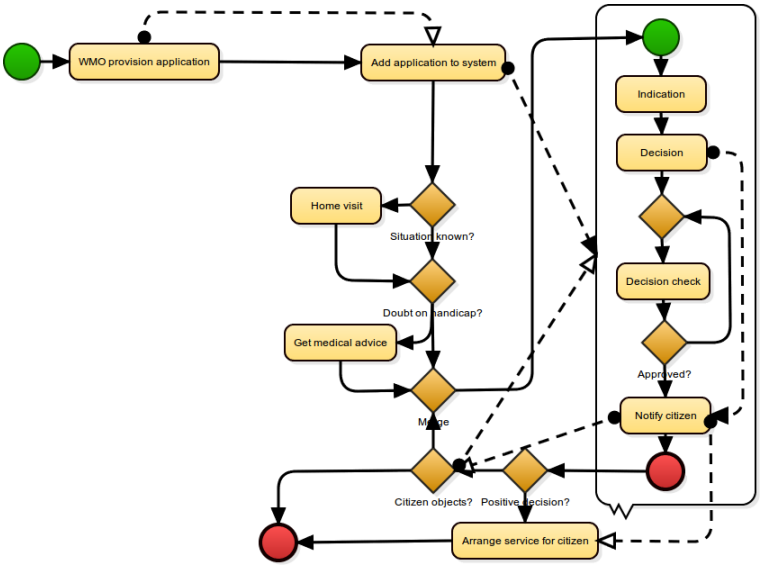


Figure 4.16: A variant of the template shown in Figure 4.15.

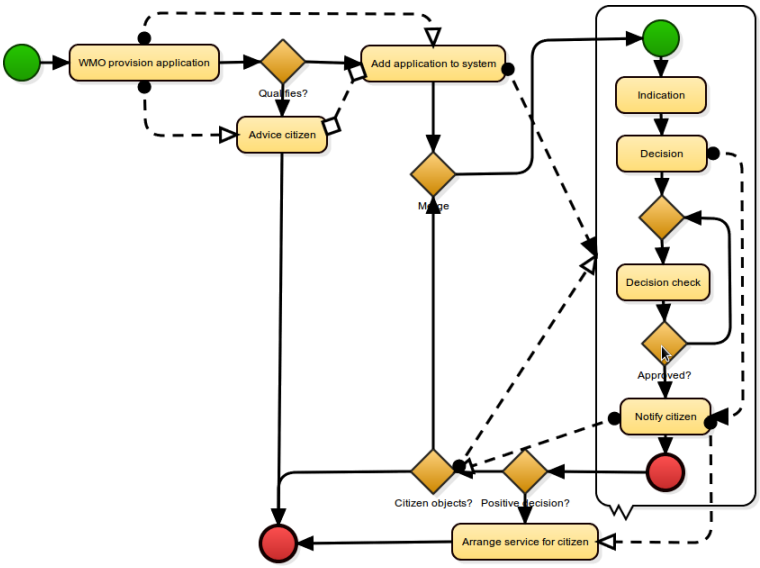


Figure 4.17: A new version of template shown in Figure 4.15.

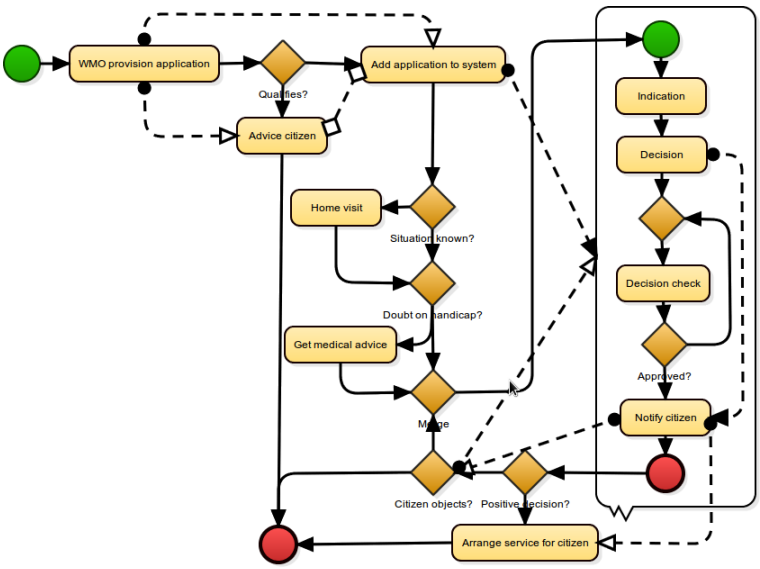


Figure 4.18: The business process model resulting from merging the template shown in Figure 4.17 and the variant shown in Figure 4.18.

Chapter 5

Implementation

5.1 Web applications

As we have seen in Section 4.1, a web application is essentially a dynamic web page which behaves as an application. In this section we will look at web applications from a more technical point of view.

