Aspect-Oriented Modeling of Business Processes

Masterarbeit
Daniel Colomer Collell
Matrikelnummer: 1827761

September 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik

Examiner: Prof. Dr. M. Mezini

Supervisor: Dr.-Ing. Anis Charfi

Supervisor: Dipl.-Inf. Heiko Witterborg

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, September 2012 Daniel Colomer Collell

Abstract

Concerns such as compliance, auditing, business activity monitoring or accounting need to be addressed in the early stages of modeling and not only at the implementation or execution levels. Mostly, such concerns are modeled as part of the normal flow in business process models. However, the crosscutting nature of such concerns leads to scattered and tangled models. When we try to model business processes that require support for some of these concerns they quickly become complex and cumbersome to understand and manage. The lack of appropriate means to modularize crosscutting concerns in process modeling languages seriously affects understandability, maintainability and reusability.

AO4BPMN 1.0 is an aspect-oriented extension of BPMN that facilitates the modularization of crosscutting concepts in BPMN models such as separation of duties, billing or monitoring. However, there are several open issues and decisions in AO4BPMN 1.0. First, there is no concrete pointcut language defined. Second, no weaving mechanism is provided to compose aspects and processes. Third the language is based on an old BPMN version and the respective editor is based on the STP BPMN editor, which is no longer developed.

In this thesis we refine AO4BPMN into its 2.0 version by defining a concrete OCL-based pointcut language. Thereby we provide extension points to allow integrating other pointcut languages. In addition, we define and implement a weaving mechanism to compose aspects and processes. In addition, we adapt AO4BPMN to the new BPMN 2.0 standard and provide an Eclipse-based editor supporting our extension.

Acknowledgement

This master Thesis was carried out at the Faculty of Informatik at the Technische Universität Darmstadt (TUD).

Special thanks go to my supervisors Dr.-Ing. Anis Charfi and Dipl.-Inf. Heiko Witteborg from SAP Research. Their dedication and guidance were crucial to the Thesis and they always had time to help.

Darmstadt, September 2012.
Daniel Colomer Collell

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

Several concerns cut across different levels of abstraction in a business process. Existing languages used to model such processes, like BPMN, do not support the modeling of such concerns in a modular way. An aspect-oriented extension of BPMN called AO4BPMN was defined in [1] that provides the necessary means to model these so called crosscutting concerns [43]. That extension has several limitations, which will be addressed in this master thesis.

1.1 Motivation

Concerns such as compliance, auditing, business activity monitoring or accounting are bound to be modeled as part of the normal flow when we use languages such as the OMG's Business Process Modeling Language (BPMN) [2]. These concerns need to be addressed in the early stages of modeling and not only at the implementation or execution levels. This concerns are called crosscutting concerns as they affect and apply to different parts of the same entity or system. The first problem that we face when we try to model processes that require some of these concerns is the fact that our diagrams rapidly scale to the point where it gets cumbersome to understand and manage them. Business process models become complex and monolithic and this seriously diminishes their understandability, maintainability and reusability.

Another problem appears when for example an accounting expert tries to understand how accounting is handled in an organization to somehow improve it. The expert must understand how accounting is currently realized across the different business processes and also must understand the whole organization model and how, when and when not audition is necessary for example. This creates big problems of understandability and scalability for the expert in our scenario: the size of the models and their complexity (which we already identified as a first problem) hinder the expert. Another hindering factor is the fact that the modeling elements related to accounting are scattered across all models. If these elements could be encapsulated in a separate module problems as the ones explained above can be addressed better. In order to provide a solution to the problems of crosscutting concern modularity, the authors of [3,4] introduce aspect-oriented workflow languages. These languages provide concepts such as aspect or pointcut that come from the paradigm of aspect-oriented programming [5].

The aim of this thesis is to refine the language design of AO4BPMN based on BPMN 2 and to implement a weaver and an editor for the AO4BPMN based on Eclipse. AO4BOPMN is an aspect-oriented extension of the BPMN language supporting the modularization of crosscutting concerns. This graphical editor should support at least one of the graphical notations of AO4BPMN and the weaver should allow the composition of aspects and processes.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

- Refining the language defined in [1] into AO4BPMN 2.0 by defining a concrete pointcut language, by providing a weaving mechanism and by supporting the new BPMN 2.0 specification.
- Implementation of a graphical editor for AO4BPMN 2 based on Eclipse by extending the BPMN2 editor tool [15] that is based on the BPMN2 meta-model [6] defined in the same project.
- Design and implementation of a weaving mechanism through model transformation using the objective QVT (QVTo) language [7].

1.3 Structure of this thesis

This thesis is structured as follows: Chapter 2 briefly describes the background concepts and technologies needed for understanding this thesis such as the Business Process Modeling Notation, aspect-orientation, model-to-model transformations and the Eclipse project. Chapter 3 describes the problem statement and in particular the problems related to crosscutting concerns in BPMN. In Chapter 4 we give a detailed description of the language AO4BPMN 2.0 including the main constructs of this language the weaving of aspects and processes. In Chapter 5 we present the implementation details related to the editor of AO4BPMN and also to the weaver, which implements the composition of aspects and processes. We present the challenges we faced during the implementation and we discuss also the termination and cost of such an algorithm. In Chapter 6 we present a case study illustrating the usage of the editor and the weaver. Chapter 7 concludes the thesis with a summary of the work done and a discussion of some limitations and directions for future work.

2 Background

In this chapter we will give the needed background knowledge to accompany the rest of the document. First we give an overview on the Business Process Modeling Notation, then we describe the aspect-oriented paradigm, we introduce model to model transformations and we close the chapter with a description of the Eclipse Project and those Eclipse subprojects that are relevant in the context of this work.

2.1 BPMN

The Business Process Model and Notation (BPMN) [2] is a standard defined by the Object Management Group (OMG) as a graphical representation of business process models. It is targeted to business users.

The current version of the standard, BPMN 2.0, is a major step in the BPMN evolution. The aim of BPMN 2.0 is to have one single specification for a notation, meta-model and interchange format. That is why BPMN no longer stands for Business Process Modeling Notation but for Business Process Model and Notation. The Major technical changes in comparison to versions 1 and 1.2 are:

- A formal meta-model
- Interchange formats
- XSLT transformations between the XMI and XSD formats

BPMN 2.0 defines the following layer structure:

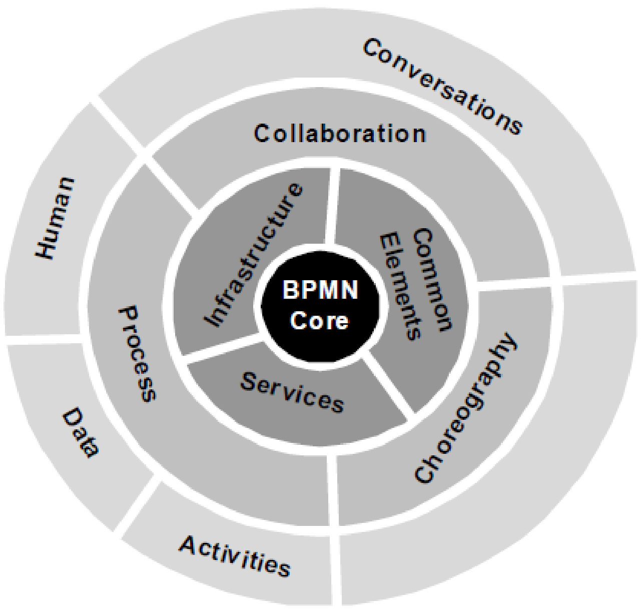


Figure 1 BPMN 2.0 layer structure [2]

The language AO4BPMN 1.0 described in [1] and further introduced in Section 2.2.3 focuses on the common core and process relevant elements, and does not cover the BPMN choreography and collaboration concepts. BPMN core modeling elements are divided in four different categories. The following table summarizes them.

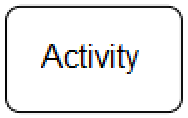
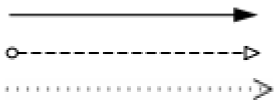
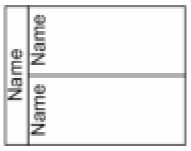
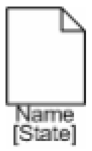
	Description	Constructs	Example
Flow Objects	These are the core elements that a business user needs to model a process	Event, Activity, Gateway	
Connecting Objects	These are the core elements that a business user needs to model a process	Sequence Flow, Message Flow, Association	
Swimlanes	Mainly used to organize Flow Elements into separate visual categories	Pool, Lane	
Artifacts	These elements are designed to help extending the core notation. Their aim is to provide additional context to specific modeling situations	Data Object, Group, Annotation	

Table 1 BPMN core modeling elements (cf. [37])

2.2 Aspect-oriented software development

The aspect-oriented paradigm is a programming paradigm that aims to break down a program in different parts. Almost all programming paradigms offer some level of abstraction/encapsulation that allows a user to build a program based on several more or less independent units. But some concerns defy these forms of abstractions because they “cut across” several of these units. These concerns are therefore known as “crosscutting concerns”. A simple example of a crosscutting concern is the concept of logging. A logging strategy affects all the units of a system that are being logged thereby crosscuts all of them.

The main concepts of the aspect oriented paradigm are:

- An **aspect** can modify the behavior of a normal unit. This construct encapsulates a general crosscutting concern – a concern that applies to different parts of a system. An example of an aspect would be the concept of logging. Logging encapsulates different tasks or behaviors that can be applied along the system, for example the behavior of logging user input and the behavior of logging the system’s output.

- An aspect encapsulates **advice**s (additional behavior). Pieces of advice are behavioral units that are related under the same crosscutting concern. Following with the previously introduced logging example, we could consider logging the input of the user at several points of a system to be an advice.
- A join point is the place where several advices might apply. Following again with the example we could consider the point right after a user fills in a certain form a join point.
- Join points are specified as quantifications or queries called **pointcuts** that detect whether a given join point matches or not. A pointcut therefore defines at which join points one or more pieces of advice should apply. Pointcuts can be independent from the rest of presented concepts. Moreover, a pointcut could be reused by several advices the which apply to the same join points.

2.2.1 Aspect-oriented programming

There exist several language-specific implementations of the aspect-oriented paradigm applied to programming languages. A key differentiating factor of these implementations is the definition of their join point model. Join point models define when an advice-related component can run, that is, what elements can be broken up and interrupted by an advice. A join point model also defines how the join points are quantified. Most of the implementations such as AspectJ (Aspect-oriented java) [20] use the same base syntax. They provide ways to specify the actual code to be executed as an aspect. Join point models are of relevant importance when comparing different implementations of the paradigm (operations permitted in advices, etc.).

AspectJ, for instance, provides the following possible join points: method execution, class initialization, exception handlers, etc. Other implementations such as AspectR (Aspect Oriented Ruby)[21] does notsupport exception handlers and other join points.

As an example the following AspectJ code snippet shown in Listing 1 adds the method *acceptVisitor* to the class *Point*.

```
Aspect DisplayUpdate {
      Void Point.acceptVisitor(Visitor v) {
            v.visit(this);
      }
      //other crosscutting code
}
```

Listing 1 AspectJ code fragment

Another remarkable characteristic is the provided way of combining aspects with the normal code and also when to apply this combinations. An aspect-weaver reads the aspect-oriented code and generates the appropriate code with the aspects integrated. Applying this combination during deployment provides a different approach than for instance applying it during run-time (deploy-time weaving implies post-processing the code for example).

2.2.2 Aspect-oriented modeling

The aspect-oriented paradigm has also been applied to modeling languages (AOM). Model Driven Engineering (MDE) utilizes models as base artifacts in the development process and the application of the aspect-oriented paradigm to this level also has its advantages such as a noticeable increase of reusability of models thanks to the modularization of crosscutting concerns found in them. A comparison of several AOM approaches is realized in [22].

An example of the advantages of AOM can be illustrated by means of the firesensor example presented in [23]. Figure 2 shows how to model the crosscutting concern of sensoring (in concrete FireSensoring).

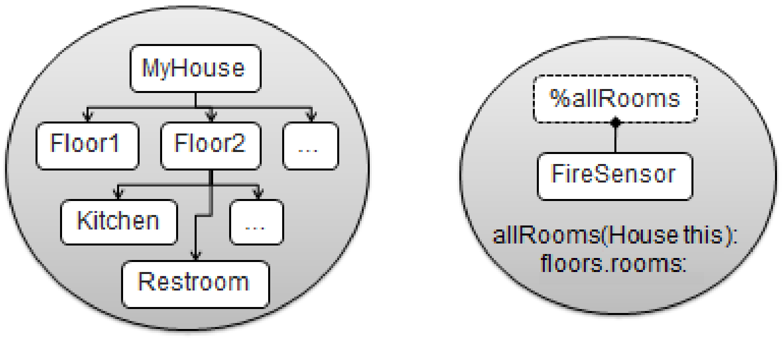


Figure 2 Model of the fire sensor example (cf. [23])

In this example the join points are all the rooms in a house and they are defined by the pointcut *allRooms(House this): floors.rooms* and the applied behavior or advice is the possession of a *FireSensor*.

2.2.3 AO4BPMN 1.0

In this section we offer an overview of the AO4BPMN 1.0 language based on the description given in [1]. AO4BPMN 1.0 aims to be an aspect-oriented extension of BPMN that facilitates the modularization of crosscutting concepts in BPMN models such as separation of duties, billing, monitoring, etc.

In the following, we present the language in more detail starting with the definition of the main constructs, i.e., the aspects containing advices and pointcuts. Based on AO4BPMN's join point model we discuss different possibilities to define pointcuts, that is, to select the join points for a specific advice. Afterwards, we introduce the concepts of aspect and advice as they are defined in AO4BPMN 1.0 and we describe the need of a composition algorithm. Next, we describe two graphical representations: a light-weight syntax that uses only standard BPMN elements thus allowing us to use it with any standard BPMN editor and a heavy-weight syntax that uses new graphical representations. Finally, we list and discuss which are the limitations and open points from the AO4BPMN 1.0 specification that will be addressed in this thesis.

2.2.3.1 Main concepts

In this section we introduce the 4 main concepts defined by AO4BPMN 1.0 as described in [1]. We describe how join points, pointcuts, advices and aspects are defined in AO4BPMN 1.0.

Join points are steps in a process where crosscutting concerns can be integrated. In Section 2.2.1 we introduced the fact that an aspect-oriented language defines the set of available join points by means of a join point model. In AO4BPMN, flow objects are supported as join points. Flow objects, as we introduced in Section 2.1, are core elements within BPMN allowing the user to define events, activities and gateways. AO4BPMN's join point model defines joint points in an implicit and this means that the base processes are standard and can be used still by any BPMN2.0 tool with or without aspect-oriented support and we also gain more flexibility for the modeler of the base process does not need to foresee the potential extensions. An explicit approach would force the modeler of the base process to define the join points manually therefore hampering flexibility and creating non standard and less reusable models. Because of this, there is no special construct for join points for they are implicitly defined by the pointcut that selects them. Examples of join points would be all the activities that require billing.
A pointcut allows us to select join points over which we want to apply a certain crosscutting concern. A pointcut has a query attribute that is used to select the join points.

There exist several approaches on how to select join points and some of them are described in [1]. One possibility is to let the modeler define explicit associations between the pointcuts and the join points but the main drawback of this approach is the lack of scalability. In real-scale processes this option might not be feasible. Another possibility is by means of join point annotation. This approach has a similar problem as the previous one and this is that our models would rapidly scale and would become difficult to manage with normal tools. Nevertheless this approach is more comfortable for business users.A third approach is base on the use of query languages. In this case we must deal with the choice of powerful versus simple languages and what is best for business users from the point of view of complexity and intuitiveness. The authors of [1] list several possibilities such as OCL [9], QVT [7] or ATL [10] query languages but this point is left open in the AO4BPMN 1.0 specification.

An advice is an AO4BPMN construct that implements some crosscutting concerns and may also include a special activity *Proceed*. The semantics of the advice is to apply the defined crosscutting concepts before, after or around the selected join points. These crosscutting concerns are merged along the different processes keeping always previous possible associations between the affected join points and other elements in the model.

The special activity *Proceed* can be used to determine how the crosscutting concerns must be applied. This *Proceed* activity conceptually replaces the selected join point therefore everything implemented before the *Proceed* activity will also be applied before the join point and the same happens with everything implemented after it. If the *Proceed* activity is not used the type of the advice must be defined (i.e. before, after or around).

An advice is self-contained, which means that no sequence or message flows are allowed between advices and other elements in the model.

Aspects are elements used to modularize the modeling of crosscutting concerns. An aspect contains one or more pointcuts and associated advices. The aim of this construct is to group crosscutting behavior under a single concern such as monitoring, compliance or security.

2.2.3.2 Composition of aspects and processes: the weaving

AO4BPMN 1.0 describes the need of an appropriate mechanism to compose the crosscutting concerns defined by aspects and the different business processes. A process transformation is a feasible approach for weaving AO4BPMN aspects with BPMN process models for BPMN is only a modeling notation and cannot be directly executed. As opposite, another approach is presented in [4] that uses an aspect-aware engine.

A model weaver matches the join points by evaluating the different pointcut queries and then the crosscutting concerns defined by the aspects are inserted according to their own definition.

The application of such a composition results in standard BPMN models that can be manipulated by any other standard BPMN tools. It also allows the user to have different views of a process, some of them hiding for example not relevant crosscutting concepts.

The weaving operation is another open point in the AO4BPMN 1.0 specification that will be addressed in this thesis.

2.2.3.3 Graphical representation

In order to use the previously defined constructs of the AO4BPMN 1.0 language, two different graphical notations are defined: a light-weight notation that uses standard BPMN elements and a heavy-weight notation that defines its own graphical syntax.

The light-weight graphical syntax uses standard BPMN elements. Models that use this notation will always be compatible with any other standard BPMN tool. Join points do not have an explicit graphical representation for they are by definition flow elements selected by a pointcut query. Pointcuts are represented as a data objects that have a text annotation containing the text “Pointcut”. The query is stored in the document property of the data object. Advices are represented as subprocesses that have a text annotation containing the text “Advice”. A further optional annotation can be used to indicate the advice’s type. Finally, aspects are represented as a standard BPMN Pools that have a text annotation containing the text “Aspect”.

The heavy-weight notation Deviates from the BPMN 2.0 standard, the heavy-weight graphical syntax introduces new notational elements for the AO4BPMN concepts. The models that use this notation thus will not be displayable with other standard BPMN tools. Join points, as we stated before, do not have explicit graphical representation for they are by definition flow elements selected by a pointcut query. Pointcuts are represented as ovals. In the heavy-weight representation advices are represented as rectangles with two parts. The upper part contains the name and optionally the type of the advice and the other part contains the subprocess activities that implement the desired crosscutting concern. Aspects are represented as pools with rounded corners.

2.2.3.4 Problems and limitations

In the AO4BPMN 1.0 specification there exist several open issues and decisions. These issues will be addressed in this Thesis.

- AO4BPMN 1.0 does not define a concrete pointcut language.
- The concept of a composition of aspects and processes is mentioned but not specified. No concrete algorithm is defines although some suggestions on possible implementation methologies are stated.
- AO4BPMN 1.0 is not based on the BPMN 2.0 specification and the existing editor is based on STPBPMN [24] which is no longer being developed.

In Section 3.3 we describe these problems in more details and Chapter 4 describes AO4BPMN 2.0 which addresses these issues.

2.3 Model Driven Software Development

Model Driven Software Development (MDSD) [40] is a software development approach that understands models as more abstract units than for example the code but at the same time models are consider to be artifacts of the system itself rather than simply documentation. MDSD has the goal to create models that accurately describe the system structure and behavior. These models will then be automatically or semi-automatically transformed into executable code that might or not need some manual refinement.

A key concept in MDSD is meta-modeling [41]. A meta-model is a collection of concepts within a certain domain that represent an abstraction of the abstraction defined by a model. A meta-model expresses the properties and structure of a model itself.

