Supporting Enactment of Aspect Oriented Business Process Models –
an approach to separate cross-cutting concerns in action

Amin Jalali
Supporting Enactment of Aspect Oriented Business Process Models

an approach to separate cross-cutting concerns in action

Amin Jalali
Abstract

Coping with complexity in Information Systems and Software Engineering is an important issue in both research and industry. One strategy to deal with this complexity is through a separation of concerns, which can result in reducing the complexity, improving the re-usability, and simplifying the evolution. Separation of concerns can be addressed through the Aspect Oriented paradigm. Although this paradigm has been well researched in the field of programming, it is still in a preliminary stage in the area of Business Process Management. While some efforts have been made to propose aspect orientation for business process modeling, it has not yet been investigated how these models should be implemented, configured, run, and adjusted. Such a gap has restrained the enactment of aspect orientated business process models in practice. Therefore, this research enables the enactment of such models to support the separation of cross-cutting concerns in the entire business process management life-cycle. It starts by defining the operational semantics for the Aspect Oriented extension of the Business Process Model and Notation. The semantics specifies how such models can be implemented and configured, which can be used as a blueprint to support the enactment of aspect oriented business process models. The semantics is implemented in the form of artifacts, which are then used in a banking case study to investigate the current modeling technique. This investigation revealed new requirements, which should be considered in aspect oriented modeling approaches. Thus, the current modeling notation has been extended to include new requirements. The extended notation has been formalized, and investigated through re-modeling the processes in the case study. The results from this investigation show the need to refine the separation rules to support the encapsulation of aspects based on different business process perspectives. Therefore, the new refinement is proposed, formalized, and implemented. The implementation is then used as a prototype to evaluate the result through a case study.

©Amin Jalali, Stockholm 2013

DSV Report Series No. 14-006
ISSN 1101-8526
Printed in Sweden by Stockholm University, Stockholm 2013
Distributor: Department of Computer and Systems Sciences, Stockholm University
This thesis is dedicated to my parents.
List of Papers

The following papers, referred to in the text by their Roman numerals, are included in this thesis.

PAPER I: **Service Oriented Modularization using Coloured Petri Nets**

PAPER II: **Operational Semantics of Aspects in Business Process Management**
DOI: [10.1007/978-3-642-33618-8_85](https://doi.org/10.1007/978-3-642-33618-8_85)

PAPER III: **Dynamic Weaving in Aspect Oriented Business Process Management**
DOI: [10.1007/978-3-642-41030-7_2](https://doi.org/10.1007/978-3-642-41030-7_2)

PAPER IV: **Aspect Oriented Business Process Modelling with Precedence**
DOI: [10.1007/978-3-642-33155-8_3](https://doi.org/10.1007/978-3-642-33155-8_3)

PAPER V: **Multi-Perspective Business Process Monitoring**
DOI: [10.1007/978-3-642-38484-4_15](https://doi.org/10.1007/978-3-642-38484-4_15)
I also contributed to this paper, which is not included in this thesis.

PAPER VI: Adaptive Case Management as a Process of Construction of and Movement in a State Space
Ilia Bider, Amin Jalali, Jens Ohlsson In 2d International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (ACM’13), Springer.
DOI: 10.1007/978-3-642-41033-8_22

Reprints were made with permission from the publishers.
# Contents

Abstract

List of Papers

Acknowledgements

1 Introduction
   1.1 Background ............................................. 13
   1.2 Research Goal ........................................... 18
   1.3 Research Question ...................................... 18
   1.4 Disposition .............................................. 19

2 Extended Background
   2.1 Aspect Oriented Programming .............................. 21
   2.2 Aspect Oriented service composition ...................... 24
   2.3 Aspect Orientation in Business Process Modeling ........ 25

3 Choice of Method

4 Method, Applications, and Results
   4.1 Operational Semantics ................................. 31
      4.1.1 Introduction ....................................... 33
      4.1.2 The Weaving Approach .............................. 33
      4.1.3 Formal Semantics .................................. 36
      4.1.4 Implementation ..................................... 38
      4.1.5 Verification ........................................ 38
   4.2 Aspect Oriented Business Process Modeling with Precedence 43
      4.2.1 Introduction ....................................... 43
      4.2.2 Construct ........................................... 43
      4.2.3 Evaluation ......................................... 46
   4.3 The Definition of a Pointcut .............................. 50
      4.3.1 Introduction ....................................... 50
5 Conclusion

References

Included Papers
Acknowledgements

I would like to express my gratitude to my co-supervisor Associate Professor Petia Wohed, without whom this work would never have started. I would also like to thank my supervisor Professor Paul Johannesson for his support and help during my study so far. I would like to thank Dr. Chun Ouyang for contributing and spending time to increase the quality of some of the papers. I would also like to thank Associate Professor Jelena Zdravkovic and my colleague Iyad Zikra for their valuable comments on early versions of this thesis. Last, but not least, I thank my parents who have always supported me.
1. Introduction

1.1 Background

Business Process Management (BPM) is an important area in Information Systems, which consists of the methods, techniques and tools to support four key activities known as modeling, enacting, managing and analyzing [1, 2] (see Figure 1.1). The activity of modeling deals with design issues of business processes [1]. The activity of enacting deals with the implementation, configuration and execution of process models [1], which are supported through systems called Workflow Management Systems (WfMS). These systems are also called Business Process Management Systems (BPMS). The process models should be precise and unambiguous in order to be implemented in a BPMS. Therefore, executable process models need to have operational semantics, which enables their implementation, configuration and execution. The activity of analyzing deals with analyzing a process model or its event logs [1]. The activity of managing deals with managing all other activities, such as adjusting a process to support exceptional and unforeseen situations during its enactment, or re-allocating resources to increase the efficiency of process execution [1].

The management of processes in BPMSs has a life-cycle, known as the BPM Life-Cycle [1] (see Figure 1.2). This life-cycle contains three phases: i) (re-)design; ii) implement/configure; and iii) run and adjust [1]. These phases

![Figure 1.1: BPM Activities, taken from [1]](image-url)
make a cyclical process which results in managing and improving business processes.

In the design phase (see Figure 1.2), business processes are modeled (fulfilling the modeling activity in Figure 1.1). There are different business process modeling techniques, such as Petri Nets [3], Coloured Petri Nets [4], Business Process Model and Notation [5], Unified Modeling Language [6], etc. Each model can also be analyzed in the design phase (fulfilling the analysis activity in Figure 1.1). This analysis is called a model-based analysis (see Figure 1.2) [1]. Business process modeling and analysis can be performed for different purposes. For example, process models can only be used to describe what is happening during the business process. The model can also be used to perform some model-based analysis to improve process models and hence business processes. Process models can be used to specify systems that should support business processes, etc.

Regardless of the purpose of designing a process model, a handmade process model can be categorized as descriptive, normative, or executable [1]. Descriptive models can describe desirable or undesirable situations in a business process, while normative models only describe desirable situations. An executable model is considered to be interpreted unambiguously by software, which enables the verification and enactment of the model. Models which are not executable lack formal definition and semantics, which opens up the possibility of subjective interpretation of the models. Different interpretations raise a lot of problems, such as reducing the interoperability of the processes [7].

In the implement/configure phase (see Figure 1.2), the process model is implemented in a BPMS. The implemented process can be configured to fulfill different requirements [9]. Process configuration enables the definition of a general process model that can contain several variations of a process model. Thus, the general process should be configured to be customized for each par-

![Figure 1.2: BPM-LifeCycle, taken from [1]](image-url)
ticular circumstance. In this way, the re-usability of a process model is improved \[9\]. The result of the implement and configure phase is a runnable process model. This phase enables the enactment of a process model, which can be considered as a part of the enactment activity in Figure [1.1].

In the run and adjust phase (see Figure [1.2]), the process model is executed. Process models are usually run over a long time period, and unexpected issues might happen during their lifetime. Thus, they should be adjusted at runtime to suit the requirements. In this phase, process models are enacted, which can be considered as a part of the enactment activity in Figure [1.1].

All these phases enable the management of business processes, so the enactment activity in Figure [1.1] is fulfilled through all of these phases. In managing business processes, there are many unforeseen situations which cannot be predicted at design time. In order to support managing processes, BPMS should support a degree of flexibility at both design time and runtime. Schoonenberg et al. \[8\] define four types of process flexibilities based upon two criteria: i) if the point in a process model that needs to be flexible is known at design time or runtime; and ii) if the solution to enable the needed flexibility is known at design time or at runtime (see Figure [1.3]).