The user uses a web browser to visit web pages. The user inputs an *Uniform Resource Locator* (URL). For example, if the user wants to visit the website of the University of Groningen, it uses the URL `http://www.rug.nl`. The web browser then creates a connection with the web server of the University of Groningen, requesting the homepage. The web server then responds with the contents of the home page. This page could contain references to other objects which are necessary to display the home page, such as images, style sheets or news feeds. The web browser automatically also requests these objects. These objects are also requested based on an URL.

The web browser and the web server usually communicate through the *HyperText Transfer Protocol* (HTTP) protocol, or the encrypted alternative *HyperText Transfer Protocol Secure* (HTTPS). These protocols are stateless request-response protocols: the web browser requests an object and the server responds with the data of this object. It is also possible to send data along with a request. For example, when submitting a form or uploading a file to a website, the web browser sends a request to the web server which contains data.

The web server responds by sending data. This can either be static data, or it can be generated dynamically. For example, the logo on the home page is probably static, but the news items on the home page are most likely dynamically generated, based on data stored in a database. As we have

already seen, web pages can be made dynamic using the scripting language JavaScript.

We can divide a web application in two parts: The *server-side* and the *client-side*. With the server-side we refer to all the operations done on the server or servers. This part of the application responds to requests generated by web browsers. This is the place where for example account data is stored. The client-side runs on the user's computer. This part is responsible for the user interface.

5.1.1 Ajax

For a long time, web applications were just static web pages which were dynamically generated by the server. JavaScript could be used to make the static web page somewhat dynamic. Ajax (acronym for *Asynchronous JavaScript and XML*) is a technique used on the client-side to create asynchronous web applications. With Ajax, a web page which is already loaded can make an HTTP(S) request using JavaScript. This way a web page can be updated with new information from the server, without the need of reloading the entire web page. This way, the client-side code and the server-side code can communicate.

Although the X in the name *Ajax* comes from XML, the transmitted data does not necessarily needs to be in XML format: plain text is transmitted. To transmit data structures, the data needs to be serialised. This can of course be done using XML, but *JavaScript Object Notation* (JSON) is also commonly used. If desired, also plain HTML can be transmitted so it can be inserted directly in the existing web page.

Ajax allows the client-side to make requests to the server-side. This can be done to request data, submit data or do both. Ajax does not allow the server to contact the client. So if the clients wants to receive updates from the server, a technique like polling has to be used: the client regularly contacts the server to see if there are any updates. There do however exist workarounds that implement server pushing. Common names for these techniques are *Comet*, *Ajax Push*, *Reverse Ajax* or *Long polling*.

5.1.2 Saving changes

In order to save the business process model as the user is working on it, the client-side and the server-side have to cooperate using Ajax. As we have seen in Section 4.3.3, the user generates *change*-objects while working on a business process model. For example, if a gateway is moved, the `x` and

`y` location of the gateway object are updated. Whenever the user makes changes, they are saved directly on the server.

All the elements in business process model (e.g. activities, gateways and flows) inherit from the abstract class `Element`. As shown in Figure 5.1, this class has the functions `get`, `set` and `commitElement`. These functions can be used to store and retrieve persistent properties of the elements.

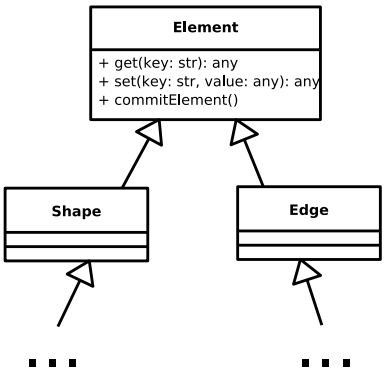


Figure 5.1: Highly simplified class diagram of the elements in the business process model.

Figure 5.2 shows what happens when the user makes changes to the business process model. For example, while dragging an object, the `x`-location and the `y`-location properties are constantly changing. When the user ends the dragging, the object has a new location. The changes can be committed. From there on two things happen.

First of all, to implement the undo and redo functionality, we have to store a history off all the activities of this session on the client-side. When the user ends the interaction, the changes of the element are committed. But it could be that multiple elements were updated in this interaction. For example, if the user wants to move multiple objects, the user can select multiple objects and drag them together. So when the interaction finishes, all changes have to be collected in an `Action`-object.

The `Action` is stored in the *Undo/Redo stack*. Every time the user does something, an `Action`-object is stored at the top of the stack. There is a special pointer that indicates the `Action`-object that represents the current state of the model. When the user pushes the Undo-button, the last `Action`-object is not removed from the stack, but the pointer is moved to the previous `Action`-object. This way the user can redo the last action. When the user performs an action, all `Action`-objects above the pointer are removed.

Besides storing changes in the Undo/Redo stack, they have to be stored in the database on the server-side. Using Ajax, they could all be submitted

to the server directly. However, if the user is busy he or she may generate multiple changes per second. This would generate a high number of HTTP(S) requests.