Model transformations [36] are the pillar of MDSD. A model transformation can be understood as a program that takes one or more models as input and produces and output. A model transformation usually specifies which models are acceptable as input and what kind of models it produces as output by means of defining a meta-model to which they must conform. Model transformations have been classified in several ways, for instance depending on the nature of their inputs and outputs or depending on the relation between the meta-models that define the input space and the ones that define the output space. Model transformations can be written in normal generic purpose programming languages but there are also specialized Model transformation languages such as the OMG's standardized model transformation languages that are collected under the name of QVT (Query/View/Transformation) languages.

2.4 The Eclipse project and tools

The Eclipse project was created in 2001 by IBM [25, 39]. It is a development platform that apart from being open it is built of several frameworks which can at the same time be extended by means of plugins.

2.4.1 Plug-ins

A crucial part of the Eclipse environment is the possibility to extend it with plugins [42] that add or modify functionality to the entire platform. The Eclipse it is just an empty carcass and it was built to guarantee and easy and fast plugin development process.

Eclipse allows plugin developers to test their own plugins in a separate Eclipse instance. The packaging and deployment of plugins is almost immediately.

Extension points are the main part of the plugin mechanism defined in Eclipse. Extension points can be used to define a point in a plugin to which another plugin can contribute. The first step to start developing a plugin is to define which extension points must be extended.

2.4.2 The Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) is a framework for Java code generation using models as a base [26]. The framework brings together three different technologies: XMI [27], UML [28] and Java [29]. The main idea behind the framework is that a user can define a model in any of these formats and then automatically transform it to another.

EMF provides Ecore [30], an EMF model that is a metamodel itself. The main aim of this metamodel is to offer the possibility for developers to define their own domain metamodels and to add runtime support that includes a reflective Application Programming Interface (API) to manipulate EMF Objects.

2.4.3 GEF, GMF and Graphiti

The Graphical editing framework (GEF) is a generic Eclipse framework which allows developers to create graphical editors for diagrams in a relatively fast and simple way [31].

The Graphical modeling framework (GMF) is an Eclipse project that is built on top of EMF and GEF to provide a runtime infrastructure that significantly improves the way developers can create graphical editors for diagram by means of defining a tooling model that encapsulates the underline properties of the model elements, a graphical model that defines how the elements are rendered and a mapping model that relates the two previous ones [32].

Graphiti is an alternative solution to the one provided by GMF. Graphiti is an Eclipse-based graphics framework that enables rapid development of diagram editors for domain models. Graphiti can use EMF-based domain models very easily but can deal with any Java-based objects on the domain side as well.

Graphiti hides dependent technologies such as GEF and Draw2D [33]. The user of the framework only needs to deal with Graphiti objects and a single Java API. [16]

2.4.4 BPMN 2 meta-model and modeler

BPMN2 is an open source component of the Model Development Tools (MDT) subproject [14]. BPMN2 Meta-model [6] aims at providing a meta-model implementation of the Business Process Model and Notation (BPMN) 2.0 OMG specification. Relevant goals of the BPMN2 component are to provide an open-source implementation of BPMN 2.0 and a basis for the integration and interchange of artifacts between different tools.

The BPMN2 modeler component is an open-source editor tool created with Graphiti that uses the BPMN2 Meta-model. The BPMN2 modeler tool is still work in progress but a stable version can be already downloaded and used [6].

2.4.5 Model-to-model project

Model-To-Model (M2M) [34] is a subproject of the Eclipse Modeling Project (EMP) [35]. It basically provides three transformations engines for model-to-model transformations. The M2M project currently offers three different transformation engines and a corresponding development environment:

- **Declarative QVT** is based on the OMG's QVT core and relations languages [7], but it is still work in progress.
- **Operational QVT** is based on the OMG's QVT operational package mappings and provides a procedural language. This package is used in the implementation of this thesis and it is explained in more detail in Section 5.3.2.
- **The ATL Transformation Language (ATL)** is a transformation language [10] with a complete IDE and debugger built on top of the Eclipse platform.

3 Problem statement

As we discussed in Section 2.2 there are some concerns that defy the typical forms of modularization because they “cut across” several dimensions. These concerns are known as “crosscutting concerns”. A simple example of a crosscutting concern in a software system is logging.

The concept of “crosscutting concern” is not restricted to programming and it can be also extrapolated to the domain of business processes. For example business activity monitoring can also be considered a crosscutting concern. Activities realizing this concern will be scattered across all the processes that need to be monitored. With state of the art process modeling language there is no module that encapsulates all functionality related to business activity monitoring.

In this chapter we explain the problems related to crosscutting concerns in the domain of business process modeling with the help of an example. Then, we discuss several open issues in AO4BPMN 1.0 [1], which introduces aspect-oriented concepts to BPMN.

3.1 Motivating example

To better motivate things, we introduce in this section a simplified version of two business processes in a web application development company. This example will be used all along this thesis for illustration purposes.

In our example we will assume that the company has its developers structured in three teams. Among all the processes of the company we will focus on two specific business processes: feature development and bug fixing. Team 1 and Team 2 are responsible for feature development while another team focuses only on bug fixing. The company uses an A/B testing methodology [8]. The aim of this methodology applied to web application development is to guarantee the most satisfactory implementation of a feature by means of developing two alternatives and comparing the reaction of the users afterwards. In Figure 3 we can see the feature development process. After a feature request is received, Team 1 and Team 2 start working on alternative implementations and afterwards their implementations are evaluated in the sub-process activity *evaluate A/B* and finally delivered. The bug fixing process follows a simpler structure. The team responsible for bug fixing solves a bug request by means of the sub-process activity *fix bug* and afterwards the patch is commited to the company intermediate servers by means of the activity *commit patch* and the process finishes with the *patch delivery* to the client.

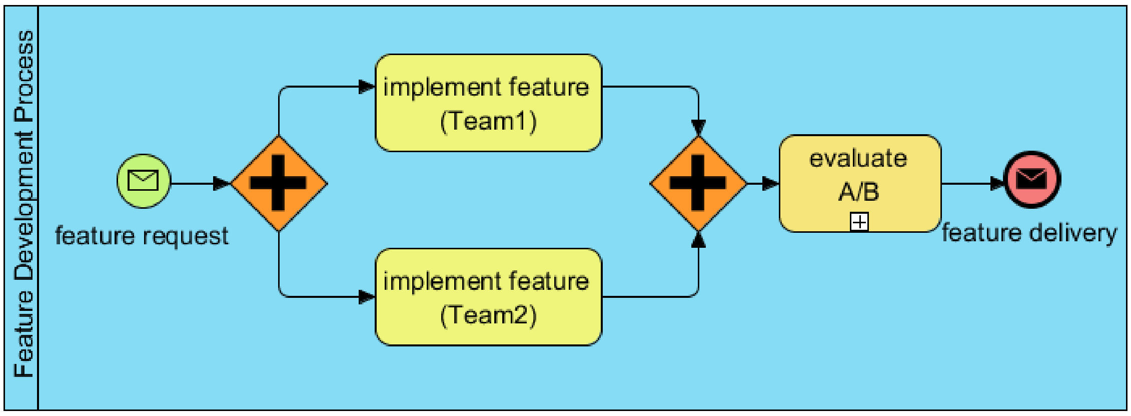

Feature Development Process
feature request
implement feature (Team1)
implement feature (Team2)
evaluate A/B
feature delivery

Figure 3 Feature development process diagram

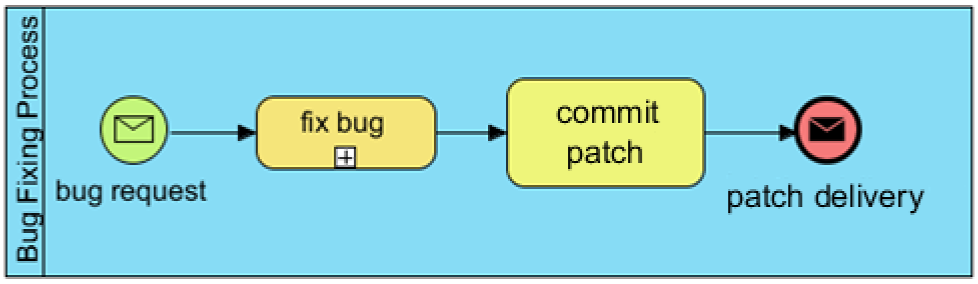


Figure 4 Bug fixing process diagram

In order to improve performance and control the process of A/B testing implemented in the feature development process the company is interested in monitoring the time each team takes to implement their alternative feature. The company also wants to guarantee that the developers follow the internal coding guidelines. To achieve these two goals the company applies *start* and *stop timer* tasks around the implementation feature tasks and before the feature delivery and solution delivery the company applies a *check guidelines* task. Figures 5 and 6 show the modified business processes in BPMN.

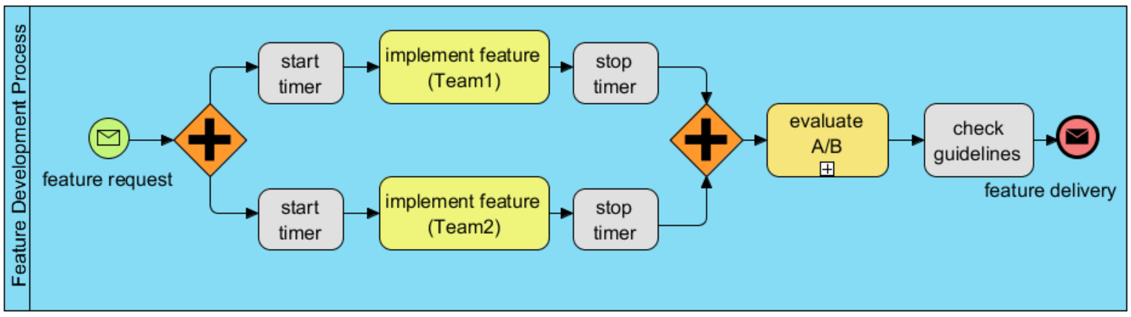


Figure 5 Complete feature gevelopment process diagram

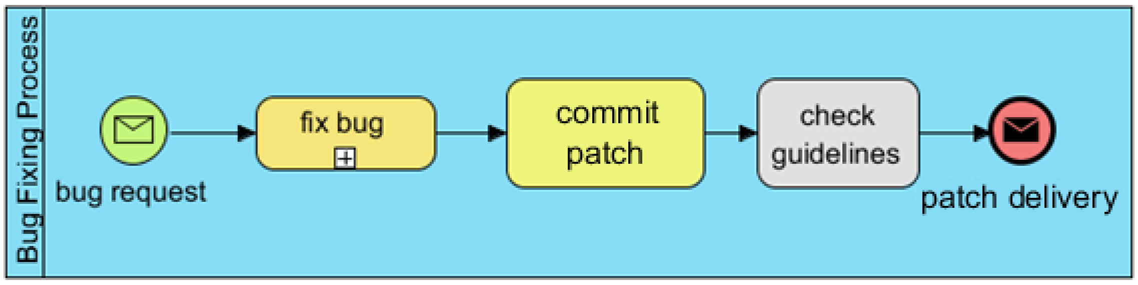


Figure 6 Complete bug fixing process diagram

Figures 5 and 6 show that the complexity of the process models increase considerably after the integration of the concerns monitoring and compliance. The lack of a way to properly modularize these concerns leads to several problems. The process morels get more complex, less maintainable and less reusable. A part from these disadvantages the concepts of monitoring and compliance are not much related to the core functionality of the processes of feature development and bug fixing in our example therefore understandability is also affected.

3.2 Crosscutting concerns in BPMN

The example presented in the previous section allowed us to see several problems due to the crosscutting nature of the concerns monitoring and compliance. The first two problems that we observe when crosscutting concerns are modeled in the business process are scattering (activities realizing business activity monitoring are spread in two processes) and tangling (activities realizing different concerns are mixed).

If we compare the number of elements shown in Figure 3 and those in Figure 5 we see how this number increased considerably after introducing the activities related to the crosscutting concerns monitoring and compliance. This scattering makes the models difficult to manage. Every time we would like to add a new feature implementation team to implement for example an A/B testing approach that would consider more than two alternatives our process would get much more complex. For example, if we want to add an *implement feature* task for a new team, two more monitoring tasks would have to be added to the process.

Another problem is that the more the models grow, the less separation of concerns we have. The crosscutting concerns are scattered through all the processes and this blurs the main purpose of them. In the case of the feature development process we can distinguish that the tasks related to implementation and A/B testing are the ones semantically significant for the purpose of the process and the concepts of monitoring and compliance do not do anything else than tangling our model.

From these problems mentioned above several non-functional properties are hampered. The more our processes grow, the less understandable they become. Due to the tangling our processes become less understandable by experts who would like, for instance to improve them. In this same direction maintainability is also affected. If the company wants to review the monitoring approach that is already implemented, it will be a difficult task to search through all the process models to understand where monitoring was applied and where it has to be applied later. The reuse of the base processes (that is the processes without the crosscutting concerns) also becomes a cumbersome task because the concerns of monitoring and compliance are not properly modularized.

3.3 Limitations of AO4BPMN 1.0

As we described in Section 2.2.3, the language AO4BPMN 1.0 is introduced in [1]. Some concepts in that language are still open and need to be defined. In this section we will describe in more details the open issues in AO4BPMN 1.0 and we will explain how these problems are approached in this thesis.

The authors of [1] presented the main concepts of AO4BPMN 1.0 including the join point model, pointcut language, and advice language. In addition, the authors proposed two different graphical notations. However, in [1] the authors only discussed several alternatives for the pointcut language such as OCL [9], QVT [7] or ATL [10]. They did not go further by presenting a concrete pointcut language for AO4BPMN. In this Thesis we address this problem by providing an OCL-based pointcut language and also giving the possibility for other developers to add support for their own pointcut languages. More details on this topic are given in Section 4.2 from a conceptual point of view and in Section 5.2.4 from an implementation point of view.

The authors also identified in [1] the need for a weaving mechanism to compose aspects and processes. However, there was no concrete mechanism for that. In this thesis we address this problem by defining a weaving algorithm and providing an implementation of such an algorithm following the suggestions stated in [1] for using a model to model transformation to produce the output models of such a composition. More details on the design and implementation of the weaver are given in Section 4.6 and Section 5.2.

Furthermore, AO4BPMN 1.0 was not based on the BPMN 2.0 specification and the existing editor is based on STPBPMN [24], which is no longer developed. In this thesis we address this third problem by implementing an editor for AO4BPMN 2.0, which is an extension of an existing Eclipse based and BPMN 2.0 compliant editor. More technical-related details on the editor implementation are given in Chapter 5.

4 AO4BPMN 2.0: language overview

In this chapter we offer an overview of the AO4BPMN 2.0 language. AO4BPMN 2.0 aims to be an aspect-oriented extension of BPMN that facilitates the modularization of crosscutting concepts in BPMN models such as separation of duties, billing, monitoring, etc. AO4BPMN 2.0 also aims to cover the open issues left in the specification of AO4BPMN 1.0 presented in Section 2.2.3.

In the following sections we present the language in more detail starting for the definition of the main constructs and a description of the different graphical representations that the language offers: A light-weight and a heavy-weight approach. We close this chapter explaining the theoretical details behind the design of the weaving algorithm.

4.1 Join points

As we already introduced in Section 2.2.3.1 join points are steps in a process where crosscutting concerns can be integrated. As introduced in Section 2.2.1 an aspect-oriented language defines the set of available join points by means of a join point model. In AO4BPMN, flow objects are supported as join points. Flow objects, as we introduced in Section 2.1, are the main describing elements within BPMN allowing the user to define events, activities and gateways. AO4BPMN's join point model defines joint points in an implicit way; this means that the base processes are standard BPMN constructs and can be used still by any BPMN 2.0 tool with or without aspect-oriented support. We also gain more flexibility for the modeler of the base process does not need to foresee the potential extensions. An explicit approach would force the modeler of the base process to define the join points manually therefore hampering flexibility and creating non standard and less reusable models. Because of this, there is no special construct for join points for they are implicitly defined by the pointcut that selects them. Examples of join points would be all the activities that require billing.

Figures 7 and 8 show the processes of our web application development company example described in Section 3. The underlined elements are join point candidates according to the join point model defined by AO4BPMN.

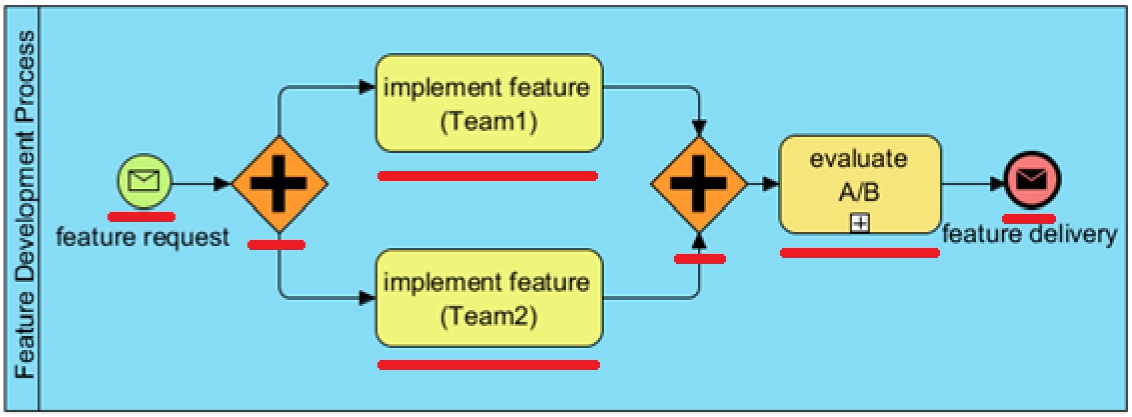

Feature Development Process
feature request
implement feature (Team1)
implement feature (Team2)
evaluate A/B
feature delivery

Figure 7 Feature development process with candiadate join points underlined

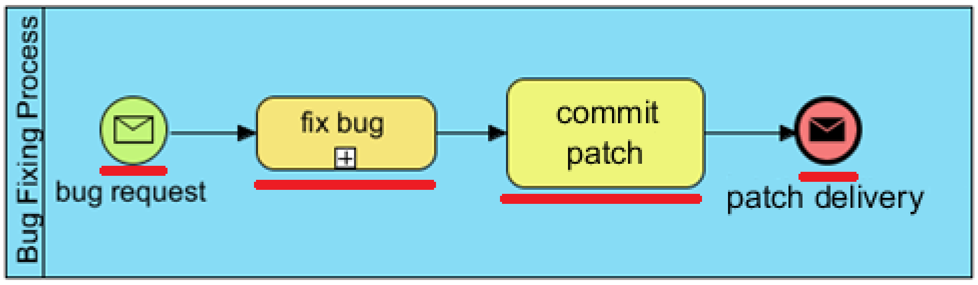


Figure 8 Bug fixing process with candidate join points underlined

4.2 Pointcuts

As we previously introduced in Section 2.2.3.1 a point cut allows us to select join points over which we want to apply a certain crosscutting concern. A pointcut has a query attribute that is used to select the join points.

There exist several approaches on how to select join points and some of them are described in [1]. One possibility is to let the modeler define explicit associations between the pointcuts and the join points. The main drawback of this approach is the lack of scalability. In real-scale processes this option might not be feasible.

Another possibility is by means of join point annotation. This approach has a similar problem as the previous one: our models would rapidly scale and would become difficult to manage with normal tools. Nevertheless this approach is more comfortable for business users.