If the point of flexibility, the solutions, and how the solutions should be applied, are completely known at design time, the solution can be captured within the process model. This means that the process definition is complete, and the flexibility is handled at design time. This sort of flexibility is called Flexibility by Design \[8\] (see Figure [1.3]). This kind of flexibility is capable of providing a solution for different situations that could happen in a process model. For example, what should be done if the payment has failed, or what sort of actions should be taken if there is no longer available a resource to

![Diagram](image_url)

**Figure 1.3:** Process flexibility types, taken from \[8\]

<table>
<thead>
<tr>
<th>Flexibility Configuration</th>
<th>Process Definition Completeness</th>
<th>Design-time</th>
<th>Run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underspecification (Late binding)</td>
<td>Partial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Underspecification (Late modeling)</td>
<td>Full</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


answer customers? This sort of flexibility is limited to the design phase of BPM life-cycle. This means that it cannot fulfill the goals of flexibility at runtime.

There are situations where the point of flexibility and the solutions are known at design time, but the decision regarding how the solutions should be applied is not known at design time. In these situations, process models can contain a placeholder to specify the point of flexibility, and the solution is bound to these points at runtime. This means that the process model is partially specified, the solutions are modeled at design time, but they are bound later. Thus, it is called Flexibility by Underspecification (Late binding) \[8\] (see Figure 1.3). This approach enables the re-usability of solutions in other process models as well, since they are not modeled in a particular process model. This sort of flexibility is limited to the design phase of BPM life-cycle, and it enables reusing solutions.

The other situation is when the point of flexibility is known at design time, but the solutions are not known at design time. Clearly, when the solution is not known, the knowledge of how to apply them is not available either. In these situations, the process models can contain a placeholder to specify the point of flexibility, and the solution can be modeled later (when the process model is enacted). This means that the process model is partially specified, and the solutions are modeled at runtime. Thus, it is called Flexibility by Underspecification (Late modeling) \[8\] (see Figure 1.3). This approach enables the re-usability of solutions in other process models as well, since they are not modeled in a particular process model. Moreover, it enables supporting process flexibility in the run/adjust phase of BPM life-cycle.

The last situation is when neither the point of flexibility nor the solutions are known at design time. In these situations, the process models cannot specify the flexible point, so their specifications are complete. Two approaches can be taken for such situations: i) change the process specification at runtime and add a solution to support the flexibility; ii) do not change the process specification, but enforce its execution to deviate from what is specified in the process specification \[8\]. The first approach is called Flexibility by Change, and the second approach is called Flexibility by Deviation \[8\] (see Figure 1.3). These approaches support process flexibility in the run/adjust phase of BPM life-cycle as well.

Different sorts of flexibilities support different solutions to model business processes. Indeed, in all of the flexibility types except the flexibility by design, the solution is separate from the process specification at design time. This means that each concern can be modeled separately. Separating the concerns can result in reducing complexity, improving re-usability, and simplifying evolution \[10\]. The way that the concerns are separated from the process models is called decomposition.
Most decomposition techniques suffer from separating all important concerns, since they usually have a so-called “dominant dimension of separation at a time” [10]. For example, case-driven business processes are defined based on the decomposition of different aspects of business processes, called perspectives, separately. These perspectives define different concerns of a process model [11]. However, the control-flow perspective is the dominant dimension in this approach, which defines the order of tasks in a business process. Such a dominant dimension disables the separation of concerns which are scattered and tangled in a system [10]. Scattering means the repetition of a concern in different process models, and tangling means the incorporation of more than one concern in a process model. Concerns which are scattered and tangled in a system are called cross-cutting concerns. Charfi et al. enumerate compliance, auditing, business monitoring, accounting, billing, authorization, privacy, and separation of duties as examples of these concerns [12].

As a real example in Swedish public organizations, it is compulsory to inform citizens whether a decision has been made on their applications. Accordingly, an inform activity is required in all business processes that contain a decide activity. Moreover, a process may contain several decide activities, implying the need for several inform activities. If the inform activity is changed, or if the policy regarding the informing concern is modified, we have to find and update all business processes containing a decide activity. To conform with the law, when designing a new business process, one has to remember to add the inform activity after each decide activity. Such efforts add costs to the design, updating and monitoring of business processes, and increase the risk of inconsistency in their process models. Moreover, concerns are tightly coupled with individual business processes and could not be reused. As a result, models of business processes have become more complex, less re-usable, and more costly to design and maintain [13].

The aspect-oriented paradigm addresses these problems by separating different concerns from the main process. Separation of cross-cutting concerns is investigated in terms of aspect orientation in different Information Systems disciplines such as programming and requirement engineering, where Aspect Oriented Programming (AOP) [14] and Aspect-Oriented Requirement Engineering (AORE) [15] approaches are introduced correspondingly.

In the BPM area, the separation of cross-cutting concerns is investigated in the design phase of the BPM life-cycle through Aspect Oriented Business Process Modeling, e.g. [12][16]. These modeling techniques do not have any formal definition and semantics, which makes different interpretations of process models feasible. Such a gap also limits these modeling techniques to the design phase of the BPM life-cycle. The investigation of how these techniques can be applied in other phases of the BPM life-cycle can enable the enactment
of aspect oriented business process models at runtime. Such an investigation can increase the re-usability of aspects in process models. It also facilitates the maintenance of the system, since if a concern changes, the corresponding aspect which encapsulates the concern should be changed. Moreover, it can enable the configuration of aspect oriented business process models to meet different requirements. It can also enable the adjustment of process models with different aspects when aspects are changed.

1.2 Research Goal

The goal of this thesis is to extend the support for separating cross-cutting concerns in the BPM area using aspect oriented principles. The extension is intended to be made through introducing and enabling aspect orientation in the phases implement/configure and run and adjust within the BPM life-cycle through designing and developing models and tools.

1.3 Research Question

This thesis aims to answer the question, “How can the separation of cross-cutting concerns be managed over the whole BPM life-cycle through aspect orientation?” To answer this question, three sub-questions are defined, which cover the separation of cross-cutting concerns in the whole BPM life-cycle. These questions are:

Question 1. How can an aspect oriented business process modeling technique be formally defined? The answer for this question removes the ambiguity in understanding these sorts of models. Thus, it enables interoperability between these models and also enables the verification, analysis and implementation of such models.

This question is answered in Paper IV.

Question 2. How can an aspect oriented business process model be implemented/configured? The answer for this question would enable the configuration of aspect oriented business process models for different situations. It also enables BPMS to interpret these models, which is required for running such models.

This question is answered in Papers I, II and III.

Question 3. How can aspect oriented business process models be run/adjusted? The answer for this question would enable running the implemented and configured aspect oriented business process models. It also
clarifies how this modularization technique facilitates applying changes in aspects and business process models.

This question is answered in Papers III and V.

1.4 Disposition

This thesis is organized as follows. Chapter 2 presents and discusses the extended background for aspect orientation in the areas of programming and BPM. It explains the basic terminology in the programming area, which is later adopted for the BPM area. It also explains the basic concepts in the BPM area using an example. Chapter 3 presents the choice of method. It briefly explains how the work is conducted. Chapter 4 summarizes the results which are reported in full details in the published papers. It starts by presenting the semantics for an operational semantics of aspect oriented business process models. Then, it defines the concept of formal definition for aspect oriented business process models. Finally, it defines the concept of formal definition for the definition of rules, and how a BPMS can be configured to capture those rules. Chapter 5 concludes the thesis and outlines future research.
2. Extended Background

This chapter shows how aspect orientation has evolved to support process modeling over time. It initiated in the programming area as the starting point for modularizing cross-cutting concerns in codes of systems. Then, this chapter will show how this modularization could be extended to capture the same sort of concerns in service composition. Afterwards, it will present the current state of aspect orientation in the Business Process Management area.

2.1 Aspect Oriented Programming

Aspect Oriented Programming has been considered as a solution to modularize cross-cutting concerns in programs [14]. It aims to solve the problem of code tangling and scattering. Code tangling refers to the situation where a program module contains more than one concern, and code scattering implies the repetition of one concern in different program modules. For example, a logging mechanism can be considered as a cross-cutting concern candidate, which is scattered in a system code. To change scattered code, a programmer should find all the pieces used and change them. Even if (s)he encapsulates them in a separate module, the changes in the condition of use enforce finding and updating all reference points. These problems increase the cost of maintenance, reduce the re-usability, and hinder the traceability of the concerns.

Aspect Oriented Programming (AOP) is a paradigm for addressing a complexity issue in the programming domain. This matter is performed by separating the core concern of an application from cross-cutting concerns of relevance, that has typically been captured through code scattered through the application program [17]. It also solves two problems of the traditional programming approach, that are described as follows.

Traditionally, different aspects of a program are written in different code modules (see Figure 2.1). Therefore, each module contains a different concern of the program, each of which handles one aspect, such as transaction management, security, or exception handling. Code tangling refers to the phenomenon when a module of code handles different concerns.