By using a small buffer, we can easily reduce the number of requests. All new changes are placed in the *change buffer*. Changes are stored for a maximum of three seconds in this buffer. This way changes are still sent almost instantly. After three seconds, they are moved to the *send buffer*. The objects are serialized using JSON and submitted to the server. The server than sends a confirmation to the client. When the client receives a confirmation, the changes have been saved successfully and the send buffer is emptied.

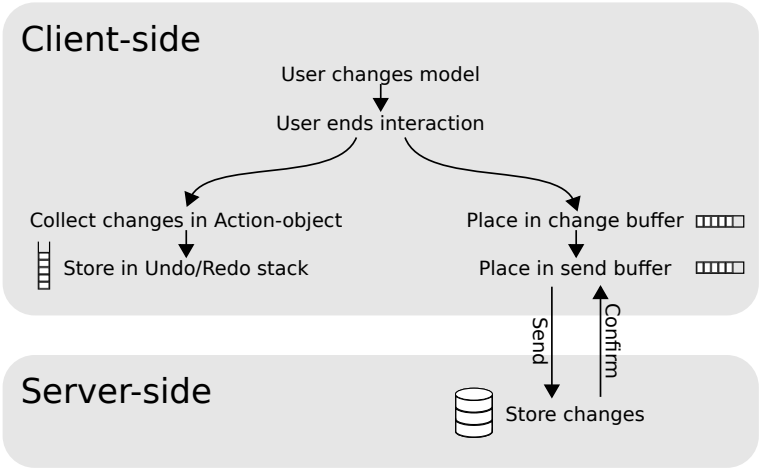


Figure 5.2: Saving user generated changes.

5.2 Client-side

The client-side is the part of the applications which runs inside the web browser of the user. As already discussed briefly before, there are several techniques involved on the client-side:

HyperText Markup Language (HTML) is a markup language for describing the contents of web pages.

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation and formatting of HTML documents.

JavaScript is a prototype based scripting language which can be used to create dynamic websites. JavaScript supports imperative, object-oriented

and functional programming styles. The language is standardised under the name *ECMAScript* [1]. JavaScript is interpreted and executed by the web browser.

5.2.1 Used libraries

In order to create the user interface, we have used a collection of freely available libraries. The main advantage of these libraries is their cross-browser support. There are several popular web browsers available. Unfortunately, these browsers do not offer the same support for standards or do not implement them in the same way. Cross-browser libraries usually offer a simple interface, with different implementation for different browsers. Using libraries also reduces development time, since they usually offer functions for common problems. The editor greatly depends on Raphaël for the graphics.

Raphaël

Raphaël [6] is an open source cross-browser JavaScript library for creating vector graphics. It also provides advanced functions for geometry, event handling, dragging and animation. Raphaël can be used to draw vector graphics inside a canvas HTML element. In order to draw the graphics, Raphaël will use in most browsers SVG, or VML in older versions of Internet Explorer. The main advantage of using Rapaël is the cross-browser support and allowing JavaScript code to easily modify the graphics.

jQuery

jQuery [3] is a cross-browser JavaScript library designed to simplify client-side scripting. It mainly simplifies selecting, modifying and traversing elements from the HTML page as well as it's styling. It also includes Ajax and utility functions. jQuery offers a fluent interface which makes code compact and easier to write.

Twitter Bootstrap

Twitter Bootstrap [44] is an open source collection of tools for creating user interfaces for websites and web applications. It is created and maintained by Twitter. It is a collection of HTML, CSS and JavaScript. It includes standard styles for layouts, typography, forms, buttons and navigation. It offers a collection of complex controls which are popular, but not supported

by HTML, such as drop-down menus and autosuggestions. The main advantage of Twitter Bootstrap is the ability to create an attractive looking user interface in a very short time.

5.2.2 Design

The client-side of the application is very JavaScript intensive. The editor is essentially one web page. The editor page consists of three parts: The top shows the title of the page and document and provides an undo and redo button, a button to evaluate the model and in the case of editing a template a button to save the current version as a tag. Underneath is a big canvas-element in which the business process model elements are drawn. The left panel is also part of the canvas-element. At the bottom there is an options panel. This panel contains many forms for adjusting prototypes of the business process elements. Depending on which element is selected, the right form is shown. Raphaël is used for drawing and dragging all the business process model elements.

The JavaScript code is divided in three parts: The model representing the business process model, the user interface and the communication with the server-side. The client-side is developed in an object-oriented fashion.

Model

Every business process model has one `model`-object, which represents the complete business process model. As we have already seen in Figure 5.1, all elements in the model inherit from the abstract class `Element`, which takes care of keeping track of changes. The `model`-object has functions which apply to the complete model, such as adding and removing elements or selecting all elements inside a certain area.

`Element` has two subclasses: `Shape` and `Edge`. Shapes are objects that can be dragged, such as activities, events and gateways. Edges connect shapes. `Shape` has for example functions to decide whether the current shape is selected, while `Edge` has function to determine how to draw a line from one element to another.