A third approach is based on the use of query languages. In this case we must deal with the choice of powerful versus simple languages and what is best for business users from the point of view of complexity and intuitiveness. Possible candidate languages could be OCL [9] or more concrete implementations of Model Query languages such us EMF ModelQuery [17] or EMF IncQuery [18]. The problem with these implementations is that we cannot assume the business user has knowledge about them or is willing to learn it. Nevertheless EMF ModelQuery2 offers the possibility to execute queries in a SQL-like syntax. Listing 2 exemplarily shows the syntax needed to formulate a query that selects the Manuscripts with the title "BPMN tutorial" using EMF ModelQuery meanwhile Listing 3 shows exactly the same with EMF ModelQuery2 [19].

```
new SELECT(
      new FROM(selectedObjects),
      new WHERE(new EobjectAttributeValueCondition(
            EXTLibraryPackage.eInstance().getTitle(),
            new StringCondition.stringValue(„BPMN Tutorial‟)
      )
)
```

Listing 2 EMF ModelQuery code fragment that selects the manuscripts with the title „BPMN Tutorial"

```
From Manuscript as m
      Select m.title
      Where m.title = „BPMN Tutorial‟
```

Listing 3 EMF ModelQuery 2 code fragment that selects the manuscripts with the title „BPMN Tutorial"

In the first case we observe a certain degree of complexity related to the syntax and probably a business user would not be familiar with such a language even if it has a syntax close to SQL. The query shown in Listing 3 is closer to what a business user might be able to use. The use of selection constructs defined in languages such as QVT [7] or ATL [10] is closely related to the use of OCL [9] for these languages offer their own extended version of OCL.

In our implementation of AO4BPMN, a query-based pointcut language is used. In Chapter 5 we discuss how we overcame this problem of choosing a language to give support to by offering a built-in OCL implementation and allowing at the same time for expert users to extend our implementation and define their own query-based pointcut Languages. As part of future work in this direction we could also offer a wizard-based query creation process that would allow non expert users to define queries without writing actual code.

4.3 Advices

As introduced in Section 2.2.3.1 an advice is an AO4BPMN construct that implements some crosscutting concerns and may also include a special activity *Proceed*. The semantics of the advice is to apply the defined crosscutting concepts before, after or around the selected join points. These crosscutting concerns are merged along the different processes keeping always previous possible associations between the affected join points and other elements in the model.

The special activity *Proceed* can be used to determine how the crosscutting concerns must be applied. This *Proceed* activity conceptually replaces the selected join point therefore everything implemented before the *Proceed* activity will also be applied before the join point and the same happens with everything implemented after it. If the *Proceed* activity is not used the type of the advice must be defined (i.e. before, after or around).

An advice is self-contained, which means that no sequence or message flows are allowed between advices and other elements in the model.

4.4 Aspects

Aspects, as described in Section 2.2.3.1, are elements used to modularize the modeling of crosscutting concerns. An aspect contains one or more pointcuts and associated advices. The aim of this construct is to group crosscutting behavior under a single concern such as monitoring, compliance or security.

4.5 Graphical Representations

In order to model using the previously defined constructs of the AO4BPMN 2.0 language we offer the same notations introduced in Section 2.2.3.3 and first defined in [1]. The two different graphical notations are: a light-weight notation that uses standard BPMN elements and a heavy-weight notation that defines its own graphical syntax. Figures 9 and 10 show an example of the respective graphical notation using the motivation example introduced in Chapter 3.

4.5.1 Light-weight notation

The light-weight graphical syntax uses standard BPMN elements. Models that use this notation will always be compatible with any other standard BPMN tool. Figure 9 shows an example of this lightweight notation.

- **Join point:** They do not have explicit graphical representation for join points are by definition flow elements selected by a point cut query.
- **Pointcut:** This construct is represented as a data object that has a text annotation containing the text “Pointcut”. The query is stored in the document property of the data object.
- **Advice:** This construct is represented as a sub-process that has a text annotation containing the text “Advice”. A further optional annotation can be used to indicate the advice’s type.

- **Aspect:** This construct is represented as a standard BPMN Pool that has a text annotation containing the text “Aspect”.

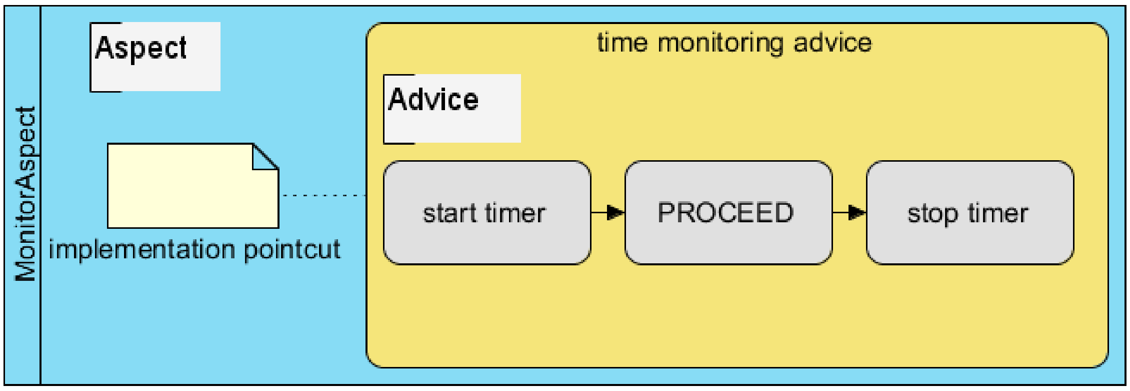


Figure 9 Light-weight notation monitor aspect diagram

4.5.2 Heavy-weight notation

Deviating from the BPMN2.0 standard, the heavy-weight graphical syntax introduces new notational elements for the AO4BPMN concepts. The models that use this notation thus will not be displayable with other standard BPMN tools.

- **Join Point:** As we stated before these elements do not have explicit graphical representation for join points are by definition flow elements selected by a pointcut query.
- **Pointcut:** This construct is represented as an oval.
- **Advice:** This construct is represented as a rectangle with two parts. The upper part contains the name and optionally the type of the advice and the other part contains the sub-process activities that implement the desired crosscutting concern.
- **Aspect:** This construct is represented as a pool with rounded corners.

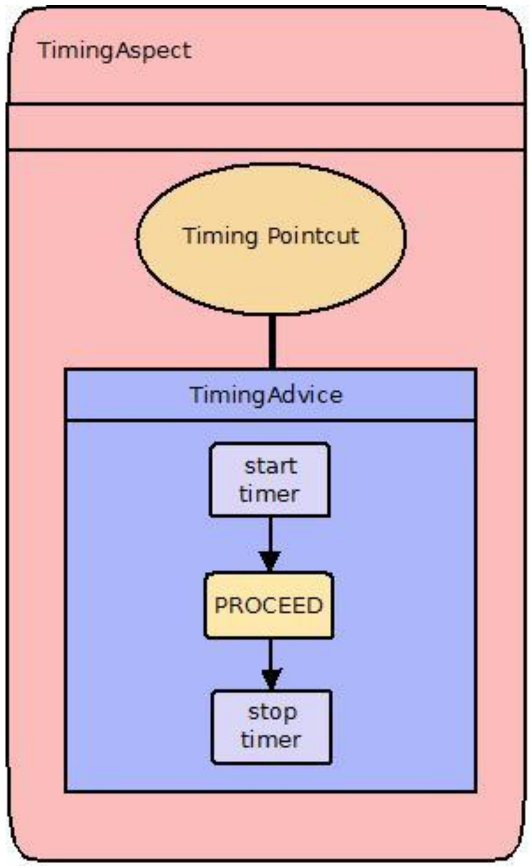


Figure 10 Heavy-weight notation timing aspect diagram

4.6 Composition of aspects and processes: the weaving

AO4BPMN needs an appropriate mechanism to compose the crosscutting concerns defined by aspects and the different business processes. A process transformation is a feasible approach for weaving AO4BPMN aspects with BPMN process models for BPMN is only a modeling notation and cannot be directly executed. As opposite, another approach is presented in [4] that uses an aspect-aware engine.

A model weaver matches the join points by evaluating the different pointcut queries and then the crosscutting concerns defined by the aspects are inserted according to their own definition.

The application of such a composition results in standard BPMN models that can be manipulated by any other standard BPMN tools. It also allows the user to have different views of a process, some of them hiding for example not relevant crosscutting concepts.

General graph transformation techniques [38] can help us understand the problem. This composition method in particular can be implemented using model transformations. In Chapter 5 we describe the details of the implementation of such an algorithm.

In this section we describe an example of such a composition and the main conceptual issues related to the design of such an algorithm. Section 4.6.2 describes how the algorithm must deal with the process structure. Section 4.6.3 defines special branching cases that need to be considered. Section 4.6.4 closes the section with comments about the need of a re-linking strategy.

4.6.1 A weaving example

Taking part of the example about the web application development company Figures from 11 to 13 illustrate the different states involved in the composition method. Figure 11 shows the base process of bug fixing; Figure 12 shows the aspect that encapsulates the crosscutting concern of guidelines compliance and Figure 13 shows the result of composing the previous models. In this example we use the light-weight notation introduced in Section 4.5.1.

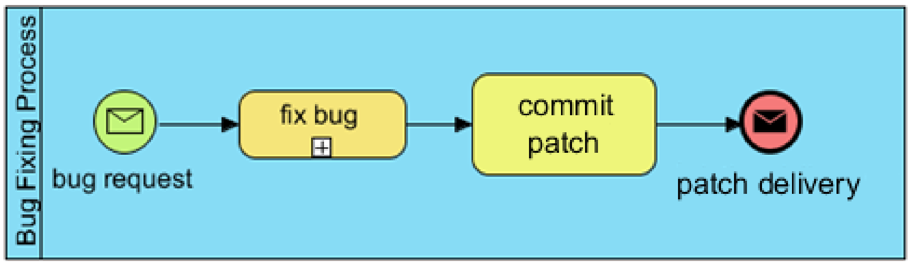


Figure 11 Bug fixing process

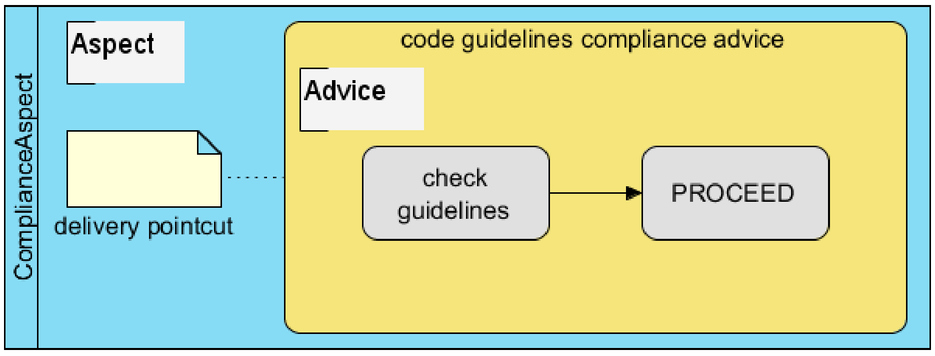


Figure 12 Light-weight notation compliance aspect

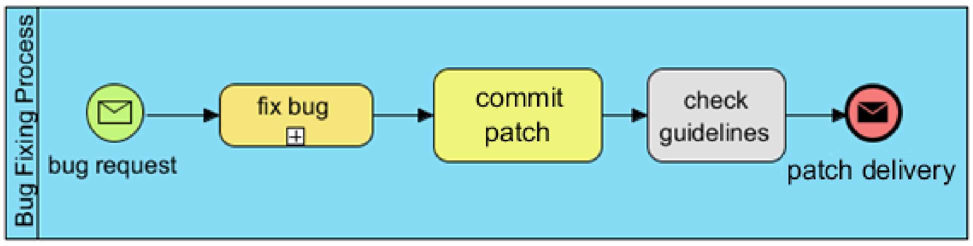


Figure 13 Bug fixing process after the weaving operation

4.6.2 Clone and merge

The weaving defined in [1] can be reduced to a clone and merge algorithm. The main idea behind the algorithm is that the structure defined in the different aspects has to be cloned and merged with the selected Join points.

The cloning algorithm is responsible of cloning the needed information from the aspect-oriented model to the process model. Given a join point and a *Proceed* element as input our knowledge is limited to whatever is directly attached to these elements. Taking advantage of this situation we will design an algorithm that clones by level.

We define level as a set of all the elements that are at the same distance of the proceed element and we define distance between two elements as the number of associations in the shortest path that links the two elements.

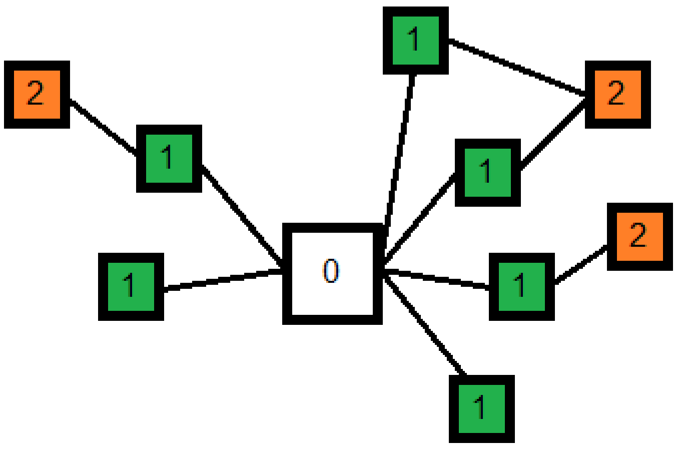


Figure 14 Abstract structure with the elements and their distance to the central element

Figure 14 represents an abstract structure with elements and their distance to the central element (element with distance 0). Our algorithm will clone first all the elements at distance 1, then at distance 2, and so on and so forth until the last level is reached.

4.6.3 Special branching cases

a In a BPMN process diagram the different flow objects and other elements such as data objects are related by means of different kinds of connecting elements. As we introduced in Section 2.1 these are sequence flow, message flow and association. Because of these three different constructs, we find in a process diagram three different flow levels:

- Everything that is linked by means of a sequence flow connection belongs semantically to the process itself as part of its sequence of steps. It is essential then that the algorithm supports this flow level for it is the level that contains the semantic part of the process.

- Everything related by means of an association connection is used to express relevant information that is found along the steps of the process but it does not belong to the normal flow. This information must also be considered by the algorithm for it contains relevant objects that contribute to the semantics of the process.
- Message flows are used to express communication between processes or organizations. The definition of aspect as it is stated in [1] does not allow the use of message flows for an advice is a self-contained unit that cannot receive any kind of external communication. That is why message flows should be ignored by the algorithm.

Because of these three flow levels several branching issues arise at the implementation level. Section 5.2 describes these issues and the applied solution as well as other technical issues that are found in the implementation of such an algorithm. In concrete, Sections 5.2.7 and 5.2.8 deal with branching-related problems.

4.6.4 Re-linking strategy

As we stated in Section 4.6.2 this algorithm is by nature a clone & merge algorithm. The merging part of the algorithm also presents some problems that need to be solved. It is important not to lose whatever was connected to the join point before executing the weaving therefore these elements need to be re-linked to or from the newly cloned elements.

First of all after the cloning is performed we need to distinguish what elements are candidates to be targets or sources of the so called re-linking phase. One of the main characteristics that we observe is that these candidates must have either only incoming or outgoing edges. This condition is still not strong enough as we will see later on this Section there some special branching cases to take into account.

A problem appears when observing the most generic case. Taking into account only tasks and sequence flows our target process model might look like the model depicted in Figure 15.

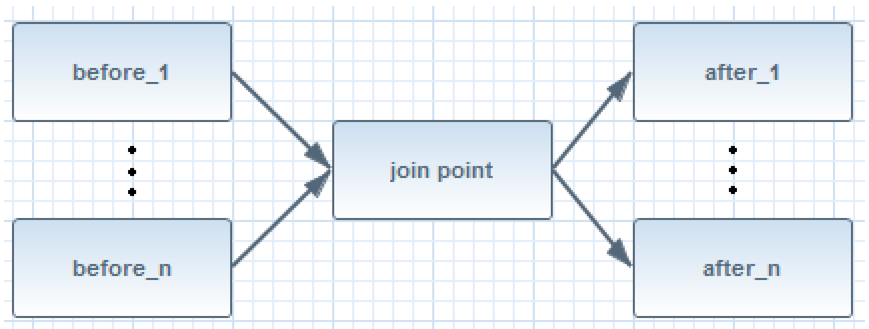


Figure 15 Generic join point structure with n incoming and outgoing connections

In the same conditions our aspect-oriented model might look like the model depicted in Figure 16.

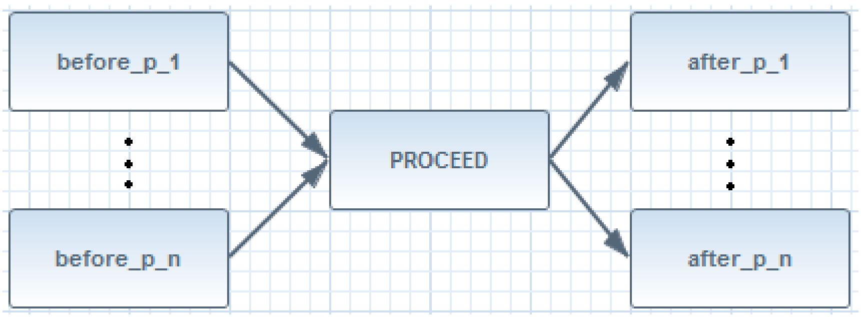


Figure 16 Generic *Proceed* structure with *n* incoming and outgoing connections

We need to define a re-linking strategy that merges the cloning structure with the original keeping the semantics and trying not to overload the model with edges. We cannot afford on generating $2n^2$ edges, therefore, unless *n* equals 1 we must define a clearer way to merge these models.

According to the BPMN 2.0 specification the following structures depicted in Figure 17 are equivalent as well as in Figure 18.

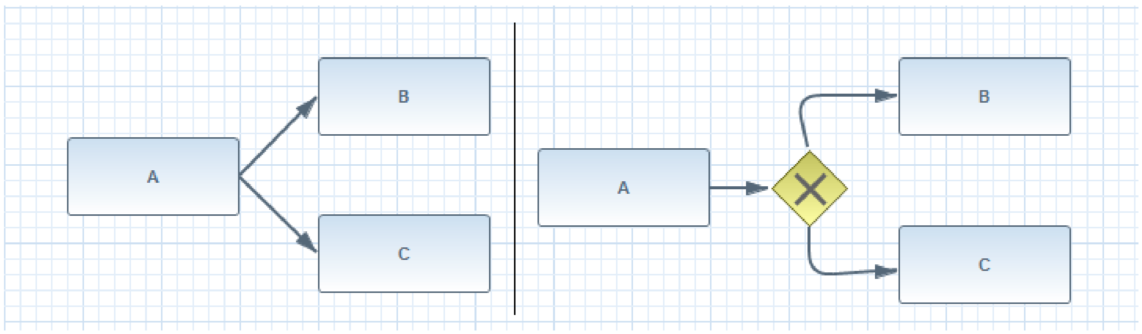


Figure 17 Equivalence between splitting structures according to the BPMN 2.0 specification

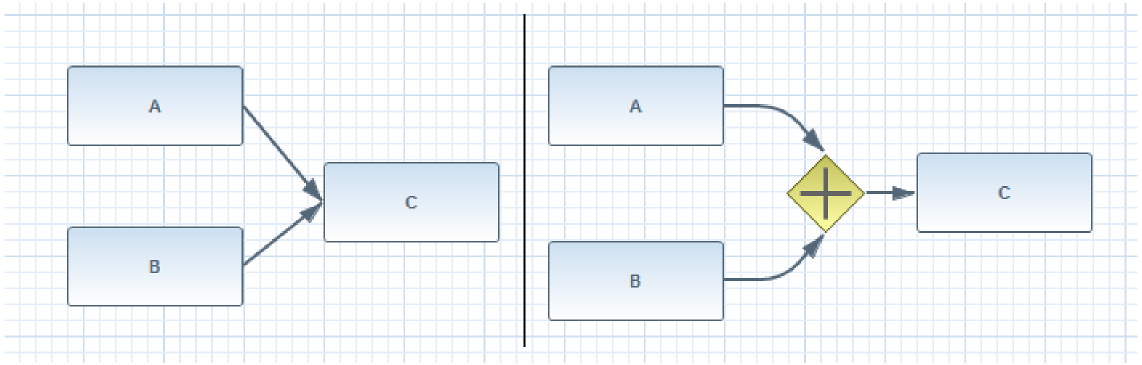


Figure 18 Equivalence between merging structures according to the BPMN 2.0 specification

Taking into account the equivalence of these structures we define the following generic re-linking strategy. All the cloned elements that are candidates for this re-linking will be connected to the originally steaming out elements through the use of a parallel gateway connected to an exclusive gateway as depicted in Figure 19.