In the other method of implementing concerns in an application, cross-cutting concerns are implemented in separate modules, but each module should
call this functionality to implement relevant concerns (See Figure 2.2). Code scattering refers to the phenomenon of implementing the same cross-cutting concerns within different program modules or applications [17]. An example of code scattering is the implementation of security concerns for authentication and authorization in the Automated Teller Machine (ATM), Accounting and Database modules within an application (See Figure 2.2) [17].

There are some problems that arise by following these methods of programming. First, it is hard to apply a change to an aspect. As a case in point, if the transaction management needs a change, all codes that handle this concern should be found and changed all around the program. This means spending lots of resources and time to apply a change in a concern. Therefore, applying changes will be more costly. Second, if a piece of code within a module is missed by a programmer, the program will be inconsistent. Thus, these methods make the program less reliable and consistent. Third, pieces of code should be repeated in new modules if they should implement relevant concerns. Hence, these methods of programming do not support a good degree of re-usability, so less productivity will be achieved. Last but not least, it is impossible to trace a special concern if such a service is needed because concerns are spread throughout the code. As a consequence of code scattering and code tangling, the software possesses less traceability, re-usability, quality and

---

**Figure 2.1:** The Code Tangling Problem, taken from [17]

**Figure 2.2:** The Code Scattering Problem, taken from [17]
AOP proposes implementing concerns as individual program modules and linking them to the core concern, to solve the problems of code tangling and code scattering (See Figure 2.3). It also defines some basic terminology, listed below.

- **Join points**: Join points are defined as identifiable points in the application [17]. These points enable the addition of one or several cross-cutting concerns to a core concern. There are three different types of join points through which functionality can be added to a program module: field, method and type. These types are also called signatures.

- **Pointcut**: A language that enables the definition and selection of relevant join points in the application. As a case in point, a logging concern might be needed to be executed when more than 500 EUR is withdrawn from an account. Pointcut language supports the definition of this condition in which a special concern should be executed.

- **Advice**: A construct that enables adding or altering behavior in the core concern; these constructs are cross-cutting concerns like the security concern. Each advice could be executed before, after, or around the advised join point. ‘Advised join point’ designates the join point for which the advice should be executed.

- **Weaving**: A mechanism to alter the structure of a core concern using the execution of cross-cutting concerns. The weaving mechanism executes advices for advised join points which are determined by the pointcut language.

- **Aspect**: A module which contains a cross-cutting module is called an aspect. An aspect contains pointcuts and advices. In the example of the ATM, the security concern should be implemented in a module, called
the security aspect. This aspect contains a pointcut and an advice. The pointcut specifies the advised join points on which the advice should be executed. The advice specifies the security code that should be executed before, after, or around the advised join point. The type of advice is also specified in the advice specification.

Writing and executing programs are not the only issues that matter in the programming area. Discovering potential aspects from codes and from the program logs is also a matter of interest, which is called Aspect Mining. There has been a lot of research about Aspect Mining in the Aspect Oriented Programming area, e.g. [18]-[21]. This research can be divided into two categories: static and dynamic program analysis techniques.

Static program analysis techniques aim to discover aspects from the source code. Two approaches exist in these techniques: discovering advices without any previous knowledge or with previous knowledge about structure of advices. For example, a technique called the “Formal Concept Analysis of Identifiers” is used by some researchers to discover cross-cutting concerns from Smalltalk and Java code [22]-[23]. These techniques assume that the user has some understanding of the structure of advices. Other researchers have used “Natural Language Processing on Source Code” to solve this problem [24]. Furthermore, there are other works that discover cross-cutting concerns from code without any previous knowledge about the concerns [25]. The advantages of the static approaches are the fact that they consider all possible execution scenarios in the source code.

Dynamic program analysis techniques aim to discover aspects from program traces of the method invocations. There are also different works that have been done in this area. For example, discovering cross-cutting concerns by detecting recurring execution patterns [26], or applying formal concept analysis to achieve this goal [27]. There are a lot of works about aspect mining that have been performed in the AOP area [28]. It should be noted that for each approach, different algorithms can be developed.

2.2 Aspect Oriented service composition

The investigation of aspect orientation in the programming area revealed a new direction of research in service composition, where cross-cutting concerns could be separated from the service specifications. Thus, aspect orientation is also investigated in service decomposition to enable the separation of cross-cutting concerns. AO4BPEL [29]-[30] is one of those examples. AO4BPEL is an extension to the Business Process Execution Language (BPEL) to support aspect orientation. This extension is defined based on the soap message life-
cycle in which Charfi defines the order of advices as: “before → around → before soapmessageout → around soapmessageout → around soapmessagein → after soapmessagein → after” [31]. This implementation does not consider the life-cycle of activities in relation to human involvement, i.e., the BPEL4People activity life-cycle. Thus, the approach is limited to service decomposition. Such a limit prevents AO4BPEL from addressing the need to separate cross-cutting concerns in the BPM area. This need is even acknowledged by Charfi when he points out that “these security concerns will not be shown in BPEL code because BPEL does not support human participants. There is however, a recent proposal for such an extension” [31]. To consider the proposal (BPEL4People), the solution (AO4BPEL) should be changed to comply with the BPEL4People activity life-cycle.

Furthermore, AO4BPEL cannot be used to study the need for the separation of concerns for other business process perspectives, since BPEL does not support all business process perspectives [31]. Such a need can be exemplified in the situation where a senior employee in the bank shall confirm all activities of a newly employed clerk during the first week. This separation needs a definition of pointcuts to be specific for the resource perspective. Such a separation cannot be investigated by AO4BPEL since it is not developed based on BPEL4People activity life-cycles.

2.3 Aspect Orientation in Business Process Modeling

Aspect oriented business process modeling was inspired by the same trend in service composition, where the proposed approaches can be categorized into two categories in which the main process knows about cross-cutting concerns or not. It aims to separate different concerns from the main process. Concerns are captured in terms of aspects associated to a business process and thus can be handled outside the main process. There have been various papers (e.g., [12; 32]), which provide means for aspect oriented business process modeling. An aspect Oriented extension for Business Process Model and Notation (AO4BPMN) [12] is one such approach, which defines the terminology and suggests a notation based on BPMN for modeling processes according to the aspect oriented principle. This work was suggested by the result of applying aspect orientation to the service decomposition area.

To explain the basic terminology, let’s consider an example of a business process involving different concerns. Figure 2.4 describes a simplified version of a Transfer Money Process in the banking domain using BPMN [1]. First, a customer fills in the information. Next, if the money is transferred to

1For simplicity, we omit pools/lanes in this process model.
the customer’s own account, the transfer is performed straight away; otherwise, the transaction needs to be signed beforehand. Finally, the transaction is archived. The Sign Transaction activity is part of the security aspect, and the Archive Information activity is part of the logging aspect. These aspects describe different concerns related with the Transfer activity.

Figure 2.5 depicts the above process using AO4BPMN [12]. The concerns are removed from the main process and modeled separately through aspects. Hence, the main process contains only the Fill Information and Transfer activities. The two additional models annotated with an Aspect label are called aspect models. They capture the Logging Aspect and Security Aspect, respectively. An aspect consists of one or more advices, which are specified by individual process models annotated with an Advice label. An advice captures a specific part of a concern under a certain condition called a pointcut. A pointcut indicates when and where the advice should be integrated with the main process. The possible points of integrations are called join points. A join point can be related to an aspect via a pointcut, and in such a case, it is called an advised join point. In AO4BPMN, join points are activities. A pointcut condition, on the one hand, is captured in an annotation associated with an

Figure 2.4: Transfer Money Process model in BPMN

Figure 2.5: Transfer Money Process model in AO4BPMN
advised join point activity in the main process, and on the other hand, is specified in a data object as input to the corresponding advice process model. In Figure 2.5, the Transfer activity is an example of an advised join point with two pointcuts: one referring to the advice related to information archiving in the Logging Aspect, the other to the advice related to transaction signing in the Security Aspect.

Next, a PROCEED activity within an advice model acts as a “placeholder.” This placeholder is for carrying out the relevant advised join point activity during the execution of the advice. As such, the position of a PROCEED activity determines how the execution of an advice interleaves with the related advised join point activity. There are three scenarios: an advice occurring before, after or around an advised join point. For example, in Figure 2.5, the Archive Information activity occurs after the Transfer activity, while the Sign Transaction activity occurs before the Transfer activity. It is possible that an advice does not have a PROCEED activity. Such an advice is called an implicit advice, and is executed before the related advised join point activity [12].