In order to show the elements in the user interface, they all own one or multiple Raphaël-objects. We could say that every element has a *view*, graphically representing the element. For example, an activity has a rectangle with rounded edges, a shadow and a label for showing the name. All these objects are stored with the `Element`-object. All elements implement three functions for the view:

`createView`: creates all the Raphaël-objects in order to graphically represent the element

`updateView`: update the view based on the properties of the element (e.g. change the location of the element)

`removeView`: remove the Raphaël-objects graphically representing the element

Server-communication

There is one central `server`-object, which takes care of all the communication with the server-side. This includes loading the model when the editor is launched, valuating the model on the server-side and sending changes to be stored on the server-side, as described in Section 5.1.2.

User interface

There is one central `gui`-object. This object has two tasks: creating the left panel which can be used to create new business process elements, as well as interacting with the rest of the web page. For example, at the bottom there are forms for manipulating the properties of the selected element. The `gui`-object is responsible for showing the right form and saving it's contents.

5.3 Server-side

The server-side is the part of the application that runs on the server or maybe several servers. The code is written in the Java programming language [35]. The application should be able to accommodate a lot of users, so it was designed to be vary scalable. The application is designed to be hosted at a *Infrastructure-as-a-Service* platform.

5.3.1 Cloud computing

Cloud computing is a term which is used for delivering computation or storage as a service on demand [37]. A core concept is that the user does not know where this computation or storage physically is (it is somewhere *in the cloud*). The user does not know, and probably does not care where it is.

There are three types of cloud computing services:

Software-as-a-Service (Saas) is a model in which a software tool is delivered as a service. The previously mentioned service Google Drive is

an example of SaaS: Since Google Drive is a web application, the software is provided as a service, not as a product. Also, documents are stored somewhere on-line, but the user does not know where exactly. This doesn't matter, as long as the user can access them using Internet. Since the framework proposed in this thesis is also a web application, it could also be delivered with the SaaS model.

Platform-as-a-Service (Paas) is a model in which developers are provided with a platform which includes all systems and environments which support the complete development live-cycle. Usually services are provided for specific tasks, such as storing data. The advantages of these services are that they can make development easier by dealing with scalability. A downside is that most of these services are not standardised. Using such services makes it difficult to switch providers.

Infrastructure-as-a-Service (Iaas) is a model in which a computer infrastructure is provided as a service. This usually means one can rent virtual machines, possibly for a very short time. Virtual servers provide a great deal of flexibility, but also require maintenance. Another advantage is the usage-based payment scheme, which allows the required computing power and storage to adapt to changes very quickly. Because servers can be added and removed quickly, the application should be designed to handle appearing and disappearing servers.

We have chosen to design the application in such a way that it can be hosted on an Infrastructure-as-a-Service platform. This way we are most flexible in finding servers to host the application.

5.3.2 Data storage

Data storage is an important aspect of the application. The solution must be reliable and scalable. In order for the data storage to be reliable, back-ups need to be made; especially in a cloud computing environment. A lot of users could be using the application simultaneous, so the data storage should be able to handle a lot of traffic.

We have chosen MongoDB [5] as the database software for the application. MongoDB is a open source NoSQL database system. MongoDB comes with bindings for many programming languages.

MongoDB is a document-oriented database. That means that data is not stored in tables, such as in relational databases, but in *JavaScript Object Notation* (JSON) objects. JSON objects are essentially associative arrays which can store multiple data types, including other objects. Objects are stored in

named collections, and can be queried by any property. The advantage is that these objects map very easily to objects in object-oriented programming languages. It is not necessary to use an object-relational mapping framework or implement it yourself.

MongoDB is designed to be reliable and highly scalable. It is possible to have multiple instances, usually on multiple machines, cooperating in order to store a single database. In order to enhance reliability and scalability, MongoDB supports *replication sets* and *automatic sharding* [15].

Replication sets are multiple MongoDB instances which store copies of the same data. The replication sets automatically elects a master node: This node is in charge of writing data to the database. Other nodes can only be used to read data. So, in a replication set, read load is distributed among the nodes. The master server is automatically selected. When the master server fails, another server will be elected as master. It is possible to influence the master selection algorithm by prioritizing the servers. For example, if one wants a real-time backup server at a remote location, a remote node can be added to the replication set with a low priority. If a failed sever comes back on-line, the data is automatically synchronised.

MongoDB also supports automatic sharding (or *partitioning*) of the dataset. With sharding, the dataset is distributed among several instances, while preserving order. So, for example, when we have three machines storing information about persons, the first server could store people with a name which starts with the letters A-J, the second with the letters K-P, and the third the letters Q-Z. When necessary, these boundaries change automatically. Depending on the query, the client will have to connect to only one shard or all of them. MongoDB supports the adding and removing of shards on the fly. Sharding allows horizontal scaling across multiple nodes.