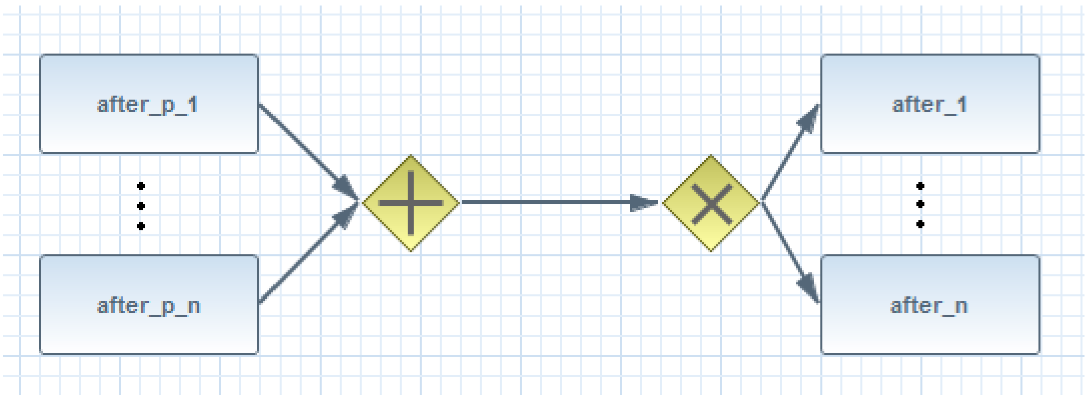


Figure 19 Generic case of our re-linking strategy

In the case of n = 1 we can avoid the creation of the intermediate gateways and simplify the re-linking using a direct connection via a sequence flow link. This structure is generated in both sites of the selected join point. We need to generate it for both the incoming and the outgoing elements.

5 Implementation and tooling

In this section we document the implementation and the functionality of the AO4BPMN editor tool and the details of the weaving algorithm. This section realizes the concepts introduced in Chapter 4.

5.1 Editor tool

The editor tool described in this section is the part of the implementation that realizes de modeling concepts presented in Sections from 4.1 to 4.4. The main goal of the editor tool is to enable the creation and modeling of BPMN diagrams using the aspect-oriented constructs described in Chapter 4.

Section 5.1.1 describes the implementation technology used to implement the editor tool. The sections from 5.1.2 to 5.1.6 describe the realized extensions as the key points of the implementation.

5.1.1 Implementation technology

There exist already several running implementations of editors focused on the BPMN modeling language. The main purpose of the thesis was the AO4BPMN language. That is why we decided to extend an existing editor rather than implementing one from scratch. This decision was partly taken to avoid the complexity of implementing a full editor. Regarding the selection of an adequate editor the following requirements were established:

- The Editor shall be compliant to the BPMN 2.0 specification.
- The Editor shall provide clear mechanisms for its extension.

Eclipse and the BPMN2 editor that comes with the Eclipse Model Development Tools (MDT) project [14] fits these requirements and its components are open source which adds value to the decision.

The BPMN2 Modeler [15] is a graphical modeling tool which allows creation and editing of BPMN diagrams. The tool is built on Eclipse Graphiti [16] and uses the BPMN 2.0 EMF meta-model [6] currently being developed within the Eclipse Model Development Tools (MDT) project [14]. This meta-model is compatible with the BPMN 2.0 specification [2] proposed by the Object Management Group.

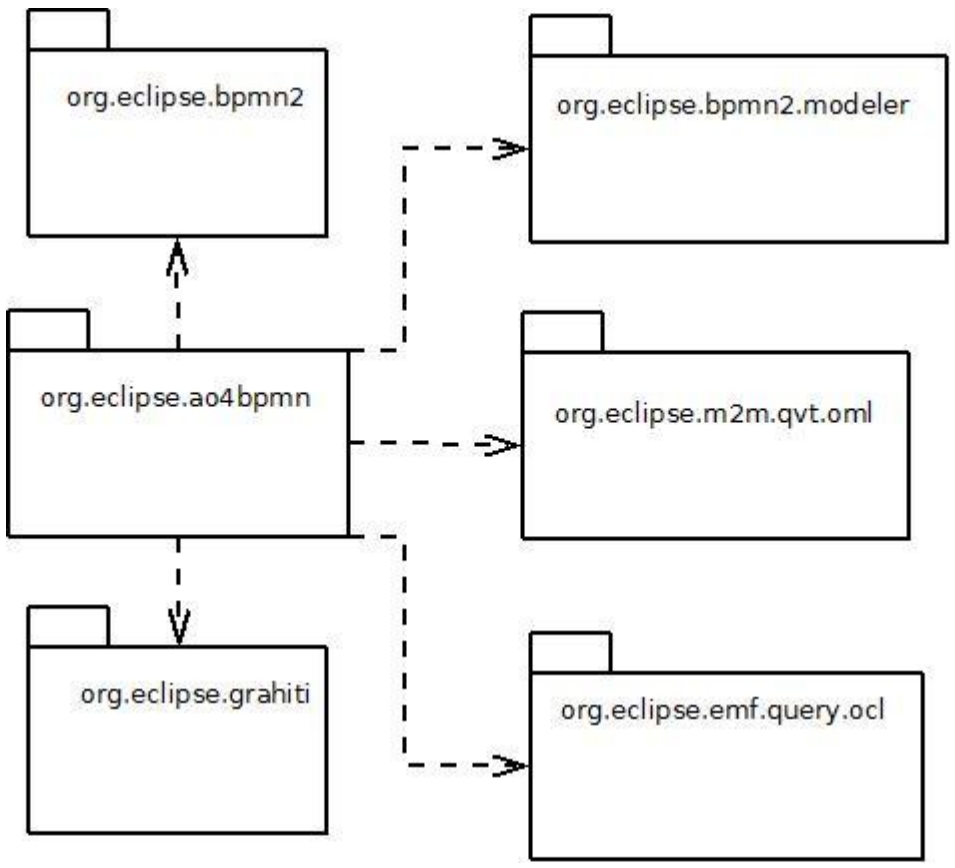


Figure 20 Plug-in structure in packages

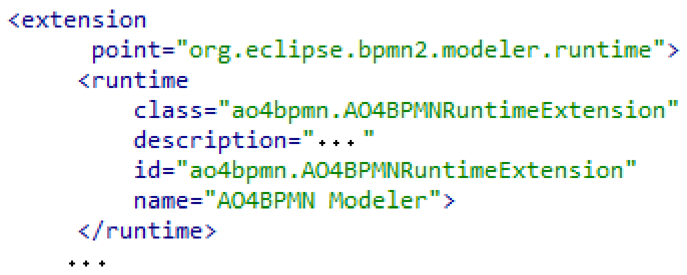
Figure 20 shows what are the main dependencies of the implemented project ao4bpmn. The ao4bpmn implementation aims at extending the BPMN2 modeler therefore it depends on both the BPMN2 meta-model and modeler packages. As we descrive later in Section 5.2.1, the QVTo Language [7] is used to implement theweaving algorithm and we also offer built-in support for OCL [9] as a point cut query language. The ao4bpmn package also makes use of some Graphitti [16] concepts and that is why a dependency appears in the figure. The BPMN2 modeler tool defines a single extension point that allows us to define our own runtime variation of the tool. This extension point has the following identifier: *org.eclipse.bpmn2.modeler.runtime*

The extension point offers the following possibilities:

- To extend the property tabs of any modeling element defined in the BPMN2 meta-model (the actual eclipse implementation).
- To define your own custom tasks.
- To define your own model extension. That is, to extend the meta-model with your own properties.
- To enable/disable the availability of one or more modeling elements.
- To provide your own feature containers for available modeling element. That is, to override part of the behavior such as the rendering or linking restrictions of user tasks for instance.
- To define your own style sheet. That is, to define background colors and styles for the tool.

5.1.2 AO4BPMN runtime extension

The first step of our implementation is to define the root of our extension. We need to define our own runtime rxtension to which the rest of functionality will be linked. Figure 21 shows the XML structure of the runtime extension point. It is used to define the name and description of our extension as well as to point to the plugin class.



```
<extension
      point="org.eclipse.bpmn2.modeler.runtime">
     <runtime
         class="ao4bpmn.AO4BPMNRuntimeExtension"
         description="... "
         id="ao4bpmn.AO4BPMNRuntimeExtension"
         name="AO4BPMN Modeler">
     </runtime>
    ...
```

Figure 21 XML fragment of the runtime extension point

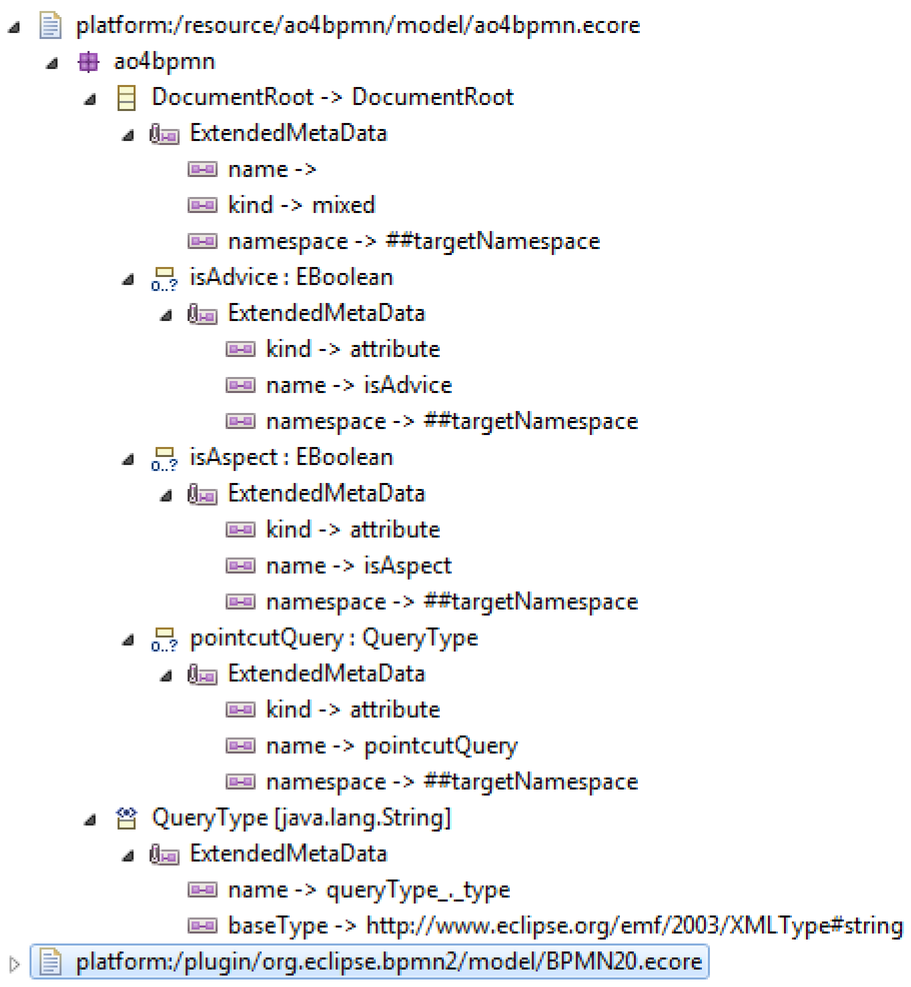
The runtime extension point does not allow us to define our own modeling elements therefore the possibility to extend the meta-model and define the different aspect-oriented elements that we need is not feasible. In [1], the different aspect-oriented modeling elements are described and mapped to actual BPMN modeling elements. Taking this into account we decided to implement the extension using the following approach:

- **Aspect:** We extend the pool element (participant class in the tool meta-model) with a boolean property named *isAspect* that will allow the user to use this element as a normal pool or as an aspect.
- **Advice:** We use the same technique described above. A property named *isAdvice* is defined for the sub-process that will allow the user to use this element normally or as an advice.
- **Pointcut:** In this case we considered two possibilities. On the one hand, we can implement the Pointcut element as a text annotation steaming out of an advice. This text annotation would then contain the query information and it might also be linked by more advices in case the query needs to be reused. On the other hand, we also considered the possibility of having our own modeling element for a Pointcut. To achieve this we could only define a *CustomTask* that would also contain the query and could also be linked to several advices independently. Both options were implemented but only the light-weight notation was kept active. Part of the future work could be to allow the user to switch between these representations.
- **Join point:** The join point element does not need to be defined in an explicit way. A join point according to our initial definition would be any element that could be selected by the user-defined pointcuts.
- **Proceed element:** In this case we opted for the opted for the option of defining our own *CustomTask* that would appear in the user palette as a proceed element for the user to use.

The following sections describe in more detail how the above described approach was implemented. All the different extensions we define from now on must point to our runtime. In order to achieve this all these elements will have their *runtimeId* property set to *ao4bpmn.AO4BPMNRuntimeextension*.

5.1.3 Editor model extension

In order to inform the tool about how our properties must be serialized we have to define our editor model extension. We do this by defining our own Ecore model and registering it to our runtime extension. The main purpose of this Ecore model is to inform the tool about how our properties must be serialized.



```
platform:/resource/ao4bpmn/model/ao4bpmn.ecore
  ao4bpmn
    DocumentRoot -> DocumentRoot
      ExtendedMetaData
        name ->
        kind -> mixed
        namespace -> ##targetNamespace
      isAdvice : EBoolean
        ExtendedMetaData
          kind -> attribute
          name -> isAdvice
          namespace -> ##targetNamespace
      isAspect : EBoolean
        ExtendedMetaData
          kind -> attribute
          name -> isAspect
          namespace -> ##targetNamespace
      pointcutQuery : QueryType
        ExtendedMetaData
          kind -> attribute
          name -> pointcutQuery
          namespace -> ##targetNamespace
    QueryType [java.lang.String]
      ExtendedMetaData
        name -> queryType_._type
        baseType -> http://www.eclipse.org/emf/2003/XMLType#string
platform:/plugin/org.eclipse.bpmn2/model/BPMN20.ecore
```

Figure 22 Ecore structure showing our extension properties

Figure 22 shows how the *DocumentRoot* element from the BPMN2 meta-model is extended. This element is the root element and contains all the definitions and available modeling elements and properties of the BPMN2 meta-model. Each of the properties must have its own *ExtendedMetaData* element that holds the necessary information for the tool to handle the generation of these properties.

After defining our editor model extension we must map the properties that we defined to the actual modeling elements. To achieve this we must define our own modelExtension elements in the plugin file as shown in Figure 23.

```
<modelExtension
    id="ao4bpmn.modelExtension.SubProcess"
    runtimeId="ao4bpmn.AO4BPMNRuntimeExtension"
    name="Advice extension"
    type="SubProcess">
        <property name="isAdvice" value="false"/>
        <property name="pointcutQuery" value="No Query"/>
</modelExtension>

<modelExtension
    id="ao4bpmn.modelExtension.Participant"
    runtimeId="ao4bpmn.AO4BPMNRuntimeExtension"
    name="Aspect extension"
    type="Participant">
        <property name="isAspect" value="false"/>
</modelExtension>
```

Figure 23 XML fragment corresponding to the editor model extension

To finish we must ensure that our model has a valid URI therefore we need to register the model package to the extension *org.eclipse.emf.ecore.generated_package* as shown in Figure 24.

```
<extension point="org.eclipse.emf.ecore.generated_package">
   <package
         class="ao4bpmn.model.Ao4bpmnPackage"
         genModel="model/ao4bpmn.genmodel"
         uri="http://ao4bpmn/1.0">
   </package>
</extension>
```

Figure 24 XML fragment that registers our ecore model to the extension *org.eclipse.emf.ecore*

5.1.4 PropertyTab extensions

Once we have our properties defined we need to make them available to the end user so he can set their values whenever aspect-oriented modeling elements would be required. The properties of the different modeling elements are always available through the property tabs. In order to make our properties available we must therefore extend these property tabs as shown in Figure 25.

```
<propertyTab
    id="ao4bpmn.AspectTab"
    runtimeId="ao4bpmn.AO4BPMNRuntimeExtension"
    class="ao4bpmn.properties.AspectPropertySection"
    type="org.eclipse.bpmn2.Participant"
    label="AO4BPMN">
</propertyTab>
```

Figure 25 XML fragment corresponding that extends the *PropertyTab* of the *Participant* element

A part from this we need to define the corresponding *PropertySection* classes. That means that we need to create our *AspectPropertySection* as well as our *AdvicePropertySection* class. This class extends the *DefaultPropertySection* class and we only need to override two methods:

- *appliesTo(p:IWorkbench, s:ISelection):Boolean*. The aim of this function is for the platform to know wether a property tab must be rendered or not when certain element is selected in the canvas.
- *createSectionRoot():AbstractBpmn2Composite*. The aim of this function is to create the *Composite* element that contains the real elements of the property tab.

To have our *PropertyTabs* ready to use we need therefore one more thing: The *Composite* classes. That means we must also define an *AspectPropertiesComposite* and an *AdvicePropertiesComposite* class that hold the real implementation of these properties. These classes are responsible for the rendering of the properties as well as of the logic behind setting and unsetting them.

5.1.5 Proceed element as a CustomTask

In order to define our *Proceed* model element we decided to directly define our own *CustomTask*. The advantage of this method is that this element will appear in the palette for the user to use it directly. This is a much more comfortable approach and also it gives user friendlier result. The only restriction is that you can only define custom tasks, not any other modeling element.

To do so we need first of all to add the following to our plugin file:

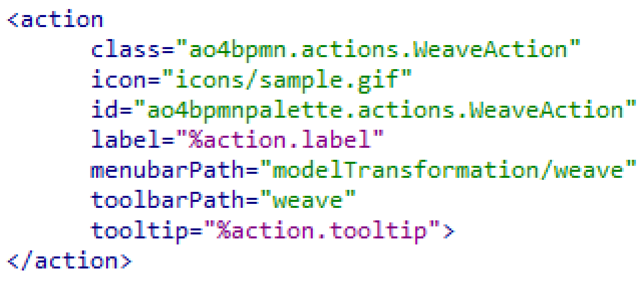
```
<customTask
    featureContainer="ao4bpmn.features.ProceedTaskFeatureContainer"
    id="ao4bpmn.ProceedTask"
    runtimeId="ao4bpmn.AO4BPMNRuntimeExtension"
    name="Proceed"
    description="%customTask.description"
    type="Task">
</customTask>
```

Figure 26 XML fragment corresponding to a *CustomTask*

In order to define the behavior of this *Proceed* element we have to provide the tool with our own *FeatureContainer*. Our *ProceedTaskFeatureContainer* class will only make sure that the name of this task is always set to “proceed”. We do not define any decorator or any other specific functionality.

5.1.6 Triggering the weaving action

In order to allow the end user to trigger the weaving we must also extend the extension point *org.eclipse.ui.actionSets* with our own action as shown in Figure 27.



```
<action
      class="ao4bpmn.actions.WeaveAction"
      icon="icons/sample.gif"
      id="ao4bpmnpalette.actions.WeaveAction"
      label="%action.label"
      menubarPath="modelTransformation/weave"
      toolbarPath="weave"
      tooltip="%action.tooltip">
</action>
```

Figure 27 XML fragment correcponding to the weaving action

This will create an action button and menu entry that will allow the user to trigger the weaving action as described in the next section.

5.2 Weaving

Aspect models are semantically separated from the process models therefore an appropriate composition method is needed. We are dealing with several models as input containing both aspect related information and the different BPMN processes.

Section 5.2.1 describes the technologies used in the implementation of the weaving algorithm. Sections 5.2.2, 5.2.3 and 5.2.4 talk about the steps previous to the algorithm itself. In Sections 5.2.5 and 5.2.6 we present the algorithm itself and the data structures used. Sections from 5.2.7 to 5.2.9 present special cases and problems related to the algorithm as well as the approach adopted to solve them. We conclude this section by presenting a discussion about the termination and cost of the implemented algorithm in Section 5.2.10.