Comparing the process models in Figure 2.4 and Figure 2.5 shows that aspect oriented process modeling increases re-usability because an aspect can be related to different activities and even different processes. It also makes the maintenance of process models easier, since any change to a concern would only affect the relevant aspect. Finally, it reduces the complexity of process models that capture the core processes.

There have been different approaches proposed for aspect oriented business process modeling, which are summarized below as they have progressed over time and contributed to each other.

Wang et al. tries to visualize the AO4BPEL to propose an approach for business process modeling [33]. A similar approach is also proposed by Shankardass [34], extending BPMN to support the separation of cross-cutting concerns. However, they introduced a new element to relate cross-cutting concerns to models, which requires design time knowledge about aspects when designing process models. Moreover, they ignored the importance of the role of people in the BPM area, so their work can be considered as a visualization approach for AO4BPEL.

The first modeling approach which considered the role of people was proposed by Cappelli et al. [32]. This proposal is defined further and investigated through a case study [16]. This approach proposes to modularize cross-cutting concerns in separate modules. However, it has no formal definition and semantics, which reduces the interoperability of the process models and makes different interpretations of models possible by different researchers [7]. For example, it is not clear how around scenarios should be modeled using this
approach, and there is no example of around advices to help readers to model them.

This gap is addressed by Charfi et al. by proposing Aspect Oriented for Business Process Model and Notation (AO4BPMN) [12]. This extension also lacks formal definitions and semantics. It also does not consider the precedence feature for aspects. The same approach (without considering precedence) is also followed and implemented by Collell [35]. However, both of these approaches require design time knowledge about the aspects when designing the process models, since they annotate the advised join point using intermediate conditional event.

Jabeen et al. also proposes a modeling approach for aspect oriented business process modeling [36], yet the proposed modeling technique considers start and end message events for modeling the start and end of an advice. Such a definition has no formal definitions. Moreover, if this approach is supposed to be interpreted using the BPMN specification, the main process should know about the cross-cutting concerns, since the advices should be invoked using message events. This makes the role of the aspects passive, which does not solve the tangling problem.

Patiniotakis et al. [37] extended the AO4BPMN [12] with two new notions for the Proceed placeholder, i.e., Replace and Bypass. The replace activity is defined to be considered instead of an advised join point, while the bypass activity ignores the advised join point. These two notions are introduced without considering the fact that several advices for an advised join point can run in parallel. Thus, the authors do not consider how such a proposal can affect other advices, and how the relation of the advices can ruin the orthogonal modularization in general. Indeed, by proposing these two activities, the aspect oriented business process modeling cannot be categorized in orthogonal modularization anymore, and extensive work is required to consider the relation between advices. This work missed considering these facts, and the result could provide a lot of ambiguity in aspect oriented business process modeling.
3. Choice of Method

Research within the field of Information Systems (IS) is mostly categorized into two paradigms, namely behavioral science and design science [38]. Behavioral science produces theories that enable the prediction of human behavior in organization [38][39]. Design Science tries to leverage the capabilities of people in organizations by developing artifacts [39]. Hevner et al. define a conceptual framework for IS research, which describes the role of these paradigms. This framework shows that every IS research tries to be both relevant and rigorous.

The relevancy of IS research is rooted in its problem definition. The problems are defined based on a business need. The need is defined based on the so-called environment, which consists of people, organizations and technology. The business need, “problem,” is perceived by the researcher. Behavioral science addresses the development and justification of a theory, but design science addresses the building and evaluating of an artifact. These two paradigms help the researcher to solve the problem through the cyclical assess and refine process.

The rigor of IS research is provided through what is called a knowledge base, which is composed of foundations and methodologies. The foundations contain the previous theories, frameworks, instruments, constructs, models, methods, and instantiations. “Methodologies” refers to Data Analysis Techniques, Formalisms, Measures, and Validation Criteria, which guide the research through the justification and evaluation phases. Justification and evaluation can be performed by different methods, including the case study and the field study [39].

The goal of design science is to develop an artifact [38]. The artifact can be a construct, model, method, or an instantiation [39]. To build an artifact, different research strategies can be employed, such as theory building, experimentation, observation, and system development [40]. This strategy follows a process with different activities, starting with “Constructing a Conceptual Framework” and followed by “Develop a System architecture,” “Analyze & Design the System of Building the (Prototype) System,” and “Observe & Evaluate the System” [40]. This process is a cyclical process in which each activity can be followed by the next one, or it can return to one of the previous activities to refine the result of the activity. To develop a system, the design science
3.5 Action Research Framework

Action research is an established research method in use in the social and medical sciences since the mid-twentieth century (Baskerville 1999). Action researchers are among those who assume that complex social systems cannot be reduced for meaningful study. The fundamental contention of the action researcher is that complex social processes can be studied best by introducing changes into these processes and observing the effects of these changes (Baskerville 1999).

In its origins, the essence of action research is a simple two-stage process:

- First, the diagnostic stage involves a collaborative analysis of the social situation by the researcher and the subjects of the research. Theories are formulated concerning the nature of the research domain.

This thesis follows the design science research framework, and it adopted this framework to answer each of the research questions differently. Thus, the applications of the method are discussed in each specific section. The contributions of this thesis are constructs and models, which enable the implementation, configuration, running and adjustment of aspect oriented business process models.
4. Method, Applications, and Results

This section gives a summary of the results, which are reported in detail in the included papers. The results are classified into three categories, each representing an artifact:

- **First Artifact**: A model for Operational Semantics of Dynamic Weaving in Aspect Oriented Business Process Management
- **Second Artifact**: Constructs and a Model for Aspect Oriented Business Process Modeling
- **Third Artifact**: Constructs and a Model for Multi-Perspective Business Process Monitoring

This section reports the result of each artifact, so it incorporates the Suggestion, Development and Evaluation steps of the Design Science Research Framework (see Figure 3.1).

The **first artifact** is a model that describes the semantics of dynamic weaving in aspect oriented business process management. The proposal is suggested in Sections 4.1.2, representing the Suggestion step. The proposed solution is developed using Coloured Petri Nets (CPN) [4], and is specified in Sections 4.1.3 and 4.1.4, representing the Development step. The developed artifact is verified through a state space analysis, which is a technique to check all possible states of the model [4]. The verification is a strong form of evaluation that proves the correctness of the model that was specified in Section 4.1.5. This section describes the result of the Evaluation step. This artifact answers the second and third questions of Section 1.3.

The **second artifact** introduces new constructs and a model to enable the definition of precedence in aspect oriented business process modeling. The proposal is made and developed through the definition of constructs and a model in Sections 4.2.2, representing the Suggestion and Development steps. The result is evaluated through a banking case study in Section 4.2.3, representing the Evaluation step. This artifact answers the first question in Section 1.3.
The *third artifact* introduces new constructs and a model to enable multi-perspective business process monitoring. The constructs and model are proposed and defined in Sections 4.3.2. They are also implemented as a prototype to enable the monitoring that is explained in Section 4.3.3. These sections represent the *Suggestion* and *Development* steps. The result is evaluated through a case study, which is described in Section 4.3.4. This section represents the *Evaluation* step. This artifact answers the third question in Section 1.3.
4.1 Operational Semantics

4.1.1 Introduction

To define how the implementation/configuration, running, and the adjustment of aspect oriented business process models can be performed, this section defines the operational semantics of the weaving mechanism. The work was initiated based on a literature review carried out for pattern work [42]. Previous research in the BPM area and other disciplines was studied to determine the functionalities needed for a weaving mechanism. Afterwards, a design was proposed for the operational semantics of weaving in the BPM area. The proposal was made using the definition of a model in Coloured Petri Nets (CPN) [4], which is a formal language. Being formal means having a mathematical definition of the syntax and semantics, which enables the verification of systems built based on this language. Therefore, the design was verified in terms of soundness, which can be considered as a construct to act as a blueprint for implementing a weaver. The nets are also implemented as a service in YAWL [43]. YAWL was selected since it i) is an open-source workflow management system; ii) is based on SOA and is hence extensible; and iii) provides extensive support for workflow patterns [43; 44] In addition to this service, a rule definition application was also developed to define the connections between cross-cutting concerns and process models. These systems enable the implementation, configuration, running, and adjustment of aspect oriented process models that support both the Implement/Configure and the Run and Adjust phases of the BPM life-cycle. A summary of this approach is given here (full details can be found in Papers I, II, and III).

4.1.2 The Weaving Approach

In this section, a generic solution in the form of a so-called Aspect Service is proposed, which extends current BPMSs with runtime support to aspect oriented modularization.