With MongoDB, shards can be stored in replication sets. This way we can combine the reliability of replication sets with the scalability of sharding. Shards, as well as replication instances, can be added and removed on the fly. So when the load or required storage becomes too high, servers can be added without a problem.

But how do the clients know to which MongoDB instance to connect? How does the client know which shards to query? First of all, we need a configuration instance which stores information about the instances. If desired, three configuration instances can be used to overcome failures. MongoDB also comes with a routing process (`mongos`), which behaves as a normal instance. Based on the information provided by the configuration server, the router process automatically selects instances from a replication set or decides which shards to query. This router process could run on any machine, but it is common to run it on the same machine as the application.

5.3.3 Web server

The web server is the application which responds on HTTP(S) requests made by a web browser. There are two types of content that can be delivered. First of all there is static content. This is the content that does not change, such as images, but also some HTML pages and the JavaScript code. Secondly, there is dynamic content. These responses are generated by the Java application and usually require communication with the database. With this application, most dynamic responses are JSON documents, but also some HTML pages are generated dynamically.

For the dynamic responses we chose the Java Servlet API [31]. A Servlet is a Java class which, given a HTTP(S) request, generates a response. Although there are more ways possible to create web applications with Java, the Servlet API is the usual way of doing this.

In order to generate dynamic HTML pages, we made use of *JavaServer Pages* (JSP). JSP allows you to use fragments of Java code inside other text documents, such as HTML documents. This way, small parts of a document can be easily made dynamic. We mainly used this for internationalisation. Labels, descriptions and alerts are loaded dynamically from a language file. This way the application can be easily translated to other languages. There is also another benefit of using a language file: widely used terms can be renamed easily if desired. For example, imagine we want to replace the term *variant* with *sub-process*. We only need to change the language file in order to make the change.

We used the web server software *Apache Tomcat* [9]. Apache Tomcat is a HTTP(S) server and a Servlet container, which means it can forward HTTP(S) requests to a Java Servlet. Besides serving dynamic content which is generated using a servlet, Apache Tomcat can also serve static content. Although Apache Tomcat was chosen, any HTTP(S) web server with a servlet container can be used.

5.3.4 Design

The logic for the server-side has been written in Java. As is usual with Java, the code is divided in packages. The code for server-side can be divided in roughly five categories.

JSP is used to create dynamic HTML pages. The code also needs to generate JSON output. In order to easily generate JSON output, we have used the library JSON.simple [4]. This package is also used to parse JSON documents generated by the client-side.

Web-related classes

There are several classes which take care of the interaction with the web server. This package contains for example the Servlet class, a filter which is used for serving static content and methods for parsing the URL and keeping track of which user is currently logged in.

Data model

The datamodel package contains all classes that need to be saved in the database. For example, all the data of users, business process models and their history is stored in these classes. As explained in Section 4.3.3, business process models are saved as a set of changes. These changes are also part of the datamodel.

Controller

The controllers interpret requests made by the client-side and processes them. This could mean that data is requested, or that new data has to be saved.

Process model

The processmodel classes represent a business process model at a certain point in time. A process can be reconstructed on the serverside based on change classes from the Datamodel. A complete business process model can be exported as JSON document, or can be evaluated using the Computation tree logic package.

Computation tree logic

The computation tree logic package was created by the authors of the PVDI framework [23]. With this package, a business process model can be evaluated according to the PVDI framework. Some minor changes have been made to the package in order to make it work in a web application context.

When the user wants to evaluate a business process model, an Ajax request is created. The server-side then reconstructs the current business process model and evaluates it. The results are send beck to the client in a JSON format. The editor than displays the results to user.

5.3.5 Server layout, scalability and redundancy

Requirement 5 states that the application should be able to be used by many people. The application is essentially a client-server application, in which the

user's browser is the client. In order to handle a high amount of users, the server-side should be able to accommodate a high amount of clients.

As we have seen in Section 5.3.3, a web server like Apache Tomcat is necessary to deliver the content. In Section 5.3.2 we saw that the database software MongoDB is necessary to store all data. For a small amount of users, we can simply run both applications on one server. But when the amount of users increases, we need to divide the workload among several servers. This is also necessary to provide redundancy in order to prevent downtime as a result of maintenance or a crashed server.

When the load increases we could simply run the web server and the database server on separate servers. Although this simple method effectively divides the load among two machines, in does not protect us from downtime.

As we have already seen, HTTP(S) is a stateless protocol. This means that every request is treated independently and is unrelated to any previous request. Although this requires a work around for implementing sessions, it has one big advantage: since every request is independent, we can have multiple web servers which respond to requests individually. An important aspect here is that every web server can reach the database software. Using multiple machines to host the web server also protects us from the consequences of downtime.

But how does the client know to which web server to connect? The different machines hosting the web server all have different addresses. Connecting to the right server can be done using a technique called *load balancing*. Load balancing makes sure that client requests are forwarded to one of the servers, usually using round-robin scheduling. There exist many techniques for load balancing, such as dedicated software, dedicated hardware or using the *Domain Name System* (DNS). Infrastructure-as-a-Service vendors usually offer load balancing as a service. Platform-as-a-Service vendors can even hide the whole load balancing process for the developer.