5.2.1 Implementation technology

The main goal of the weaving algorithm, as we described in Section 4.6, is to compose aspects and processes. Such an algorithm needs one or more models as an output taking also one or more models as input. Model to Model transformations, introduced in Section 2.3, deal with this sort of inputs and outputs.

QVTo [7] as well as BPMN [2] is based on an OMG standard, which makes it a good candidate, even though it is not as mature as ATL [10]. ATL is not based on official OMG standards therefore by using QVTo we ensure that future reuse of this implementation for other tools and contexts that are also based on OMG standards is possible. The reason we choose QVTo and not other members of the QVT language family is because of its operational nature. ATL and declarative QVT are declarative languages based on rule matching or mapping definitions. Due to the nature of our problem, which will be discussed in detail in the next section, it is more convenient for us the use of an operational language. As we mentioned in Section 4.6.2 the weaving can be reduced to a clone & merge algorithm. Using a declarative language to implement such an algorithm is not the most appropriate solution. Listing 4 shows a code fragment of an ATL transformation that maps *Authors* to *People*. A ruled-based language like ATL will not satisfy our needs. Listing 5 shows a code fragment of a QVTo mapping function that also maps Authors to People. The procedural nature of QVTo will help us better in the implementation of a clone & merge algorithm. *MMAuthor* and *MMPerson* in Listings 4 and 5 refer to the different meta-models to which the models *A* and *P*, that contain the classes *Author* and *Person*, are conform to.

```
Rule Author {
      From
            A:MMAuthor!Author
      To
            P:MMPerson!Person (
                  name <- a.name,
                  surname <- a.surname
            )
}
```

Listing 4 Code fragment of a model transformation in ATL that maps the class *Author* to *Person*

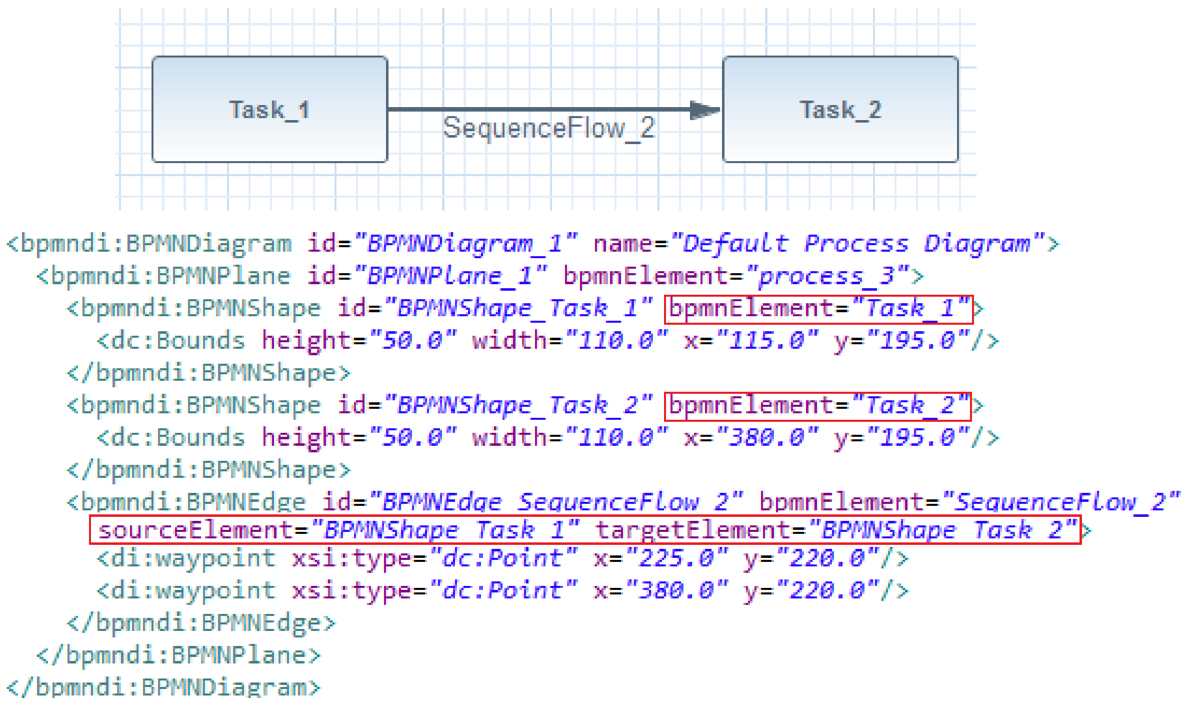
```
Mapping MMAuthor:Author::toPerson() MMPerson:Person {
      name = self.name;
      surname = self.surname;
}
```

Listing 5 Code fragment of a model transformation in QVTo that maps the class *Author* to *Person*

5.2.2 Clone and merge

As we stated in Section 4.6.2 the weaving can be reduced to a clone and merge algorithm. The main idea behind the algorithm is that the structure defined in the different aspects has to be cloned and merged with the selected join points.

The fact that we decided to extend the BPMN2 editor with an eclipse plug-in restricts us to the BPMN2 meta-model used by the editor and also how the graphical representation is managed.



```
<bpmndi:BPMNDiagram id="BPMNDiagram_1" name="Default Process Diagram">
  <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="process_3">
    <bpmndi:BPMNShape id="BPMNShape_Task_1" bpmnElement="Task_1">
      <dc:Bounds height="50.0" width="110.0" x="115.0" y="195.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="BPMNShape_Task_2" bpmnElement="Task_2">
      <dc:Bounds height="50.0" width="110.0" x="380.0" y="195.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_2" bpmnElement="SequenceFlow_2"
      sourceElement="BPMNShape_Task_1" targetElement="BPMNShape_Task_2">
      <di:waypoint xsi:type="dc:Point" x="225.0" y="220.0"/>
      <di:waypoint xsi:type="dc:Point" x="380.0" y="220.0"/>
    </bpmndi:BPMNEdge>
  </bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
```

Figure 28 XML fragment that shows how tasks are related by means of a sequence flow

As we can see in Figure 28 the objects referencing the activities do not have knowledge of the sequence flow that binds them. This information is in the association itself. This is an important problem that significantly influenced the designed weaving algorithm. Because of this restriction and given only a *Proceed* element and a join point we decided the main body of the algorithm to have the structure depicted in Listing 6.

```
...
while currentLevel != ∅ do
      foreach edge in edges do
            clone(edge);
      endForEach;
      ...
endWhile;
...
```

Listing 6 Pseudocode fragment illustrating the iteration strategy used in the weaving algorithm

The idea is that we are iterating over edges because they will give as direct access to the information we need to clone. Listing 6 shows how we iterate over the edges until *currentLevel* is empty.

5.2.3 Input and output

In the most generic of the cases we see that the algorithm needs to produce one or more models as an output taking also one or more models as input. It also can happen that the aspect-oriented information and the target processes are in the same model. That is why we need to generally define the structure of our input and output before starting with the algorithm design.

To decrease the complexity of the algorithm we will design consider only as input a model containing the aspect-oriented information and another model containing the target processes. In the case of this information being in the same model, this can be both inputs at the same time.

For simplicity purposes the join point selection will not be performed by the algorithm. This will also give us more flexibility for the implementation of such a selection process. That means that our algorithm will also take as input the unique ids of a proceed element and a selected joinPoint. This also means that the algorithm will have to be executed for every pair of proceed elements and selected join point if applicable, that is, if the join point is a candidate to match this proceed element.

As an output we will generate a model containing the applied weaving. This output model will be again used as input if more pairs exist so the different weavings will be applied sequentially without any specific order. Once all the applicable pairs of join points and proceed elements have been weaved we will have several output models generated as a results of the whole process.

5.2.4 Join point selection

This is a crucial step in the process and it can also be approached in several ways. We need to provide support for some kind of selection language that would allow the user to define their own aspects. In Section 4.2 we discussed the need of a query-based Pointcut Language and we listed several alternatives. All of the described options has advantages depending on addressed usertherefore we decided to implement our own extension point. Our extension offers OCL [9] as a built-In query-based pointcut language but any developer can register their own languages.

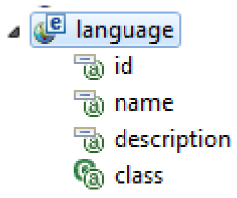


Figure 29 Pointcut query language extension point structure

As we can see in Figure 29 our language extension point needs a unique id and a name and optionally a description. The class attribute contains a reference to the class that implements the desired language. This class needs to implement only two methods:

- *canExecute() : Boolean* This function has access to a global property that represents the query string and its aim is to return whether the query string is executable or not. Here the developer may check for syntax errors, etc.
- *execute() : List<String>* This function has access also to the property representing the query string and a property representing the URI of the queried model. Using this input the aim of the function is to return a list of Strings representing the ids of the selected elements.

5.2.5 Cloning cases and conditions

In Section 4.6.2 we decribe how the algorithm visits the structure we need to clone. In order to guarantee the termination of the algorithm we must guarantee that no edge is treated more than once. A part from this we need to know what levels are already cloned. The algorithm uses two different sets of edges:

- *currentLevel* contains the elements of current level. For instance if we are cloning level 1 this set contains all the elements that are at distance 1 from the proceed element.
- *nextLevel* contains the elements so far cloned of the next level. For example if we are cloning level 1 this set contains all the elements so far cloned that are at distance 2 of the proceed element.

As we are dealing with directed edges we need to take into account source and target elements.

```
function clone(Edge: edge) : void
      if {edge.source} ⊆ currentLevel or
            {edge.target} ⊆ currentLevel then
            //Treat different cloning cases
            ...
      endIf;
endFunction;
```

Listing 7 Pseudocode fragment that illustrates the main cloning condition used by the algorithm

Given the cloning condition shown in the Listing 7 we distinguish the following three cloning cases:

- Both source and target belong to the current level: This means that they have already been cloned therefore only the information related to the link between these elements needs to be copied.
- Only the source element belongs to the current level: This means that the target element of the link belongs to the next level.
- Only the target element belongs to the current level: This means that the source element belongs to the next level.

In the two last cases we need to clone the element belonging to the next level and add it to the *nextLevel* set. Once all the edges having either source or target in the *currentLevel* set have been treated we go to the next level.

5.2.6 RealMaps

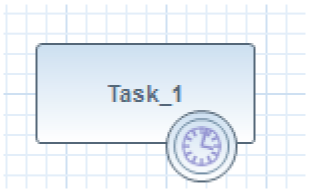
Our algorithm needs to keep track of what elements have been already cloned and, more important, the relationship between an already cloned element and its original element. The reason behind this is that when an element is cloned it might need to be merged with part of the already cloned graph.

To store this information we use two data structures: *RealMap* and *RealEdgeMap*. The aim of these structures is to keep track of the elements, edges and its clones. The idea behind both of them is the same. They are *Maps* with entries that take the original element or edge as keys and store the cloned element as the value.

5.2.7 BoundaryEvents Branching

In BPMN [2] a modeler is allowed to define boundary events. These events indicate that the activity to which they are attached should be interrupted when the event is triggered.

Because of the nature of these events and because of the fact that they are usually used for error handling they will never be candidates for re-linking. What is more, everything directly or indirectly steaming from them will never be considered by the re-linking strategy.



```
<bpmn2:task id="Task_7" name="Task_1"/>
<bpmn2:boundaryEvent id="BoundaryEvent_1" name="Boundary event"
  attachedToRef="Task_7">
  <bpmn2:timerEventDefinition id="TimerEventDefinition_1"/>
</bpmn2:boundaryEvent>
```

Figure 30 XML fragment that illustrates how a *boundaryEvent* is represented by the editor

As we can see in Figure 30 only the events themselves have the necessary information to proceed and they will not be found by our strategy that iterates over links for these links are "virtual".

To overcome this issue we need to identify and register these events beforehand as it is shown in Listing 8. Right after cloning an element we check if the element connected to it (that is, the element that will be treated in the following level) has one or more boundary events and we clone them as if they would be in this same following level (distance to the proceed element). Using this strategy we guarantee that these events will also be considered normally by the algorithm.

```
...
if currentLevel ⊆ {source} then
      Link.cloneTarget(source);
      nextLevel += target;
      Events := target.getBoundaryEvents();
      forEach event in Events do
            cloneBoundaryEvent(event);
            nextLevel += event;
      endForEach;
endIf;
...
```

Listing 8 Pseudocode fragment illustrating the treatment of boundary events

5.2.8 Data branching

This is a similar case as the one previously described. Data branches are also not considered by the re-linking strategy because of their nature. In this case there is no need of any beforehand identification because, as we can see in Figure 31, these elements are linked via associations.

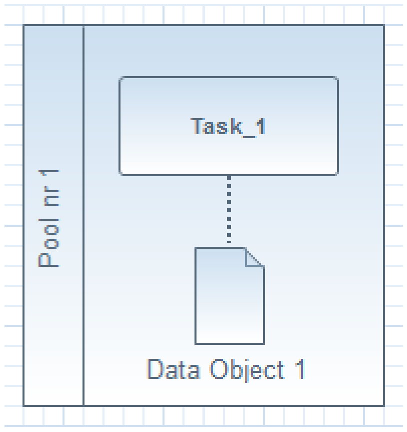


Figure 31 Example of a simple data association

Data branching also includes another issue to take into account: edge targeting. As we can see in Figure 32 the BPMN2 editor allows us to model similar structures.

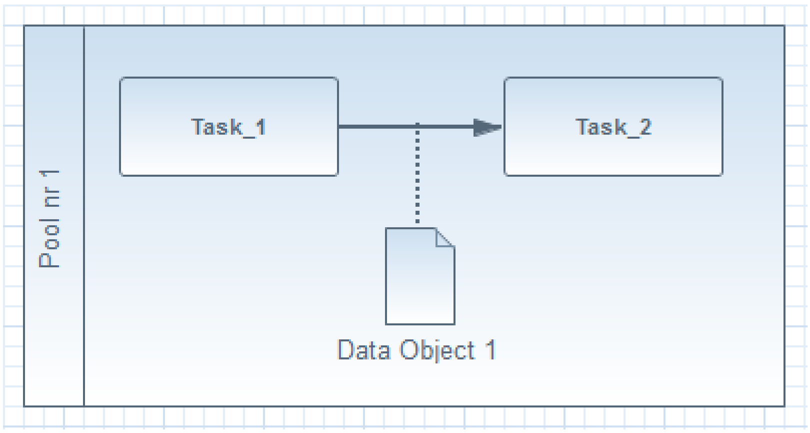


Figure 32 Example of a data association targeting an edge

This case is also considered by the algorithm. The extended editor also allows us to link objects to an association which at the same time targets another association (see Figure 33). Due to the unclear semantics of such a situation the algorithm ignores such a case.

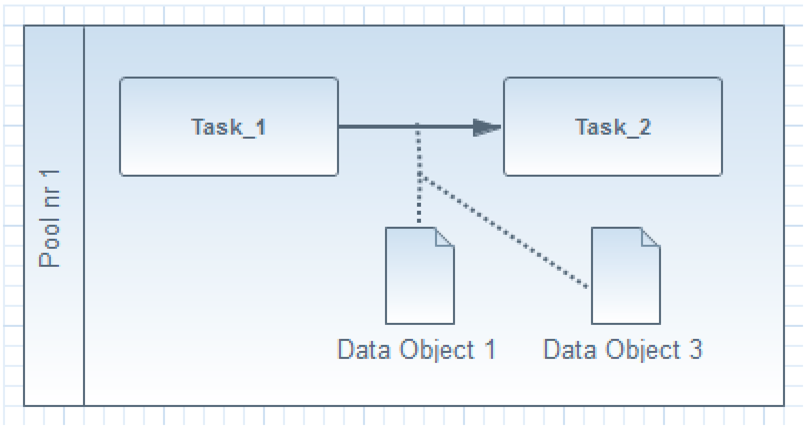


Figure 33 Example of a non clear modeling situation

5.2.9 Unique ids

During the transformation the algorithm creates new elements so we must overcome the problem of generating unique ids for these new elements. In order to achieve this we will define our own id generation strategy. We face two different cases:

- We create a new element as a copy of an existing element
- We create a new element from nothing (i.e., the generated gates in the re-linking process)

In the first case we will add a suffix to the original id to produce a new id and in the second case we will give them a fixed name. We must note that in the second situation we can assume that in the worst of the cases only four gateways will be created per transformation. As we can see this technique does not guarantee uniqueness in any of the cases. To solve that we will add a transformation prefix that will be generated before each execution of the weaving algorithm, that is, for every applicable pair of selected join point and proceed element. The uniqueness of this id is checked beforehand therefore if the prefix is unique we can imply that all the ids generated with that prefix and the previously mentioned strategy will be unique.

5.2.10 Cost and termination

In this section we will talk about the cost of the developed algorithm from a theoretical point of view and we will justify its termination.

We will focus on the concrete problem of having only one advice and one selected join point in, what we can assume, an undirected graph. Despite the fact that we work with directed edges we are going to treat them as undirected. That is because we cannot model a situation in which a *Task_1* is connected to a *Task_2* being this last one the target of the relationship and at the same time this *Task_2* is connected to *Task_1* being the latter the target of this other relationship. That means we can ignore the fact that in BPMN we can distinguish between source and target therefore simplifying our problem.

Let us consider the worst of the cases in which we have a complete graph, that is a graph *G = (V, E)* where for every pair of nodes *a* and *b* that *a, b* ⊆ *V* , it is true that {*a, b*} ⊆ *E*. By definition we have that a complete graph *G = (V, E)* :

$$|V| = n; \; |E| = \frac{n\,.(n-1)}{2}$$

This means that a complete graph *G* with *n* nodes is the simple graph with more nodes among all simple graphs with *n* nodes.

Let us also consider the unique existence of a node p_k with $0 \leq k \leq n$ representing the proceed element and $d_{i,j}$ the distance between a node p_i and another node p_j. Let d_{max} be the maximal distance, that is:

$$\forall i \mid 0 \;\leq i < n \; (d_{\max} \geq \mathrm{d}_{\mathrm{k,i}})$$

The algorithm iterates over all the edges as many times as levels the graph has. In the previous sections we defined the level of a graph *G* as a set of all the nodes that are at the same distance of the proceed element and we defined distance between two nodes as the number of edges in the shortest path that links the two elements. Therefore, we can conclude that the number of levels in our *G* graph as previously defined equals d_{max}. With all this information we can already calculate the cost of our algorithm:

$$\mathcal{O}\,(d_{\max}\,.\frac{n\,.(n-1)}{2}) = \mathcal{O}\;(\;n^2\;)$$

In practice, the algorithm will be executed for every pair of *Proceed* element and selected join point but this number is of a much lower magnitude than the *n* nodes of the graph therefore does not significantly influence the already calculated cost. There are other steps in the final implementation that require a single iteration over the whole set of nodes (i.e., registration of boundary events) but again this steps are not costly enough as to influence in this calculations. This leads us the following conclusion: Given a simple undirected graph *G* = (*V, E*) with |*V*| = *n*, the cost of our algorithm is

$$\mathcal{O}\;(\;|E|\;) \leq \mathcal{O}\;(\,n^2\,)$$

Concerning the termination of the algorithm we will give some informal comments about it in the following lines.