According to the principle of weaving, the aspects and their advices are to be executed together with the main process, and synchronizations are to be made at the Proceed placeholder as well as at the end of each advice. To explain these requirements of weaving in more detail, an abstract example of an aspect oriented process model is used, which is shown in Figure 4.1. There is a main process with four aspects, which are associated to one of the activities (activity B) in the process, and each aspect contains a single advice. Based on the fact that aspect Y has an after advice, aspect Z has a before advice, aspect X has an around advice, and aspect W has an implicit advice, it can be determined when activity B should take place together with the four advices.
Figure 4.1: An abstract example of an aspect oriented process model

Moreover, activity C, which is the activity that follows activity B in the main process, cannot occur until the executions of all the advices associated with activity B are completed. Hence, the valid execution sequence of activities in the process in Figure 4.1 will be A, followed by D, G and H in parallel, then B, followed by E and F in parallel, and finally C. Using regular expression form, this can be written as $A(D|G|H)B(E|F)C$.

At runtime, the enactment of business processes, including executions of activities in the main process and those in the associated aspects, is managed through a BPMS. The Aspect Service controls and coordinates the interactions between (the advices in) the aspects and the main process. We propose a generic approach to support the dynamic weaving of the aspects during the process enactment. It consists of the following four steps. Note that for an implicit advice, a pre-processing is required, which adds an (empty) Proceed placeholder to the end of the advice, and as such, an implicit advice is treated as before advice during dynamic weaving.

**Launching:** before executing an advised join point, the Aspect Service shall launch all valid advices associated with that join point. Each valid advice is determined by the Aspect Service through the evaluation of the corresponding pointcut condition. If the pointcut condition holds, the Aspect Service will initiate one instance for that valid advice.

**Pausing:** the Aspect Service shall pause the execution of an advice when reaching the Proceed activity in the advice and at the same time enable the execution of the corresponding advised join point in the main process. It is possible that multiple advices exist for one advised join point, in which case, the Proceed activities in these advices should be syn-
chronized to ensure a single execution of the advised join point.

**Resuming:** when the execution of the advised join point is completed, the Aspect Service shall resume the enactment of those advice instances that are interrupted by the execution of the advised join point. Note that for each launched advice (regardless of whether it is a before, after, or around advice), the control of execution will be returned to the corresponding advice instance after the enactment of the advised join point.

**Finalizing:** the Aspect Service shall complete the executions of all the launched advice instances before the enactment of the main process can continue. Only when the executions of all the launched advice instances finish will the control be returned to the main process to continue its enactment (to subsequent activities enabled after the advised join point). This signals the end of the weaving of aspects associated with that advised join point.

Figure 4.2 illustrates the dynamic weaving of the aspects with the main process using the example shown in Figure 4.1. To reflect the runtime nature of weaving, the notation of YAWL is used. YAWL is based on Petri nets but extended with advanced control-flow constructs to facilitate workflow modeling. In the left-hand side of Figure 4.2, there are four YAWL nets capturing the main process and the three advice processes, respectively. Between each advice net and the main net, there are highlighted annotations capturing the above four-phased weaving approach. For illustrative purposes, the overall behavior resulting from the weaving of three aspects with the main process of the example in Figure 4.1 is specified as the YAWL net in the right-hand side of Figure 4.2. Tasks that belong to the same advice share the same graphical annotation.

![Figure 4.2: Dynamic weaving of aspects using the example in Figure 4.1](image-url)
In the next section, the CPN model that shows the operational semantics of the weaving at runtime are described briefly.

4.1.3 Formal Semantics

The formalization of the Aspect Service is specified using hierarchical Coloured Petri Nets. The solution is a three-level model. The top-level module captures the behavior of the initiation of the service (see Figure 4.3). The second level captures the weaving behavior (see Figure 4.4). This model contains four modules capturing the requirements related to the weaving described in the previous section. It also contains a module for communicating with the BPMS and performing actions for data persistence, which is needed for the weaving. These five modules constitute the third level of the CPN Model. The model defines 57 color sets and 33 functions. We re-used some of the colour sets, variables and functions from the Worklet Service CPN model [45]. In that version, the semantic was not developed based on a workitem life-cycle; but this version is refined to be compatible with the life-cycle. This compatibility makes the semantic general for all workflow management systems.

The interaction of the Aspect Service and a BPMS is realized through passing a number of messages. These messages are called constraints and commands. **Constraints** are the messages raised by the BPMS, and **Commands** are the messages invoked by the Aspect Service.

We used standard constraints and messages according to the BPMS reference model defined by the Workflow Management Coalition (WfMC). Hence, the CPN model is shown in Figure 4.3.

![Figure 4.3: CPN: Aspect Service](image-url)
the solution is general and can be adapted to any BPMS. The constraint messages are WorkitemConstraint and CaseConstraint. The raising of one of these messages is signified in the CPN model in Figure 4.3 as a token arriving in the workitemConstraint or caseConstraint places (the places are highlighted in the figure). In other words, these places are the starting points of the net.

As can be seen in Figure 4.3, the service recognizes whether the WorkitemConstraint is related to a normal activity or to a Proceed activity (see the matchPointcut and isProceedCmd transitions). If it is a normal activity, the matchPointcut investigates whether a pointcut is defined for the activity or not. If yes, the pointcut is evaluated to see which advices should be launched using the selectPointcut subnet. If the pointcut is fulfilled, the enableAspect transition is enabled, else the notFulfilled transition is enabled. The result leads to Launching if the advised join point is just started (the enableAspect transition produces a token in the AspectInfo place). However, if the advised join point is completed, the Resuming should be started (the enableAspect transition produces a token in the AdvisedJP place). If it is a Proceed activity, no Pointcut needs to be checked (the isProceedCmd transition produces a token in the Proceed place). As a result, the Pausing can be started. The Finalizing can be started if all advices are finished. This condition is checked using the endAdvice transition, which produces a token in the completedAdvice place if a token representing the end of a launched advice appears in caseConstraint.

The net in Figure 4.3 shows how messages received from the BPMS should be processed to enable weaving. The weaving is described by the weaveAspect sub-net shown in Fig 4.4. The weaveAspect net contains five subnets:
Launching, Pausing, Resuming, Finalizing and Core. The first four sub-nets fulfills the four weaving requirements, and the last one persists the data required to perform the weaving.

The CPN model allowed us to verify the design of the Aspect Service using a state space analysis. This analysis showed that the nets were free of deadlocks. The implementation of the operational semantics is presented in the next section.

4.1.4 Implementation

Using the formal CPN specification of our proposed solution for dynamic weaving of aspects for process enactment, two artifacts are implemented, a Pointcut Editor and an Aspect Service\footnote{The artifacts, with examples and case studies, can be downloaded from http://www.aobpm.com} in the framework of the YAWL system\footnote{An open source BPMS, see http://www.yawlfoundation.org}. We chose YAWL because: 1) it provides support for the full workitem life cycle; 2) it has a formal foundation; and 3) it is open-source and based on the Service Oriented Architecture [43,44].

The Pointcut Editor (see a screen-shot of the GUI in Figure\ref{fig:4.5}(a)) enables the definition of aspects, advices and pointcuts. Each aspect can have several advices, and each advice can have several pointcuts. These pointcuts consist of the names of the process and the activity with which the advice should be weaved. The pointcut also includes a condition. The condition is used to define the data constraints that control the enactment of an advice (e.g., transfer to non own account). If the condition is fulfilled, the advice will be weaved. The condition can be written using the XPath language.

Figure\ref{fig:4.5}(b) shows the architecture of the Aspect Service and its relation to the BPMS. The service is connected to the YAWL Engine through two interfaces: B and X. Interfaces X and B are used to capture case and workitem level events, respectively. The service also reads the rules (composed by the Pointcut Editor) from the Rule Repository. The rules specify which events should be captured. During implementation, the example in Figure\ref{fig:4.1} was used for testing, since it contains all combinations of advice types.

4.1.5 Verification

The functionality of a CPN Model can be investigated using simulations or a state space analysis. There are two types of simulations: interactive and automatic. Interactive simulations facilitate the investigation of the functionality of a system using a step by step analysis of the model. Automatic simulations
Figure 4.5: Implemented artefacts to enable dynamic weaving
are typically used to analyze the performance of the system [4] and are not relevant for our work. During the design, interactive simulations are used in order to continually test the behavior of the model and build it iteratively.

It should be noted that a simulation does not investigate all possible states of the system and hence cannot be used for verifying the behavior of a model. All possible reachable states of a system are investigated through a state space analysis [4], which is also the analysis technique we used for verifying the nets. We carried out state space analyses on a number of scenarios.