Load balancers can regularly test the web servers and only forward requests to machines that are responding normally. This way a crashed server or a server in maintenance can be hidden from the client. In the case of a dedicated hardware or software load balancer it is also common to have a *hot spare* for the load balancer: a secondary load balancer which takes over in case the primary load balancer fails.

Load balancing is a very lightweight tasks. A single software or hardware load balancer can handle a lot of traffic. A single load balancer will be sufficient in almost all cases.

Every web server needs to be able to access the database in order to read and write data. As explained in Section 5.3.2, MongoDB can divide the work among multiple machines using replication sets and sharding. By running

the router process `mongos` on every machine hosting a web server, every web server can access all data.

Nodes in a replication set replicate data, thereby introducing redundancy. When a node in the replication set fails the database can still be used. The only consequence is that the other nodes have to process more queries. Read requests are distributed among the nodes in the replication set, but write requests must be coordinated with each node. With sharding the dataset is divided between multiple nodes (or between multiple replication sets). Read requests as well as write requests are distributed among the nodes.

A replication set introduces redundancy, and thereby will prevent downtime as a result of a failed server. When the amount of traffic increases and the redundancy is at an acceptable level, it is more efficient to use shards. Please note that it is possible to run multiple nodes on a single physical machine. For example, the necessary configuration server(s) could be hosted on the same machine which also hosts a node which is a member of the replication set.

Chapter 6

Evaluation

6.1 Performance

In order to get an estimation of the performance of the implementation we benchmarked the server-side of application. We used Apache JMeter [20] to generate HTTP requests and measure the response time. For the test we used Apache Tomcat 7.0.22 and MongoDB 2.0.0. The test was performed on a machine running Ubuntu Linux 11.10 64-bit with a AMD Phenom II X4 945 quad-core CPU and 4GB memory. Both Apache Tomcat and MongoDB run on this server. The HTTP requests were generated on a machine in the same Local Area Network. Both Tomcat and MongoDB are configured to use only one thread and thus can use at most one CPU core each.

We have selected five tests in order to measure the performance:

Static Requesting the *favicon*: a small static file. This should indicate the baseline performance.

JSP Requesting the *overview* page. This HTML code has to be generated using a JSP file, but the database is not needed.

Documents overview Requesting the latest list of documents the user owns or are shared with the user. Data has to be queried from the database and a JSON file is generated. When the *overview* page is loaded this request is performed every 30 seconds.

Process changes Generating two changes to the process and storing them in the database. This happens every time the user makes a change in the process model using the graphical editor. There is a buffer on the client side which ensures this request is not done more than once every three seconds.

Test	Average	Min	Max	Std. Dev.	Throughput
Static	5	1	65	4.20	1759.6
JSP	14	4	87	6.36	667.6
Documents overview	11	4	284	12.19	835.4
Process changes	7	2	99	5.20	1322.2
Evaluate model	51	12	809	74.27	186.3

Table 6.1: Performance of the application. All times are in milliseconds, throughput is in average number of requests per second.

Evaluate model Evaluating the model is a database calculation intensive operation. The model has to be reconstructed in the memory using all independent changes stored in the database before it can be evaluated.

The results of the benchmark can be seen in Table 6.1. As can be expected, requesting a small static time takes not much time. The most important request is posting changes, since this can happen every three seconds when a user uses the graphical editor. Fortunately, storing changes happens relatively quickly: when all users are only posting changes every 3 seconds, our test machine can host almost 4000 simultaneous users. Requesting the overview of documents will also happen quite often, since this is done by the *overview* page every 30 seconds. Valuating models is a request which relatively takes a lot of time. This is however a request which the user has to do manually, so it will probably not be made very often.

6.2 Requirements

In Section 4 we listed the requirements for the framework. In this section we will take a look at whether and how these requirements have been satisfied. Some requirements are satisfied using the proposed framework and some are satisfied in the implementation.

FR-1: *The relation between templates and derived variants should be tracked. When a template is updated, it should be easy to update a variant.*

Using the version control system as proposed in Section 4.3.3, the relation between templates, variants and all versions is tracked. Using the described merge process an existing variant can be updated to a new template.

FR-2: *Changes in the models should be tracked. It should be possible to*

view a previous version of the model.

Using the version control system as proposed in Section 4.3.3, every version of the business process model can be reconstructed.

NFR-1: *Scalability: The solution should be able to accommodate a large amount of users. Thus, the software must perform well and must be scalable.*

As described in Section 5.3.5, all components or the server-side can scale almost horizontally. Web servers can be added or removed, as long as the load balancer is aware of the servers. When using sharding, data can be divided among servers. Shards can be added or removed without interrupting the service. The performance is discussed in Section 6.1.