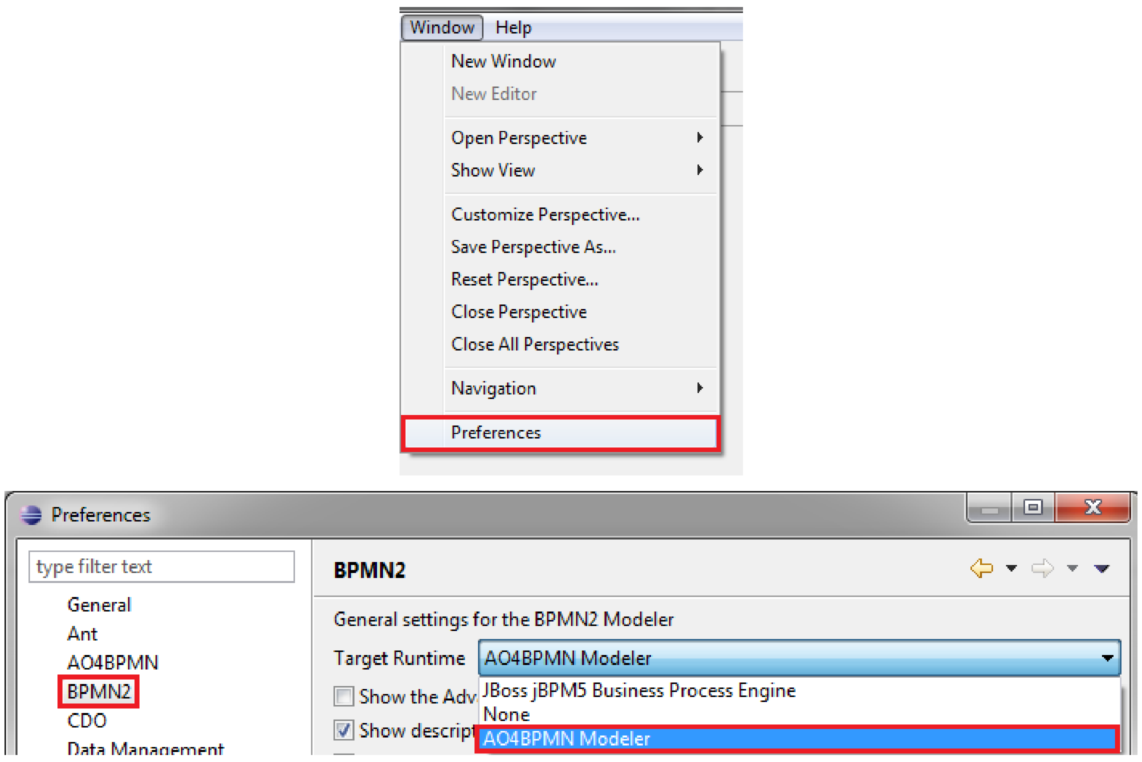
As we explained before the algorithm uses the sets *currentLevel* and *nextLevel* to explore the different levels of our graph. We can informally prove that our algorithm terminates by observing the fact that we are always in the domain of finite graphs and that we only visit the nodes maximum once. That means that at some point the set *nextLevel* will be empty and the algorithm will terminate.

6 Case study

In this chapter we illustrate the defined concepts and the respective implementations by means of a case study. There are two main parts to focus on: The editor tool and the weaving mechanism. Section 6.1 illustrates how the base processes of the case study are modeled. Section 6.2 describes how aspects are modeled. Section 6.3 shows how the weaving is started and the resulting processes.

6.1 Modeling the processes

In this section we will model the processes of the example introduced in Section 3.1. We first open Eclipse and we switch to our runtime extension by means of the preferences menu. These steps are shown in Figure 34.

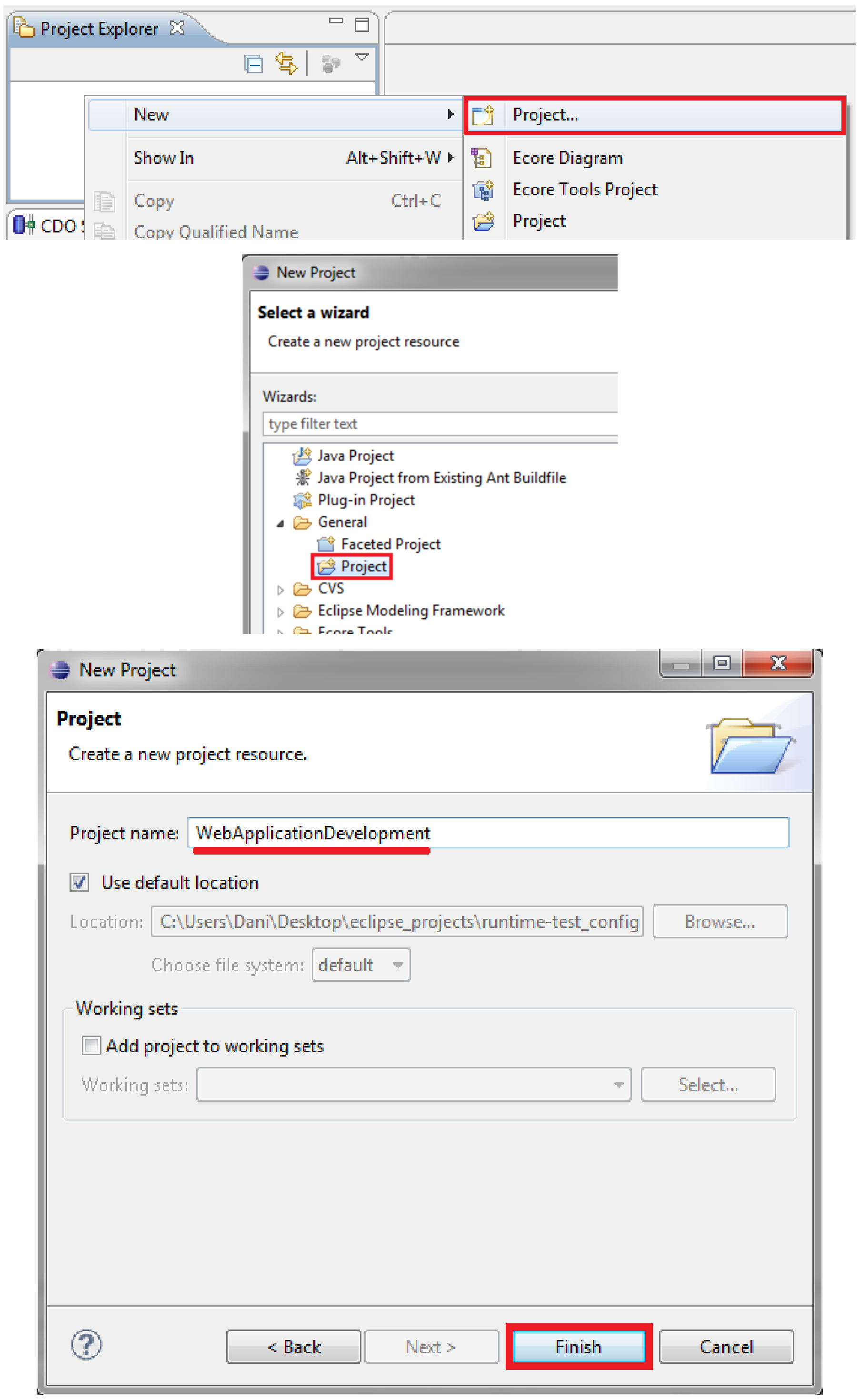


General settings for the BPMN2 Modeler

Target Runtime: AO4BPMN Modeler

Figure 34 Screenshots illustrating how to change the Target Runtime

We create an empty project called *WebApplicationDevelopment* by means of the default Eclipse project creation wizard. These steps are shown in Figure 35.



Project name: WebApplicationDevelopment

[✓] Use default location

Location: C:\Users\Dani\Desktop\eclipse_projects\runtime-test_config

Choose file system: default

[ ] Add project to working sets

Working sets:

Figure 35 Screenshots illustrating how to create the project

Then, we create a new diagram called *FeaturesAndBugs* by following the steps shown in Figures 36 and 37. After that we create the diagram by adding the corresponding modeling elements, which leads to the diagram shown in Figure 38.

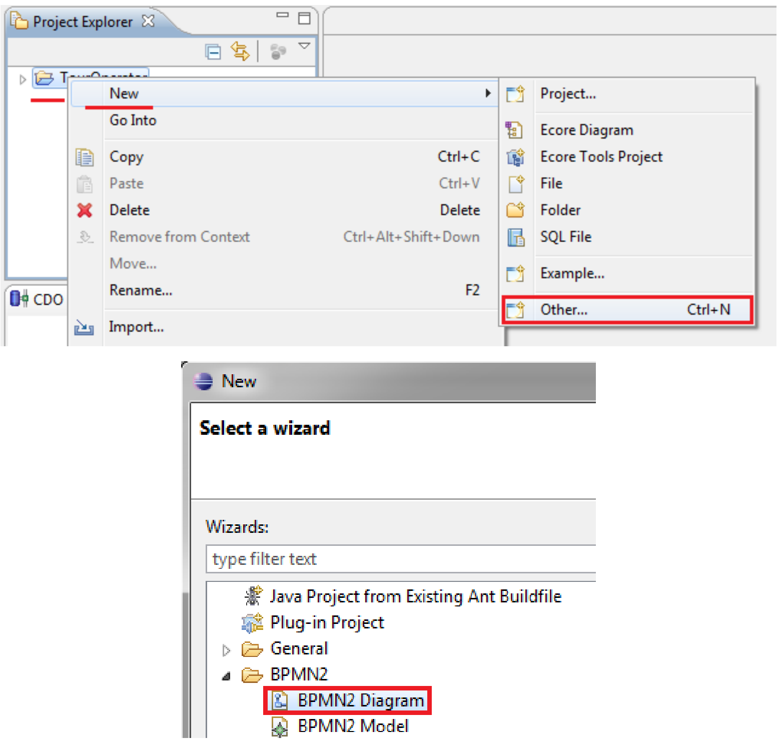
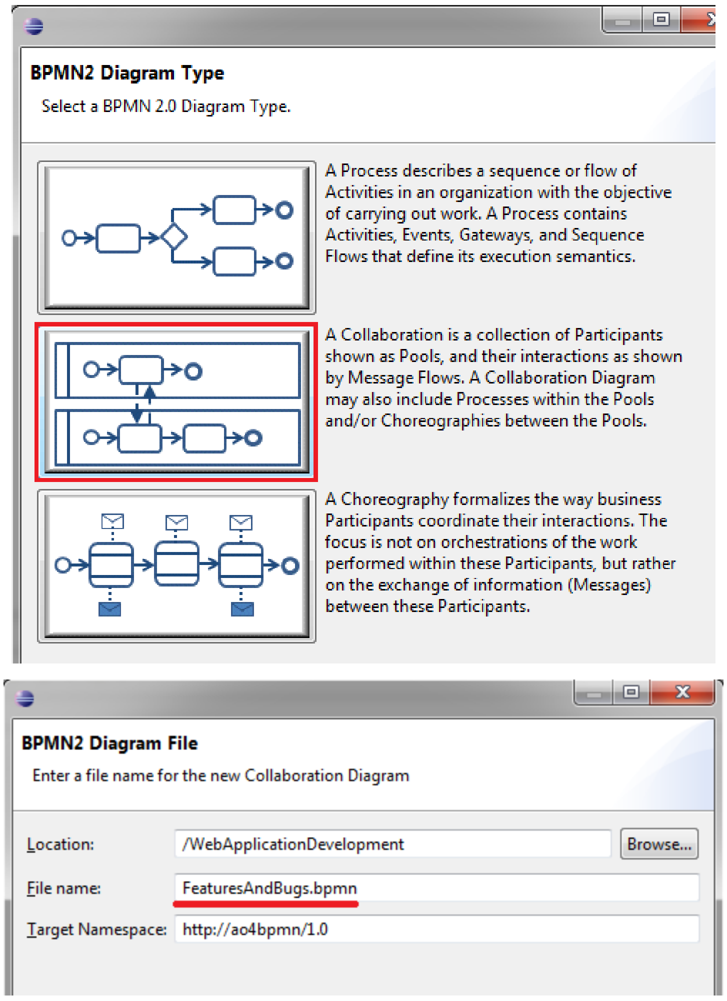


Figure 36 Screenshots that illustrate the steps to open the BPMN2 diagram creation wizard



A Process describes a sequence or flow of Activities in an organization with the objective of carrying out work. A Process contains Activities, Events, Gateways, and Sequence Flows that define its execution semantics.

A Collaboration is a collection of Participants shown as Pools, and their interactions as shown by Message Flows. A Collaboration Diagram may also include Processes within the Pools and/or Choreographies between the Pools.

A Choreography formalizes the way business Participants coordinate their interactions. The focus is not on orchestrations of the work performed within these Participants, but rather on the exchange of information (Messages) between these Participants.

Location:	/WebApplicationDevelopment
File name:	FeaturesAndBugs.bpmn
Target Namespace:	http://ao4bpmn/1.0

Figure 37 Screenshots illustrating how to create a diagram

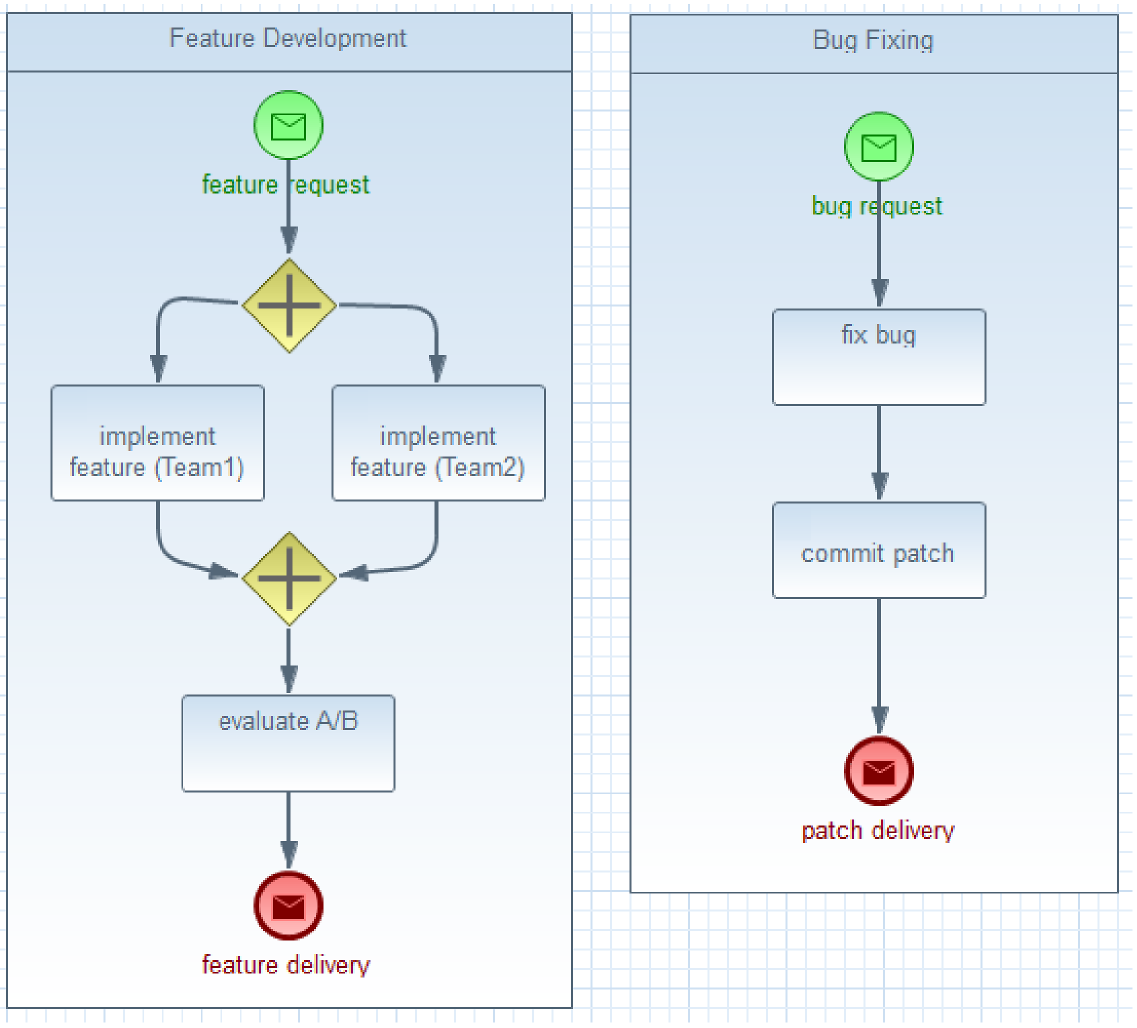


Figure 38 Screenshot illustrating the resulting diagram

6.2 Modeling the aspects

After modeling the business processes in the previous section we model in this section the monitoring and compliance aspects. To do that, we create two collaboration diagrams called Monitoring and Compliance following the same steps shown in Figures 36 and 37. Once these diagrams are created we add the necessary elements to build the aspect-oriented models for these crosscutting concerns. To identify the aspect and the advice we must check the *isAspect* and *isAdvice* checkboxes in the AO4BPMN property tabs as shown in Figures 40 and 42. The resulting aspect models are shown in Figure 39 and 41 after manually laying out their model elements.

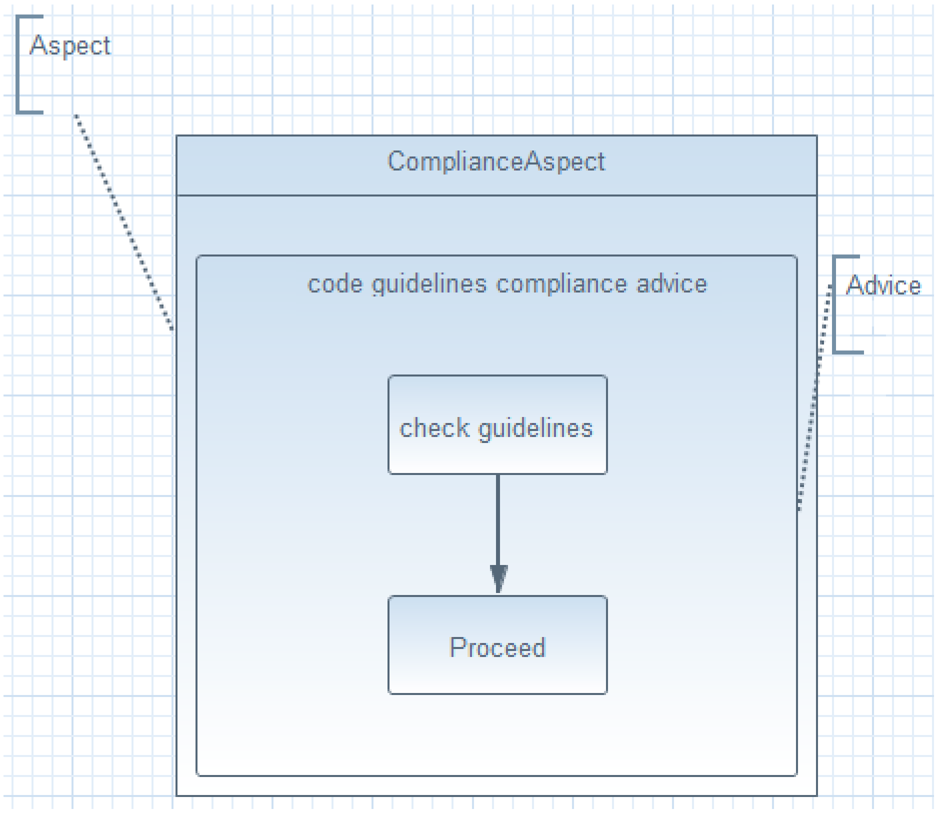
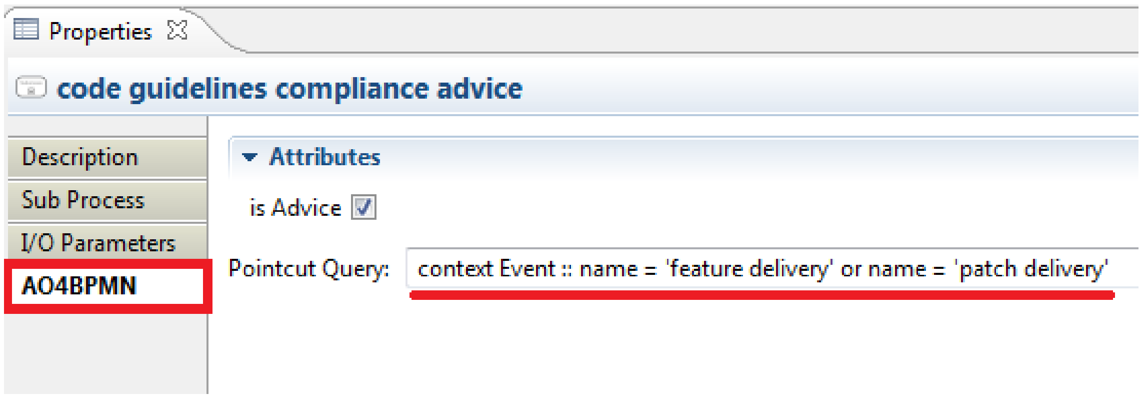


Figure 39 Screenshot illustrating the resulting compliance aspect

In this case study we use the light-weight notation to represent the aspect-oriented constructs.



is Advice [✓]

Pointcut Query: context Event :: name = 'feature delivery' or name = 'patch delivery'

Figure 40 Screenshot illustrating the property tab of the compliance advice

Next, we see in Figure 40 the pointcut definition of the compliance aspect, which is done by writing an appropriate OCL query in the AO4BPMN properties tab. In this example we use an OCL condition in the context of the type Event that compares by name.