The scenarios were designed to capture all possible behaviors that can be encountered by the Aspect Service. We started with looking at the interplay between the number of aspects and advices in a model. Because the weaving is done using the main process and related advices, the number of advices influences the behavior of a model, while the aspects alone do not influence it. As mentioned earlier, there are two types of advices: implicit and explicit; both need to be considered. Hence, four combinations of them can be derived: i) scenarios which contain implicit advices; ii) scenarios which contain implicit and explicit advices; iii) scenarios which contain explicit advices; and iv) scenarios in which the main process is not altered by any advice, i.e., either the advices are not considered since the pointcuts are not fulfilled, or there are no advices have been defined.

To verify the behavior of the model, the synchronization of the advices is essential. To study the synchronization, at least two advices are needed. However, the process to synchronize the advices to the main process is not changed by adding more advices to an advised join point. Thus, two advices are both necessary and sufficient to verify the model.

As a result, the following four scenarios are defined: Scenario A, containing two explicit advices; Scenario B, including one explicit and one implicit advice; Scenario C, containing two implicit advices; and Scenario D, describing the situation in which the pointcut condition is not fulfilled, so the Aspect Service should not alter the execution of the main business process (see the left columns in Figure 4.6).

The Aspect Service interacts with a WfMS. This interaction is captured through sets of tokens representing the raising of constraints by the WfMS. The raising of the constraints follows a specific order. Therefore, a number of transitions and places are added to the model for each scenario, to force the model to be executed according to the defined order. The results from the state space analyses for the four scenarios are shown in Figure 4.6.

As can be seen in the figure, the values for the Home Marking and the Dead Marking are the same for each scenario. This shows that the model terminates in a specific state [4]. The state space reports show that there are no live transition instances and no infinite occurrence sequences. Hence, the
Figure 4.6: State space report for the four scenarios

* No Live Transition Instance for all nets for all steps.
* No infinite occurrence sequences for all nets for all steps.

The model is sound. Furthermore, there are some Dead Transition Instances for each scenario (not captured in the figure). Dead Transition Instances indicate parts of a model that are redundant and can be removed. However, the intersection of Dead Transition Instances is empty, i.e., $D_A \cap D_B \cap D_C \cap D_D = \emptyset$, where $D_X$ denotes the set of Dead Transition Instances of scenario $X$. This means that there are no transitions in the model which are not used by any scenario. Therefore, the model does not have any redundancy.

Finally, a minor comment is that the number of nodes and arcs in the state space and the Strongly Connected Component (SCC) graphs are equal for each scenario. A Strongly Connected Component (SCC) refers to a set of nodes which are mutually reachable (i.e., the nodes that can be reached from each
other) and reveals loops in a state space graph. The nodes which are not mutually reachable to other nodes (including themselves) are called \textit{trivial Strongly Connected Components} (trivial SCCs). Since the number of nodes and arcs in the SCC graphs are equal to the number of nodes and arcs in the state space graph, all nodes in the state space graph are trivial SCCs, for each scenario. Therefore, \textit{there are no cycles in the CPN Model}, which is to be expected, since the weaving process does not contain any cycles.

\textbf{Figure 4.7:} Part of the state space graph for Scenario A

We validated the model’s fulfillment of the specified requirements through observations in the state space graphs. For instance, requirement R1.1 is fulfilled because, independently of the execution path, the advised join point is always suspended. As depicted in Figure 4.7, one of the states 29, 30 and 32 (see the bold states in the figure) is reached during the execution of the net. These states contain a token with a \texttt{ackSuspendWorkItem} command, which means that the advised join point is suspended. Similar observations were made for all the state space graphs of all four scenarios.

Finally, an aspect oriented business process model is sound if the main process model and related advice models are sound. The soundness was proved because the woven process is a composition of the main process and the models of the advices by using pairs of AND-splits and AND-joins. Therefore, the weaving process always results in a sound model.
4.2 Aspect Oriented Business Process Modeling with Precedence

4.2.1 Introduction

The enactment of aspect oriented business process models reveals a new requirement that should be considered when modeling the separation of cross-cutting concerns. This requirement is the precedence of the aspects, which defines their order in relation to elements in the process models. In this study, a new modeling approach is proposed which is an extension to AO4BPMN [12]. The extension is defined through formal definition. It is then evaluated through a banking case study. The evaluation shows how the precedence improves the previous approach in a such a way that all aspects can be separated (the full details can be found in Paper IV).

4.2.2 Construct

The formalization of the syntax of BPMN builds on the previous syntax definition (based on BPMN1.0) in [46] and extends it with the data and resource information and elaboration on events and exception constructs according to BPMN2.0 [5].

**Standalone BPMN Process** A standalone BPMN process is a tuple


where

- \( O \) is a set of objects which can be partitioned into disjoint sets of activities \( A \), events \( E \), and gateways \( G \),
- \( A \) can be partitioned into disjoint sets of atomic activities (i.e. tasks) \( A^T \) and compound activities (i.e. subprocesses) \( A^S \),
- \( E \) can be partitioned into disjoint sets of start event \( E^S \), intermediate events \( E^I \), and end event \( E^E \),
- \( G \) can be partitioned into disjoint sets of parallel gateways \( G^A \), data-based exclusive decision gateways \( G^X \), event-based decision gateways \( G^E \), and exclusive merge gateways \( G^M \),
- \( F \subseteq O \times O \) is the control flow relation, i.e., a set of sequence flows connecting objects,
- \( D \) is a set of data objects associated with the process,
- \( L \) is a set of data object names,
• \( R \) is a set of roles designated to perform tasks or events within the process,
• \( W \) is a set of organization groups involved in carrying out the process,
• \( EN = \{ \text{message, timer, error, conditional} \} \) is a set of basic event type names,
• \( Etype : \mathcal{E} \rightarrow EN \) is a function which assigns to each event an event type,
• \(Attach : \mathcal{E}^I \rightarrow A\) is a function\(^1\) which attaches an intermediate event to an activity, in this way indicating that the event might occur during the execution of that activity,
• \(\text{Excp} : \text{dom}(Attach) \rightarrow B\) is a function\(^2\) which specifies, for an intermediate event that is attached to an activity, whether or not its occurrence interrupts the (normal flow of) activity execution,
• \(Econd : \{ e \in \mathcal{E} | Etype(e) = \text{conditional} \} \rightarrow C\) is a function\(^3\) which assigns to each conditional event a condition specified as a Boolean function,
• \(Fcond : \mathcal{F} \cap (\mathcal{G}^X \times O) \rightarrow C\) is a function which maps the sequence flows emanating from data-based exclusive decision gateways to conditions, determining in this way whether the associated sequence flow is taken during the process execution,
• \(Dflw : D \rightarrow (A \cup \mathcal{E}^S \cup \mathcal{E}^I) \times (A \cup \mathcal{E}^I \cup \mathcal{E}^E)\) is a function which specifies that a data object is transferred from one activity/event to another,
• \(Act : (A^T \cup \mathcal{E}) \rightarrow 2^R\) is a function which designates one or multiple roles eligible to perform a task or event,
• \(Belto : D \rightarrow W\) is a function which assigns a role to an organization group,
• \(ADlab : A \cup D \rightarrow L\) is a function which labels each activity or data object, and
• \(Elab : \mathcal{E} \rightarrow L\) is a function which labels an event (without mandating that each event is labeled).

\(^1\)\( \rightarrow \) indicates a ‘non total function,’ i.e., there are values in the domain that do not have a corresponding value in the range.
\(^2\)\( B \) is the Boolean set \{true, false\}.
\(^3\)\( C \) is the set of all possible conditions. A condition is a Boolean function operating over a set of propositional variables that takes the values true or false.
For a standalone BPMN process \( M \), if ambiguity is possible, \( M \) is used as a subscript to each element defined in the tuple \( M \). For example, \( A^S_M \) refers to the set of subprocess invocation activities in \( M \). Next, the syntax of a core BPMN process is defined. It supports a hierarchical structure comprising a set of standalone BPMN processes.

**Core BPMN Process** A core BPMN process is a tuple \( P = (Q, M^{top}, S^\circ, map, HR) \), where

- \( Q \) is a set of standalone BPMN processes,
- \( M^{top} \in Q \) is the top level process,
- \( S^\circ = \bigcup_{M \in Q} A^S_M \) is the set of all subprocess invocation activities in \( Q \),
- \( map : S^\circ \to Q \setminus \{M^{top}\} \) is a function which maps each subprocess invocation activity to a standalone BPMN process, and
- \( HR = \{(M, M') \in Q \times Q \mid \exists s \in A^S_M \text{ map}(s) = M' \} \) is a connected graph.

Next, an *advice* process is a BPMN process in which all the start events of the (top-level) process are conditional events and there may be one or more \textit{PROCEED} activities, and an *aspect* process comprises a number of advice processes that belong to the same aspect.