NFR-2: *Usability: There should be a user-friendly environment for modeling templates and variants. This editor should be attractive and easy to use.*

Although user friendliness is hard to measure, a web application is a convenient solution for the end user. The increased popularity of web applications confirms this. As can be seen in Section 4.2, the user interface of the editor is quite similar to existing editors. This should help users getting used to the editor.

NFR-3: *Reliability: Business process models should be stored safely and always be accessible. A single failed machine should not be the cause of lost data or interrupted services.*

As described in Section 5.3.5, all components of the server-side can be made redundant and can fail without interruptions. When using a load balancer, multiple web servers can be used simultaneously. When a web server fails, the load balancer will not send any requests to that web server. The load balancer must be made redundant, but this is quite common. By using replication sets the database can be made redundant. When replication sets with at least three nodes are used, data can still be red and be written when one node fails [14]. The MongoDB router process will automatically forward requests to another server if one has failed.

6.3 Future work

The main objective of the proposed framework is making modeling business process variability more practical. A good interface and centralised version control can increase productivity. For a fully usable solution, more work is required.

First of all, this framework only focuses on business process modelling. The goal of creating business process models is of course being able to execute them. At this point, this is not possible. Since the PVDI framework is based on a subset of BPMN, it is theoretically possible to export a variant to a BPMN file. In this process the constraint logic would of course be lost, but the result would be a file which can be executed in existing execution engines. The BPMN file can of course be edited using existing tools, but then we would lose the advantages of the proposed framework.

An even better solution would be not to export the model to a file which can be interpreted by a separate execution engine, but to have an integrated solution. Be exporting a process to a file, we lose the relation with process in the centralized version control system. Exporting and importing files with various versions also is a tasks which is unnecessary and error prone. A system in which business process models can be deployed to an execution machine from the web interface would be much more user friendly, clear to the user and less likely to cause errors.

In this framework we have focused on design-time variability. As we have seen in Section 2.4, run-time variability also offers advantages. In order to support run-time variability, the editor as well as the execution engine must be adapted.

Finally, the support for simultaneous collaborating on the same model could be improved. The current implementation does not support multiple users simultaneously working on the same model. Google Drive is a web application which supports collaboration. When multiple users work on the same document at the same time, changes made by one user appear directly on the screen of the other user. It is also possible to create notes for other users and to chat with each other while working on a document. It is possible to implement such tools for business process models as well, but this requires much more effort. For example, when a user makes a change, the change must be submitted directly to other users. This requires the server to push data to the clients.

Chapter 7

Conclusion

Business process variability is an active research topic. As we have seen in Chapter 3, business process variability has the potential of saving a lot of effort, and thus money. Not only in the context of the Local eGovernments, but essentially in every situation where there are multiple variants of the same business process model in use.

The PVDI-framework solves the need for business process variability by combining imperative and declarative frameworks. Templates and derived variants are evaluated using Computational Tree Logic$^+$ formulas. These complex formulas are hidden from the user by using easy to understand graphical symbols. The authors have developed a prototype editor.

What however did not exist was a practical framework for modeling business process variability and keeping track of the relation between template and variants. In the field of Software Engineering, version control systems have proven themselves very useful. The proposed framework also implements a version control system, which makes managing different versions of business process models easy and more clear to the user. Using the proposed framework, losing business process models or making changes that cannot be undone is close to impossible. The merging process allows two sets of changes from a certain state of a business process model to be combined in the best possible way.

The framework was implemented as a web application, which can be hosted on a cloud computing platform. The web application is designed to be user friendly, scalable and reliable. Because the application is implemented as a web application, the application as well as the files can be accessed from anywhere where a modern web browser and an Internet connection is available.

Although additional work is required to create a fully functional framework for modeling and executing business process models, the proposed

framework makes a big step into the direction of successful using business process variability.

Bibliography

[1] Ecmascript website (retrieved august 2012). http://www.ecmascript.org/.

[2] Git website (retrieved juli 2012). http://git-scm.com/.

[3] jquery website (retrieved juli 2012). http://jquery.com/.

[4] Json.simple - a simple java toolkit for json (retrieved juli 2012). http://code.google.com/p/json-simple/.

[5] mongodb website (retrieved juli 2012). http://www.mongodb.org/.

[6] Raphaël javascript library website (retrieved juli 2012). http://raphaeljs.com/.

[7] Software as service for the varying needs of local egovernments (retrieved june 2012). http://sas-leg.net/web/.

[8] M. Aiello, P. Bulanov, and H. Groefsema. Requirements and tools for variability management. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, pages 245–250. IEEE, 2010.

[9] T. A. S. Apache Foundation. Apache tomcat website (retrieved juli 2012). http://tomcat.apache.org/.

[10] F. Bachmann and L. Bass. Managing variability in software architectures. *ACM SIGSOFT Software Engineering Notes*, 26(3):126–132, 2001.