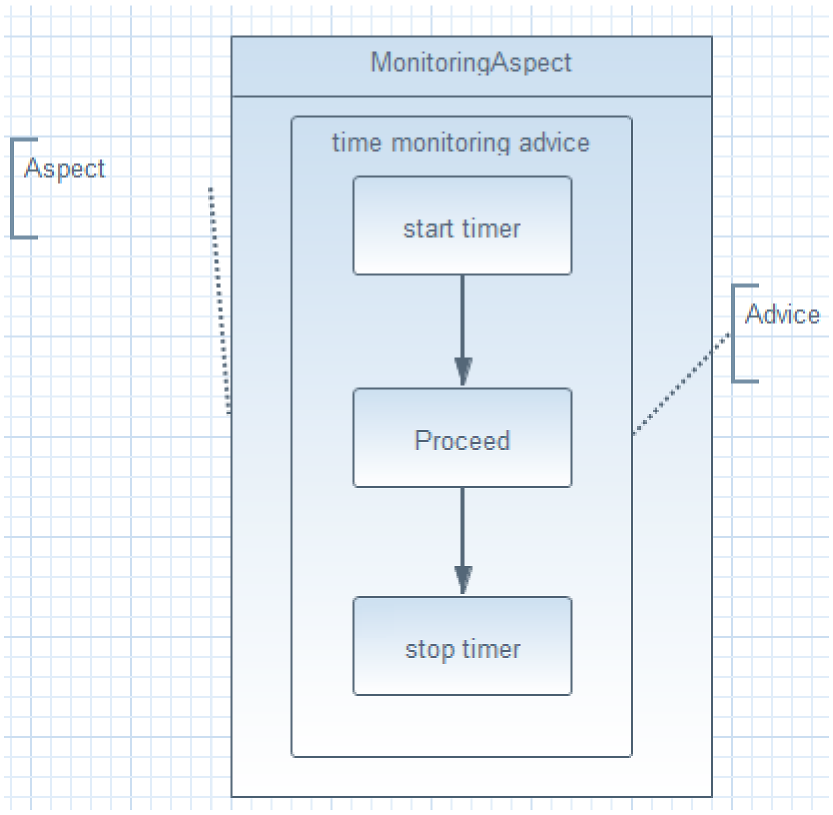
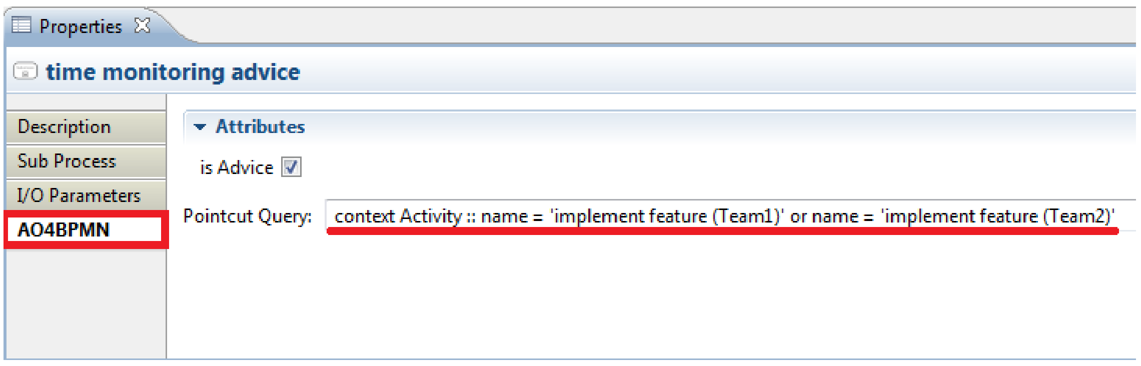


Figure 41 Screenshot illustrating the resulting monitoring aspect



is Advice ☑

Pointcut Query: context Activity :: name = 'implement feature (Team1)' or name = 'implement feature (Team2)'

Figure 42 Screenshot illustrating the property tab of the monitoring advice

Then, we define in Figure 42 the pointcut query through the AO4BPMN properties tab. In this example we use an OCL condition in the context of the type Activity that compares by name.

6.3 Using the weaver

Next we start the weaver as shown in Figures 43 and 44 to compose the process models and the aspects models. Figure 43 shows the integration of the weaver in Eclipse. Figure 44 shows the selection of the processes and aspects that will be composed. The weaver generates a new process models as shown in Figure 45.

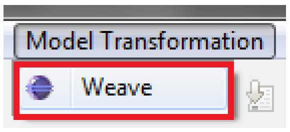
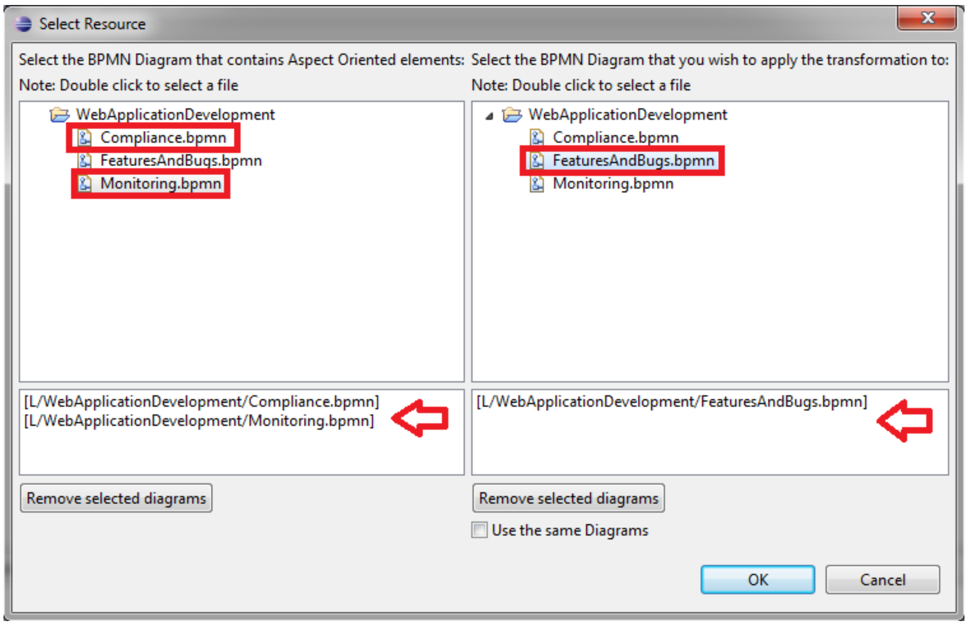


Figure 43 Integration of the weaver in Eclipse



Select the BPMN Diagram that contains Aspect Oriented elements:
Note: Double click to select a file

WebApplicationDevelopment
Compliance.bpmn
FeaturesAndBugs.bpmn
Monitoring.bpmn

[L/WebApplicationDevelopment/Compliance.bpmn]
[L/WebApplicationDevelopment/Monitoring.bpmn]

Select the BPMN Diagram that you wish to apply the transformation to:
Note: Double click to select a file

WebApplicationDevelopment
Compliance.bpmn
FeaturesAndBugs.bpmn
Monitoring.bpmn

[L/WebApplicationDevelopment/FeaturesAndBugs.bpmn]

Figure 44 Screenshot illustrating how to trigger the weaving operation

The wizard shown in Figure 44 allow us to select on the left side the BPMN diagrams containing the aspects we want to include in the weaving and on the right side the diagrams containing the base processes. If the aspects are in the same diagram as the base processes we can use the checkbox with the label *Use the same Diagrams*. By using the buttons labeled as *Remove selected diagrams* we can unselect previously selected diagrams.

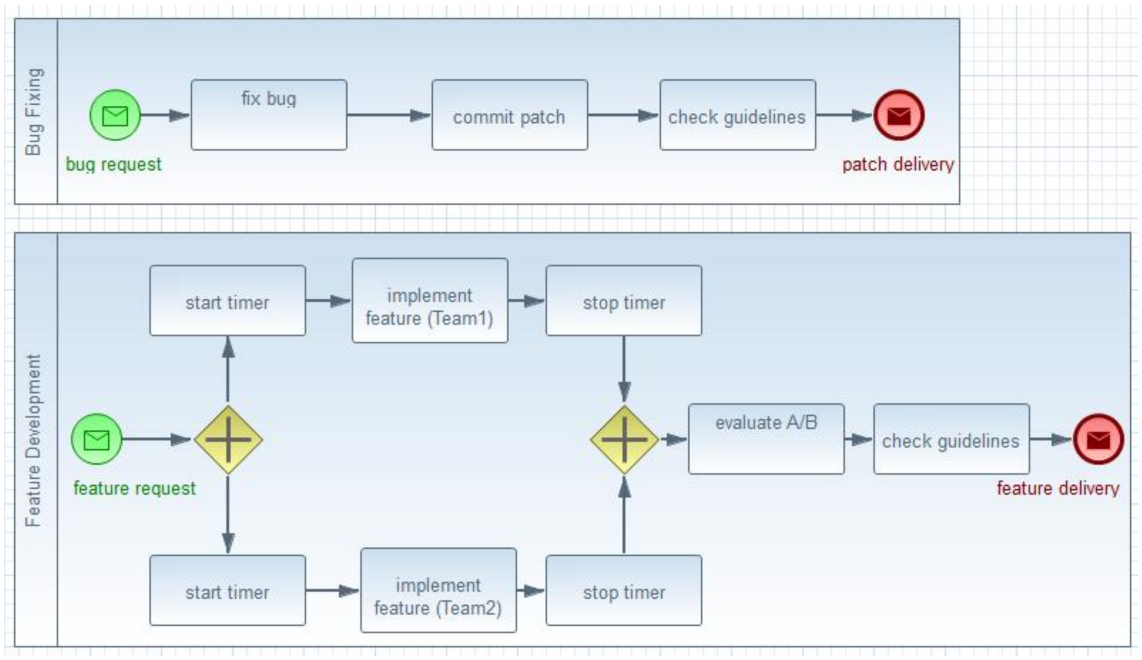


Figure 45 Screenshot illustrating the result diagram after the weaving execution

Figure 45 shows the weaving result. As we can see in that figure the elements modeled as crosscutting concerns in a separate module are now integrated in the base processes and everything is saved in a new file so that the base diagrams are not lost after weaving.

7 Conclusion

This chapter concludes the thesis with a summary of the contents presented and an overview on the limitations of this work as well as directions for future work.

7.1 Summary

In this thesis we motivated the need for a better support for crosscutting concerns in business process modeling languages such as OMG's BPMN. We also introduced and described an aspect-oriented extension of BPMN called AO4BPMN 2.0 that allows the modeler to encapsulate such concerns.

First, we motivated the problems of crosscutting concerns with an example of a web application development company. Then, we discussed the limitations of AO4BPMN 1.0. After that, we presented the design of AO4BPMN 2.0 and reported on the implementation of the editor and the weaver. AO4BPMN 2.0 is a refinement of AO4BPMN 1.0. We implemented an editor for AO4BPMN 2.0 by extending an Eclipse-based BPMN2 editor. Further we implemented a weaving mechanism for aspects and process using a model-to-model transformation and the QVTo language. We also discussed the details of the weaving mechanism and the underlying algorithm. We discussed different alternatives related to the model transformation language to choose and also related to which graphical notation should the editor tool support. We offered built–in support for OCL as a pointcut language and we provided an extensible design that would allow the users to plug in their own pointcut languages. For this purpose we have implemented an appropriate extension mechanism. Finally, we illustrated the concepts and the implementation through a case study based on the motivation example.

7.2 Limitations and future work

AO4BPMN 2.0 has some conceptual limitations that could be addressed in future work:

- AO4BPMN 2.0 does not support the concepts of choreography and collaboration. AO4BPMN 2.0 works at a process level. AO4BPMN 2.0 supports processes defined in collaborations but it does not support collaboration and choreography as a unit of work.
- AO4BPMN 2.0 supports only OCL as pointcut language. However, it gives the possibility for other developers to add support to other languages by providing appropriate extension mechanisms. In the future, further pointcut languages beyond OCL can be provided.
- The current implementation does not support the transformation of AO4BPMN 2.0 to executable aspects (for example in AO4BPEL [11, 12]). Part of the future work could be to implement such a transformation and offer it as an integrated functionality with the editor.

In addition, the current implementation has the following limitations, which can be addressed in future work:

- The concepts of AO4BPMN are not integrated to the built-in Graphiti palette defined by the BPMN2 editor tool being extended. This limitation is due to the currently provided extension points. The BPMN2 modeling tool does not offer palette extension mechanisms.
- There is not yet auto-layouting support. Part of the future work could be to integrate the Graphiti auto-layouting plugin of the KIELER project [13] for example.
- The *Proceed* element is mandatory in the current implementation. A future extension can provide support to model using the different advice types such as *before, after* and *around*.
- The current implementation supports only one of the two graphical representations described in [1]. Part of the future work could be to add support for both representations and allow the user to switch between them.
- The current implementation offers support for OCL as pointcut language. Part of the future work could be to offer a built-in wizard that would help non experienced business users in defining their own OCL queries without having to write actual code for example.

References

[1] Anis Charfi, Heiko Witteborg, Mira Mezini. Aspect-Oriented Business Process Modeling with AO4BPMN, in *Proc. of 6th European Conference on Modelling Foundations and Applications* (EC-MFA), LNCS 6138, pp. 48-61, Springer, Paris, France, June 2010.

[2] Object Management Group. Business Process Model and Notation v2.0. http://www.omg.org/spec/BPMN/2.0/. July 2012.

[3] Anis Charfi and Mira Mezini. Aspect-Oriented Workflow Languages. In *Proc. Of the 14th International Conference on Cooperative Information Systems (CoopIS),* volume 4275 of *LNCS*, pages 183-200. Springer, November 2006.

[4] Anis Charfi. Aspect-Oriented Workflow Languages: AO4BPEL and Applications. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany 2007. http://elib.tu-darmstadt.de/diss/000852/.

[5] Gregor Kiczales, Joh Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proc. Of the 11th European Conference on Object-Oriented Pprogramming (ECOOP),* volume 1241 of *LNCS*, pages 220-242. Springer, June 1997.

[6] MDT/BPMN2. Meta-model implementation. http://wiki.eclipse.org/MDT-BPMN2. July 2012.

[7] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. http://www.omg.org/docs/ptc/05-11-01.pdf, November 2005.

[8] Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M. (2009) Controller experiments on the web: survey and practical guide. In *Data Mining and Knowledge Discovery,* volume 18, Number 1, pages 140-181, 2009.

[9] Object Management Group. Object Constraint Language 2.0 Final Adopted Specification. http://www.omg.org/cgi-bin/doc?ptc/2003-10-14, October 2003.

[10] ATL. ATLAS Transformation language. http://www.eclipse.org/atl/. July 2012.

[11] Anis Charfi and Mira Mezini. Aspect-Oriented Web Services Composition with AO4BPEL. In *Proc. Of the 2nd European Conference on Web Services (ECOWS),* volume 3250 of *LNCS*, pages 168-182. Springer, September 2004.

[12] Anis Charfi and Mira Mezini AO4BPEL: An Aspect-Oriented extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue),* March 2007.

[13] Cristian-Albrechts-Universität zu Kiel. Kiel Integrated Environment for Layout Eclipse RichClient (KIELER). http://www.informatik.uni-kiel.de/rtsys/kieler/. August 2012.

[14] Eclipse Model Development Tools (MDT). http://www.eclipse.org/mdt/. July 2012.

[15] Eclipse BPMN2 Modeler. http://www.eclipse.org/bpmn2-modeler/. July 2012.

[16] Eclipse Graphiti. A Graphical Tooling Infrastructure. http://www.eclipse.org/graphiti/. July 2012.

[17] Eclipse Modeling Framework Project (EMF). Model Query. http://www.eclipse.org/modeling/emf/?project=query. July 2012.

[18] VIATRA2. EMF-IncQuery. http://viatra.inf.mit.bme.hu/incquery. July 2012.

[19] Eclipse Modeling Framework project (EMF). Model Query 2. http://www.eclipse.org/modeling/emf/?project=query2. July 2012.

[20] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold. An Overview of AspectJ. In *Proc. Of the 15th European Conference on Object-Oriented Pprogramming (ECOOP),* volume 2072/2001 of *LNCS*, pages 327-354. Springer, June 1997.

[21] AspectR. Simple Aspect-Oriented programming in Ruby. http://aspectr.sourceforge.net/. July 2012.

[22] Schauerhuber, Andrea and Shwinger, Wieland and Kapsamer, Elisabeth and Retschitzegger, Werner and Wimmer, Manuel and Kappel, Gerti. A Survey on Aspect-Oriented Modeling Approaches. Technical Report, Insitut für Softwaretechnik und Interaktive Systeme; technische Universität of Wien, 2007.

[23] Iris Groher, Markus Voelter. Xweave: Models and Aspects in Concern. In *10th Int'l Workshop on Aspect-Oriented Modeling*. Vancouver, 2007.

[24] Eclipse STP/BPMN Termination document. http://wiki.eclipse.org/STP/BPMN_Component/Termination. July 2012.

[25] The Eclipse Foundation. http://www.eclipse.org/org/. July 2012.

[26] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. EMF: Eclipse Modeling Framework, 2nd Edition. Eclipse Series. Addison-Wesley Professional. ISBN-13: 978-0-321-33188-5. 2008.

[27] Object Management Group (OMG). XML Metadata Interchange (XMI) Specification, July 2012.

[28] Object Management Group (OMG) UML Specification. July 2012.

[29] Oracle Java Language Specification. http://docs.oracle.com/javase/specs/. July 2012.

[30] Eclipse Ecore tools. http://www.eclipse.org/modeling/emft/?project=ecoretools. July 2012.

[31] Dan Rubel, Jaime Wren, Eric Clayberg. The Eclipse Graphical Editing Framework (GEF). Eclipse Series. ISBN-13: 978-0321718389. August 2011.

[32] Eclipse Graphical Modeling Framework (GMF). http://www.eclipse.org/modeling/gmp/. July 2012.

[33] Eclipse Draw2d plug-in. http://www.eclipse.org/gef/draw2d/index.php. June 2012.

[34] Eclipse (Model-To-Model) M2M. http://www.eclipse.org/m2m/. July 2012.

[35] Eclipse Modeling Project. http://www.eclipse.org/modeling/. July 2012.

[36] Stevens, Perdita. A landscape of bidirectional model transformations. In *Generative and Transformational Techniques in Software Engineering II (GTTSE)*. Volume 5235/2008 of LNCS, pages 408-424. Berlin / Heidelberg 2008.

[37] Stephan A. White. Introduction to BPMN (PDF). http://www.omg.org/bpmn/Documents/ Introduction_to_BPMN.pdf .IBM, May 2004.

[38] Reiko Heckel. Graph transformation in a nutshell. In *Electronic Notes in Theoretical Computer Science*. Volume 148, Issue 1, pages 187-198. February 2006.

[39] IBM's Eclipse Membership. http://www.eclipse.org/membership/showMember.php?member_id=656. July 2012.