**Advice Process** An advice process \( P^a = (Q, M^{top}, S^\circ, map, HR, A^{Tp}) \) is a core BPMN process that satisfies the following conditions:

- \( \forall e \in E^S_{M^{top}}, Etype(e) = \text{conditional}, \) i.e. all start events in the top level process are conditional events, and
- \( A^{Tp} = \bigcup_{M \in Q} \{a \in A^T_M \mid ADlab_M(a) = \text{PROCEED} \land a \notin \text{ran}(Attach_M) \} \), where \textit{PROCEED} is a preserved label for \textit{PROCEED} activities.

**Aspect Process** An aspect process is a tuple \( P^A = (\{P^a_1, P^a_2, \ldots, P^a_n\}, AN, Advice) \), where

- \( \{P^a_1, \ldots, P^a_n\} \) is a set of advice processes,
- \( AN \) is a set of advice names, and
- \( Advice : \{P^a_1, \ldots, P^a_n\} \to AN \) is a bijective function which assigns to each advice process a unique advice name.
Finally, an AOBPMN process comprises a main BPMN process and a set of associated aspect processes. The interactions between the main process and the aspect processes, which are defined in the pointcut specifications, are carried out at the corresponding advised join point activities.

Core AOBPMN Process A core AOBPMN process is a tuple \( A\mathcal{P} = (\mathcal{P}, \mathcal{E}_A, \mathcal{A}_{JP}, \{\mathcal{P}^A_1, \mathcal{P}^A_2, \ldots, \mathcal{P}^A_n\}, \mathcal{CN}, \text{Aspect}, \text{Pointcut}) \), where

- \( \mathcal{P} = (\mathcal{Q}, \mathcal{M}^{top}, \mathcal{S}^\circ, \text{map}, \mathcal{HR}) \) is a core BPMN process,
- \( \mathcal{E}_A = \bigcup_{M \in \mathcal{Q}} \{ e \in \text{dom}(\text{Attach}_M) \mid \text{Etype}_M(e) = \text{conditional} \land \text{Excp}_M(e) \} \) is the set of intermediate conditional events attached to an activity in \( \mathcal{P} \),
- \( \mathcal{A}_{JP} = \bigcup_{M \in \mathcal{Q}} \{ a \in \mathcal{A}_M \mid \exists e \in \mathcal{E}_A \text{Attach}_M(e) = a \} \) is a set of advised join point activities in \( \mathcal{P} \),
- \( \{\mathcal{P}^A_1, \ldots, \mathcal{P}^A_n\} \) is a set of aspect processes,
- \( \mathcal{CN} \) is a set of aspect names,
- \( \text{Aspect} : \{\mathcal{P}^A_1, \ldots, \mathcal{P}^A_n\} \rightarrow \mathcal{CN} \) is a bijective function which assigns to each aspect process a unique aspect name,
- \( \text{Pointcut} : \mathcal{E}_A \rightarrow 2^{\text{Expr}} \), where \( \text{Expr} = \{ \langle \text{cond}, \text{pos}, \text{cn}, \text{an}, \text{order} \rangle \mid \text{cond} \in C \land \text{pos} \in \{\text{before}, \text{after}, \text{around}\} \land \text{cn} \in \mathcal{CN} \land \text{an} \in \mathcal{AN}_{\text{Aspect}^{-1}(\text{cn})} \land \text{order} \in \mathbb{Z}^+ \} \), is a function which relates an event \( e \in \mathcal{E}_A \) to a set of expressions, and each expression specifies for the corresponding advised join point \( a \in \mathcal{A}_{JP} \):
  - the condition capturing the constraints for triggering an advice (\( \text{cond} \)),
  - when the advice should be triggered in relation to \( a \) (\( \text{pos} \)),
  - the aspect name (\( \text{cn} \)) and the advice name (\( \text{an} \)), and
  - the precedence order of triggering this advice among the multiple advices associated with \( a \) (\( \text{order} \)).

4.2.3 Evaluation

In this section, the proposed approach is applied to a case study from the financial domain. The case study demonstrates how the approach can be applied to modularize cross-cutting concerns in a banking process model. In particular, the case confirms the relevancy of the precedence requirement.
This banking case was selected due to previous knowledge in that domain. To choose the appropriate processes, i.e., fairly simple yet representative processes with at least a couple of cross-cutting concerns, an interview was conducted with a domain expert from a bank. For reasons of confidentiality, the bank asked to remain anonymous. Two processes were selected. Here, one of them, namely the Change asset deal process, is presented. Detailed information about the process was derived through a follow-up interview with the same domain expert.

Generally, the assets of the bank are in two forms, cash and non-cash. Cash assets are either in the form of the account balances of the bank or in marketable securities. The Change asset deal process (see Figure 4.8) handles deals for exchanging assets of the bank from one currency to another. The process starts with a back office employee filling in a position sheet (Fill position sheet activity). Then, the general manager either confirms or denies the deal. If the sheet is denied, the process ends. If the position sheet is approved, it is archived. Then, a junior dealer makes the deal and fills in a deal slip. Next, both a chief dealer and the general manager sign the deal slip, after which the deal slip is archived.

After the deal slip has been archived, two parallel sets of activities are performed. On the one hand, the amount of money dealt is sent to the external partner of the deal. For this, first an employee of the Swift department provides a swift draft for sending the money. Then, for security purposes, the dealer, chief dealer and general manager sign the swift draft. Finally, an employee of the Swift department sends out the swift. On the other hand, the amount of money dealt should be received. This part starts when an employee of the Swift department receives an NT300 swift message. The employee sends this message to the general manager. The general manager issues an order to the Back office department and to the dealer to control the swift message. These orders are issued separately. When each one of them has been controlled, the messages are archived (separately). When the deal is made, a back office employee registers a voucher in the accounting system. Finally, the deal is archived.

Figure 4.8 shows the models including both BPMN and AOBPMN versions of the change asset deal process. We distinguish the following results of applying the aspect oriented modularization approach:

- The aspect oriented solution documents additional knowledge of a business processes in the model. This knowledge specifies the relation between the cross-cutting concerns and the specific activities. For example, in Figure 4.8a, two security concerns are associated to the Send Swift activity while this knowledge is missed in the model in Figure 4.8b. I.e., it is not obvious to which of the two activities, Provide
Figure 4.8: The case study process: (a) traditional modeling; (b) AOBPMN modelling
Swift Draft or Send Swift, the security concerns are related.

- The aspect oriented approach presented here enables the separation of several concerns. For instance in Figure 4.8b, two different aspects are associated to the Fill DealSlip activity. In this way, security policy makers could easily define and change their related policy without changing the main process or the archiving concerns.

- This approach enables the separation of concerns which have different orders for consideration. For example, a deal slip should be first confirmed and then archived (see Fill DealSlip activity). Other approaches [12, 16] are not able to separate the archive concern in this example, because they do not capture the precedence requirement in the definitions of the advices. Therefore, they could not separate all the concerns from a business process. In contrast, our approach supports a full degree of separation.

- The aspect oriented model in Figure 4.8b is less complex, in terms of the number of activities, than the model in Figure 4.8a. While the model in Figure 4.8a contains 20 activities, the model in Figure 4.8b contains 10 activities in the main process and six in the advice processes. Hence, communicating the aspect oriented model to business users is expected to be easier [47].

- Aspect orientation increases the re-usability since policies are defined once and can be used many times. See, for instance, the use of the Confirm advice in Figure 4.8b, where it is associated both with Fill DealSlip and Send Swift in the core business process.

- It also facilitates the maintenance of the system. If a policy is changed, the changes can be applied to one model rather than to all the business processes involved. E.g., if the Confirm concern is changed, the updates are reflected in the corresponding advice in Figure 4.8b rather than in a number of places in a process or, even worse, in several processes (Figure 4.8a).

- Last, but not least, aspect oriented modeling enables agile development of business processes, due to its faster response to changes, better adaptability, and flexibility [48, 49]. This enables incremental development of business processes, i.e., the ability to add or change aspects even some time after the development of the main process.
4.3 The Definition of a Pointcut

4.3.1 Introduction

The evaluation of aspect oriented business process modeling with precedence shows that the pointcuts should enable the selection of join points based on information from different perspectives. Thus, this section proposes an approach to refine the pointcut definition to enable such selection. To do that, this section considers the join points as points which should be monitored, called monitoring points. Moreover, the pointcuts which incorporate the rules are called monitoring rules. Therefore, first a formal definition of monitoring points and rules are given, then the algorithm to monitor such rules is specified. Two artifacts are also implemented to support the definitions of the monitoring rules and the observation of the process models (the full details can be found in Paper V).