[11] S. Balko, A.H.M. ter Hofstede, A.P. Barros, M. La Rosa, and M.J. Adams. Controlled flexibility and lifecycle management of business processes through extensibility. In *EMISA 2009 : Enterprise Modelling and Information Systems Architectures, Proceedings of the Workshop in Ulm*. GI, 2009.

[12] W. Bandara, M. Indulska, S. Chong, and S. Sadiq. Major issues in business process management: an expert perspective. 2007.

[13] S. Chacon. *Pro Git*. Springer, 2009.

[14] K. Chodorow. *Scaling MongoDB*. Oreilly & Associates Inc, 2011.

[15] K. Chodorow and M. Dirolf. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2010.

[16] Cordys. Business process management suite website (retrieved august 2012). `http://www.cordys.com/bpms-business-process-management-suite`.

[17] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, 2002.

[18] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of computer and system sciences*, 30(1):1–24, 1985.

[19] O. Ezenwoye and S.M. Sadjadi. Robustbpel-2: Transparent autonomization in aggregate web services using dynamic proxies. In *School of Computing and Information Sciences, Florida International University, 11200 SW 8th St., Miami, FL 33199*. Citeseer, 2006.

[20] Apache Foundation. Apache jmeter website. `http://jmeter.apache.org/`, .

[21] The Apache Software Foundation. Apache ode (retrieved june 2012). `http://ode.apache.org/`, .

[22] F. Gottschalk, W.M.P. Van Der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *International Journal of Cooperative Information Systems (IJCIS)*, 17(2), 2008.

[23] H. Groefsema, P. Bulanov, and M. Aiello. Declarative enhancement framework for business processes. *Service-Oriented Computing*, pages 495–504, 2011.

[24] Khronos Group. Webgl - opengl es 2.0 for the web (retrieved juli 2012). `http://www.khronos.org/webgl/`, .

[25] YAWL Group. Yawl: Yet another workflow language (retrieved june 2012). http://www.yawlfoundation.org/, .

[26] B. Hailpern and P. Tarr. Model-driven development: The good, the bad, and the ugly. *IBM systems journal*, 45(3):451–461, 2006.

[27] IBM. Ibm blueworks live website (retrieved august 2012). https://www.blueworkslive.com/.

[28] JBoss. jbpm website (retrieved august 2012). http://www.jboss.org/jbpm/.

[29] M. Koning, C. Sun, M. Sinnema, and P. Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Information and Software Technology*, 51(2):258–269, 2009.

[30] Microsoft. Microsoft visio 2010 website (retrieved august 2012). http://office.microsoft.com/en-us/visio/.

[31] Rajiv Mordani. Jsr 315: Java^{TM}servlet 3.0 specification. http://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-eval-oth-JSpec/servlet-3_0-final-spec.pdf, Dec 2009.

[32] OASIS. Web services business process execution language version 2.0 (retrieved june 2012). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, April 2007.

[33] OMG. Omg model driven architecture website (retrieved august 2012). http://www.omg.org/mda/index.htm.

[34] OMG. Documents associated with business process model and notation (bpmn) version 2.0 (retrieved june 2012). http://www.omg.org/spec/BPMN/2.0/, January 2011.

[35] Oracle. Oracle technology network for java developers. http://www.oracle.com/technetwork/java/index.html.

[36] C. Peltz. Web services orchestration and choreography. *Computer*, 36 (10):46–52, 2003.

[37] B.P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. Ieee, 2009.

[38] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Aalst. Process flexibility: A survey of contemporary approaches. *Advances in Enterprise Engineering I*, pages 16–30, 2008.

[39] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19–25, 2003.

[40] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. *Software Product Lines*, pages 25–27, 2004.

[41] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Modeling dependencies in product families with covamof. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pages 9–pp. IEEE, 2006.

[42] C. Sun and M. Aiello. Towards variable service compositions using vxbpel. *High Confidence Software Reuse in Large Systems*, pages 257–261, 2008.

[43] C. Sun, R. Rossing, M. Sinnema, P. Bulanov, and M. Aiello. Modeling and managing the variability of web service-based systems. *Journal of Systems and Software*, 83(3):502–516, 2010.

[44] Twitter. Bootstrap website (retrieved juli 2012). `http://twitter.github.com/bootstrap/`.

[45] W.M.P. van der Aalst and S. Jablonski. Dealing with workflow change: identification of issues and solutions. *Computer systems science and engineering*, 15(5):267–276, 2000.

[46] W3C. Scalable vector graphics (svg) 1.1 (second edition). `http://www.w3.org/TR/SVG/`, August 2011.

[47] B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *Advanced Information Systems Engineering*, pages 574–588. Springer, 2007.

[48] S. White. Using bpmn to model a bpel process. *BPTrends*, 3(3):1–18, 2005.

[49] S.A. White. Process modeling notations and workflow patterns. *Workflow Handbook*, pages 265–294, 2004.

[50] M. Zur Muehlen, J.V. Nickerson, and K.D. Swenson. Developing web services choreography standardsthe case of rest vs. soap. *Decision Support Systems*, 40(1):9–29, 2005.