[40] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, Simon Helsen. Model-driven software development – technology, engineering, management. ISBN: 978-0-470-02570-3. Pitman 2006.

[41] Tomas Kühne. Matters of (Meta-)Modeling. In *Software and System Modeling*. Volume 5, Number 4, pages 369-385. 2006.

[42] Eric Clayberg, Dan Rubel. Eclipse Plug-ins (3th Edition). Eclipse Series. ISBN-13: 978-0321553461, 2008.

[43] Fabiana Jack Nogueira Santos, Claudia Cappelli, Flávia Maria Santoro, Julio Cesar Sampaio do Prado Leite, Thaís Vasconcelos Batista. Analysis of heuristics to identify crosscutting concerns in business process models. In *Proceedings of the 27th Annual ACM Symposium of Applied Computing*, pages 1725-1726. Italy 2012.

A. Test Cases for the weaver

In order to evaluate the weaving we designed our own suit of test cases. Basically we defined several sample pairs of models containing a proceed element and a selected join point. It is not the focus of this concrete evaluation to test the selection language therefore we directly provide the algorithm with the id of the proceed element and the id of the selected join point. The test results presented in this section do not pretend to be exhaustive but rather to present some of the most illustrative and significant cases.

The structure of the tests is the following:

- **Simple cases:** This testing package includes a set of simple models. There is no boundary or data branching and they are built using only tasks and sequence flow elements.
- **Complex cases:** This testing package follows the same philosophy as the previous one but here we include more complex structures and elements such as events and gateways but always leaving out special cases like boundary or data branching.
- **Branching cases:** The aim of this package is to test the previously described boundary event and data braching.

In this chapter we describe some of the relevant conducted tests and their results. In order to simplify the results and to guarantee the document's readability we omit advices, aspects and other modeling elements whenever they are irrelevant in the pictures. We also assume that a layout after the weaving has been applied either for a human user or an auto-layout engine. In order to present the test cases and their results we will use the template that appears in Table 2. In some cases we show the test picture on the left side and the expected results on the right side of the same row for space reasons.

Package	<Package name>	**Number**	<Test number>
Advice	**Proceed element**	<Name of the proceed element>	
	Pointcut Query	<Informal description of the query>	
…			
<Test picture>			
<Expected results>			
Result	<results: {As expected, Not as expected}>		

Table 2 Template to show the test results.

A.1. Simple cases

As stated before this testing package includes a set of simple models. There is no boundary or data branching and they are built using only tasks and sequence flow elements.

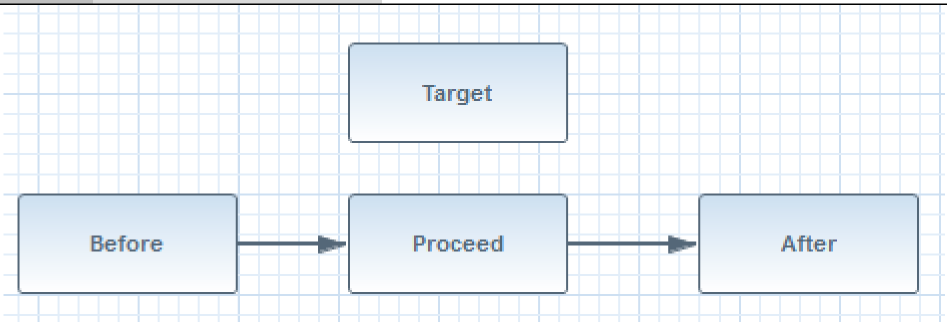
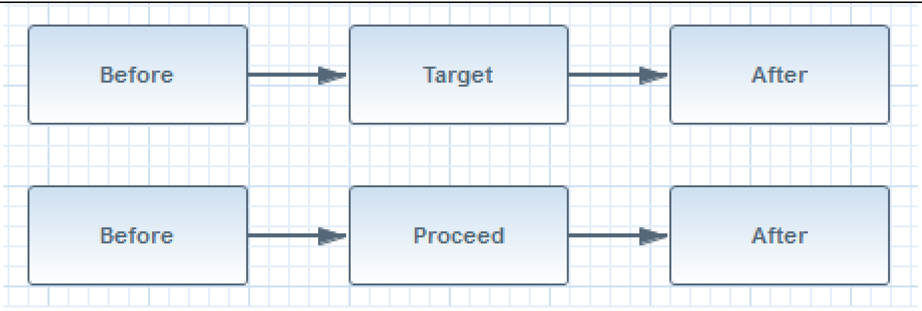
Package	Simple cases	**Number**	1
Advice	**Proceed element**	Element with the name „Proceed”	
	Pointcut Query	Element with the name „Target”	
			
			
Result	as expected		

Table 3 Simple cases test results. Number 1

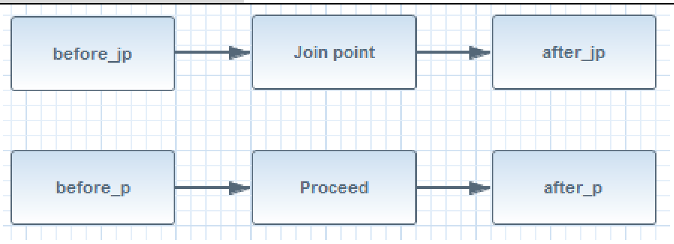
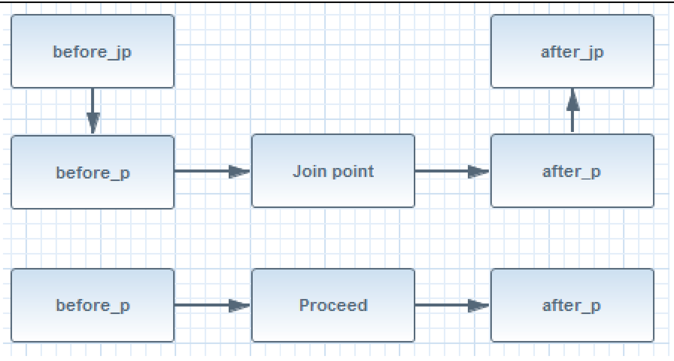
Package	Simple cases	Number	2
Advice	Proceed element	Element with the name „Proceed”	
	Pointcut Query	Element with the name „Join point”	
 before_jp Join point after_jp before_p Proceed after_p			
 before_jp after_jp before_p Join point after_p before_p Proceed after_p			
Result	as expected		

Table 4 Simple cases test results. Number 2

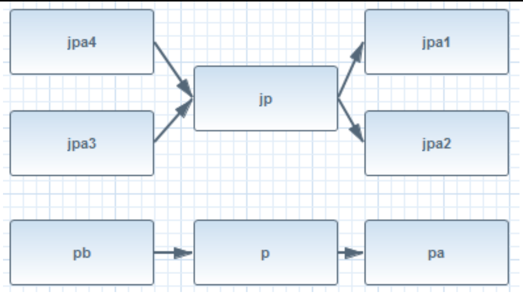
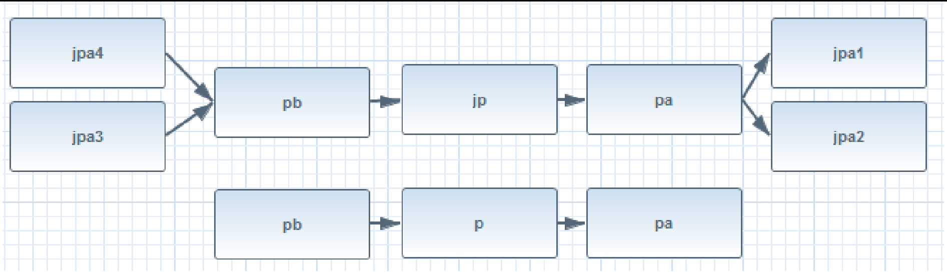
Package	Simple cases	Number	3
Advice	Proceed element	Element with the name „p”	
	Pointcut Query	Element with the name „jp”	
			
 jpa4 jpa1 pb jp pa jpa3 jpa2 pb p pa			
Result	as expected		

Table 5 Simple cases test results. Number 3

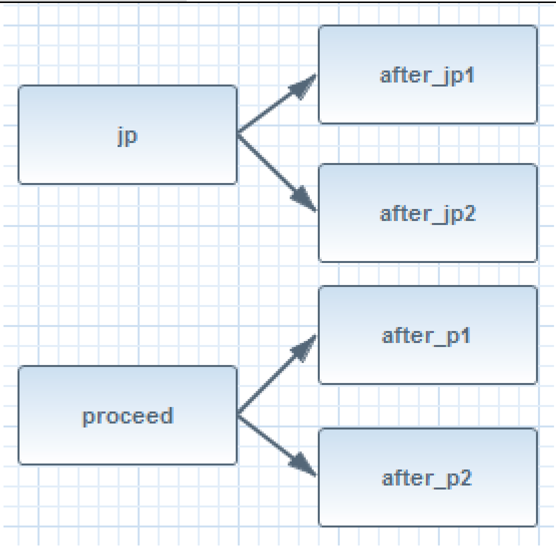
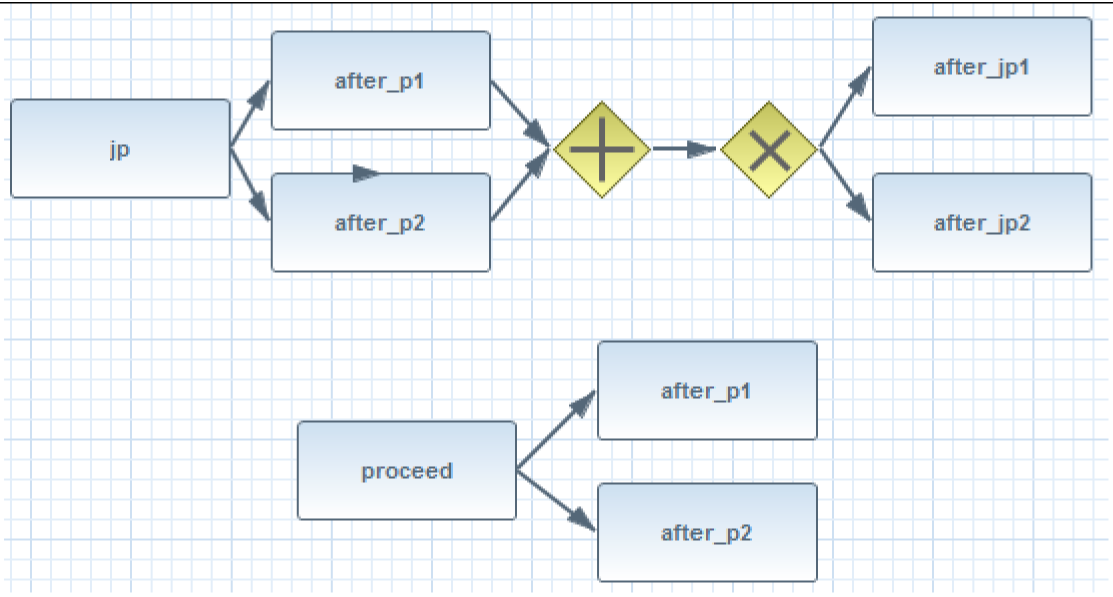
Package	Simple cases	Number	4
Advice	Proceed element	Element with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
			
			
Result	as expected		

Table 6 Simple cases test results. Number 4

A.2. Complex cases

In this section we present some of the tests that belong to the complex cases package. As we stated before these cases do not content any branching but they test situations with elements different than activities.

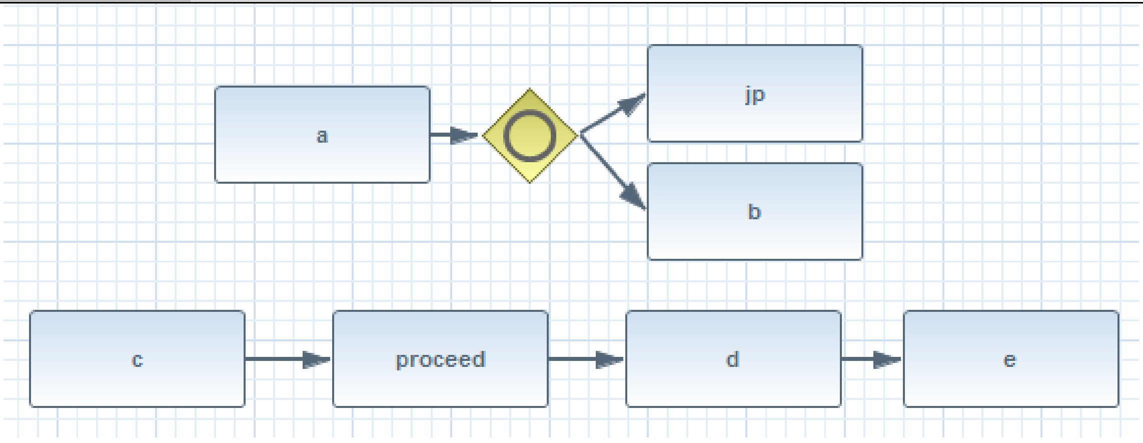
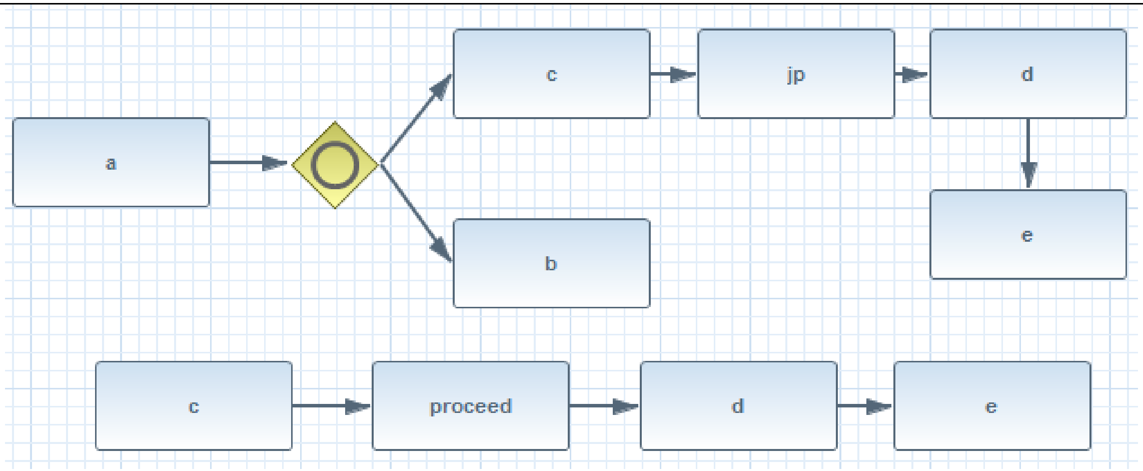
Package	Complex cases	**Number**	1
Advice	**Proceed element**	Eelemtn with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
			
			
Result	🟩 as expected		

Table 7 Complex cases test results. Number 1

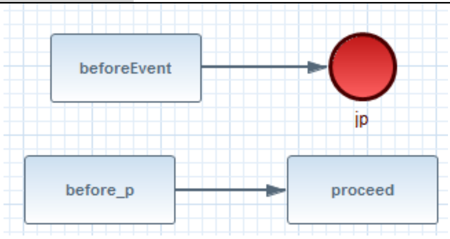
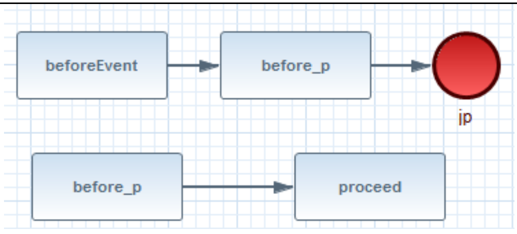
Package	Complex cases	**Number**	2
Advice	**Proceed element**	Element with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
			
			
Result	as expected		

Table 8 Complex cases test results. Number 2

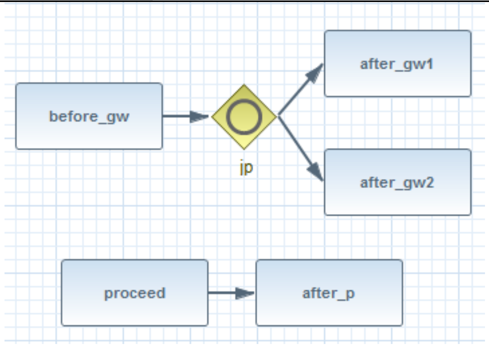
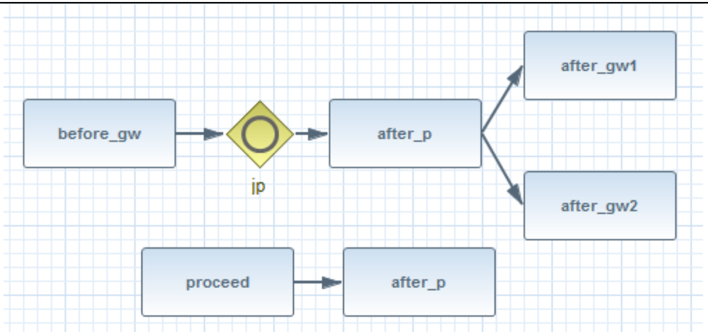
Package	Complex cases	**Number**	3
Advice	**Proceed element**	Element with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
			
			
Result	as expected		

Table 9 Complex cases test results. Number 3

A.3. Branching cases

In this section we will show 2 representative cases of branching. The first example shown in Table 10 is an event branching case and the results of table 11 show a data branching case.

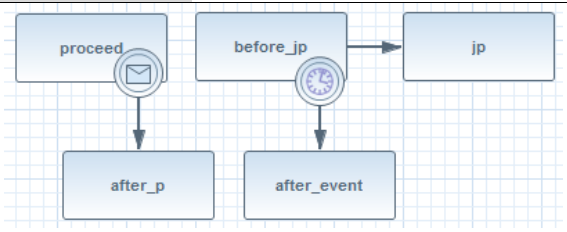
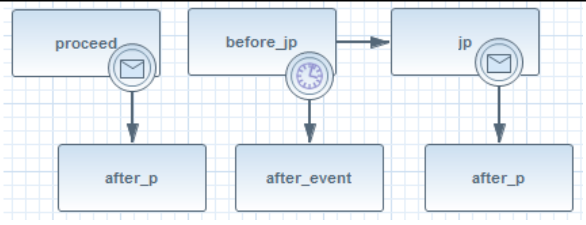
Package	Branching cases	Number	1
Advice	Proceed element	Element with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
			
	 proceed before_jp jp after_p after_event after_p		
Result	as expected		

Table 10 Branching cases test results. Number 1

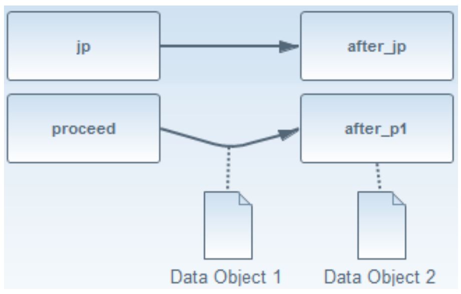
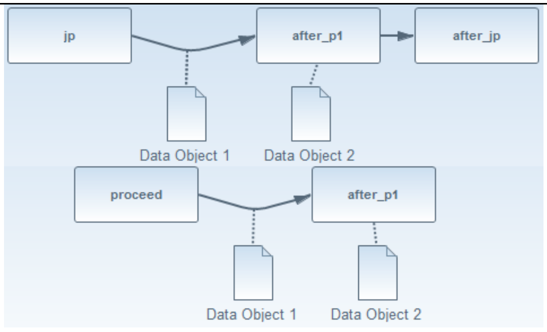
Package	Branching cases	Number	2
Advice	Proceed element	Element with the name „proceed“	
	Pointcut Query	Element with the name „jp”	
	 jp after_jp proceed after_p1 Data Object 1 Data Object 2	 jp after_p1 after_jp Data Object 1 Data Object 2 proceed after_p1 Data Object 1 Data Object 2	
Result	as expected		

Table 11 Branching cases test results. Number 2