4.3.2 Formal Definition

Basic Definition

- \( \mathcal{P} = \{ \text{Control-flow, Data, Task, Resource} \} \) is a set of Perspectives. Here, the approach is limited to four perspectives, but they can be added just as a member of the set.

- \( \mathcal{L} = \{ \text{Case, Workitem} \} \) is a set of Levels. There are two levels for monitoring: Case and Workitem. Case represents the executed instance of a process model. The workitem is an executed instance of a task.

- \( \mathcal{W}_{en} = \{ s:enable, s:create, s:start\_on\_create, s:offer\_m, R:start\_s, R:allocate\_s, R:allocate\_m, R:start\_m, R:suspend, R:resume, R:fail, R:complete, \} \) is a set of WorkItem Event Names. These names are derived from the Workitem life cycle. We also added \( s:enable \) to monitor the workitem when it gets enabled.

- \( \mathcal{C}_{en} = \{ \text{create, suspend, resume, complete, fail} \} \) is a set of Case Event Names.

- \( \mathcal{E}_{n} = \mathcal{C}_{en} \cup \mathcal{W}_{en} \) is a set of Event Names, which is a union of case event names and workitem event names.

- \( \mathcal{E}_{d} = (\mathcal{P}, \text{Value}) \) is the EventData, which is a tuple. It contains a perspective and its values.

  Value is a simple string. This string can contain, for example, xml representing the data perspective.
• $\mathcal{E}_{ds} = \{\mathcal{E}_d\}$ is a set of Event Data.

• $\mathcal{C} = \mathcal{E}_d$ is Condition. The condition is an EventData, which is a tuple containing a perspective and its values. We distinguish between event data and a condition because event data is what happened in execution, but a condition is an abstract representation of the situation that should be monitored.

• $\mathcal{C}_s = \{\mathcal{C}\}$ is a set of conditions.

• $\mathcal{E} = (\mathcal{E}_n, \mathcal{E}_{ds})$ is Event. The event is a tuple. It includes an event name and a set of event data. In this way, each event can carry different data from different perspectives.

Definition of the Monitoring Rules

• $\mathcal{M} = (\mathcal{L}, \mathcal{E}_n)$ is MonitoringPoint. It is a tuple, which contains a level and an event name. This means that a monitoring point can be any event in case or workitem level.

• $\mathcal{M}_s = \{\mathcal{M}\}$ is a set of Monitoring Points.

• $\mathcal{R} = (\mathcal{M}, \mathcal{C}_s)$ is Rule, which is a tuple. It contains a monitoring point and a set of conditions. This means that a rule defines the criteria that capture the monitoring points, which can be limited from different perspectives.

• $\mathcal{R}_s = \{\mathcal{R}\}$ is a set of Rules (Ruleset).

“Algorithm 1” shows how events can be examined to see whether the conditions in the ruleset are satisfied. The algorithm gets the event, level and the ruleset. It checks the rules based on the conditions specified in the ruleset, and returns the set of rules that are satisfied. The condition can have ‘*’ as a value, which means that all values can be accepted. This algorithm is not designed for any specific perspective, so it is general. As a result, by adding any perspective to the set of perspectives, the algorithm will not be changed. This algorithm is implemented in the Observer service, which is described in the next Section.

4.3.3 Implementation

To enable the definition of rules in a way that supports all combinations, a rule editor and an Observer service are implemented.  

1Both the rule editor and the Observer Service can be downloaded from http://people.dsv.su.se/~aj/ObserverService/
Algorithm 1 Evaluate Monitoring Rules

Input: \( l: \mathcal{L}, e: \mathcal{E}, rs: \mathcal{R}_s \)

Output: \( \mathcal{R}_s \)

1: \( \mathcal{R}_s \) result;
2: for each \( \mathcal{R} \) r in rs do
3:    if \( r.\mathcal{M}=(l, e) \) then
4:       Boolean ruleResult := true;
5:       for each \( \mathcal{C} \) c in r.\( \mathcal{C}_s \) do
6:          if ruleResult=true AND c.value<>''*' then
7:             for each \( \mathcal{E}_d \) ed in e.\( \mathcal{E}_d \) do
8:                if ed.\( \mathcal{P} \)=c.\( \mathcal{P} \) AND ed.Value<>c.Value then
9:                   ruleResult := false;
10:              end if
11:          end for
12:       end if
13:     end for
14:    if ruleResult=true then
15:       result.Add(r);
16:    end if
17: end if
18: end for
19: return result;
The Rule Editor

The rule editor is designed in a very generic way that can be extended easily to support other states and perspectives. It reads the perspectives and states from an XML file. The XML indicates what information exists in each event. The result is shown to the user when s(he) wants to define a monitoring rule (see Figure 4.9(b)). For each state, the editor shows what perspectives are available to be limited by monitoring rules. For example, Data_Workitem and Resource are not available for workitem enabled monitoring points. This awareness supports users in defining rules that comply with the context of the events.

The user can limit the data for each perspective in the editor (see Figure 4.9(a)). To do that, the user should select the level (workitem or case) and the state for which s(he) wants to observe the process. The editor enables the user to apply some limitation to the monitoring point based on the information that exists for that point. This information can be limited from different perspectives.

For example, the bank manager might be interested in monitoring all tasks that have been done by a specific clerk if s/he works on collateral which is worth more than 1,000,000 USD in all processes. S(he) should select the row from the table that has ‘Workitem’ as level and ‘Completed’ as the state. Then, the editor recognizes what information is available in that state, viz., Control-flow, Data (both at the case and workitem levels), and Resource. The control-flow should not be limited to any process, so ‘*’ should be written—which indicates all processes. The Case-data should also be unlimited, so ‘*’ should be written. The Workitem-data should be limited to 1,000,000, so an xpath can be written to check the data condition, viz., ‘//Collateral/Amount >1000000.’ The Resource should also be limited to the specific clerk, so the name of the clerk can be written in the Resource section.

This editor writes all rules in an XML file, which is used by Observer Service to monitor the process instances. We limited the user to selecting the
level and states when defining the rule. However, if the user is interested in defining a rule for all levels or states, (s)he can still do that by changing the level or state field to ‘*’ in the XML file. The architecture of the service is explained in the next section.

Architecture

The Observer Service is responsible for tracking process instances based on monitoring rules, which are composed using the editor. The service is based on Service Oriented Architecture. It monitors process instances using events, which are received from WfMS. Therefore, it can be configured to observe any WfMS. We chose YAWL as the WfMS for which we monitor process instances. YAWL is selected since it supports a full workitem life cycle, and it supports many workflow patterns. It also has formal definitions and semantics. Moreover, it is open-source and was developed based on Service Oriented Architecture [43, 44].

Figure 4.10 shows the architecture of our service and its relation to the WfMS. The service is connected to the YAWL Engine through two interfaces: B and X. Interface X is used to monitor case monitoring points and the “s:enable” event from workitem monitoring life cycle; while interface B is used to monitor the workitem life cycle.

The resource service also plays an important role here. It is responsible for offering and allocating workitems to the users. Therefore, it initiates changing some workitem states. This service collaborates with the YAWL engine through interface B to change the state of Workitems. Interface A is used to upload a specification to the engine when a user launches a new process.
The resource service also reads the organizational model through Interface O, which can be used for extending monitoring rules.

The Observer Service does not track other services; instead, it tracks the changes in the workitem and case states through the engine. The service also reads the rules (composed by the Rule Editor) from Rule Repository. The rules specify which events should be captured. In the next section, there will be presented a case study which was conducted to evaluate the approach and artifact.

4.3.4 Evaluation

This section evaluates this proposed approach through a case study. In this study, different banking processes were considered. From among them, the “release collateral” business process was selected. The aim of this process is to release the collateral when the debt is not fully paid. The bank can decide about releasing the collateral based on the customer’s record. There are many types of collateral, such as shares, stocks, warrants, options, and a co-signer. We excluded co-signer collateral since it makes the process much more complex. This complexity would make the presentation of the process in this paper impossible.

Figure 4.11 shows this process model from the control-flow perspective. The process starts when a small business manager receives and registers documents from a customer to release his or her collateral. Then (s)he checks all supporting documents. If a document is missed, then (s)he asks the customer to send complementary documents. When all documents are collected, they are sent to the high-risk fraud manager for additional review and validation of the originality of the documents. It is a security policy in the bank to check the originality of all documents which are received from other parties except other banks. Then the branch manager can come up against any of the following situations:

- Declaration of a criminal case: If the high-risk fraud manager detects a document as a fake, the branch manager will declare the case to be criminal.

- Completion of documents: The branch manager may ask for more supporting documents before any decision is made as to the collateral release.

- Rejection: If (s)he decides to reject the release, then the customer will be notified.
Acceptance: if s/he grants the release request, then two different activities can be performed, depending on the type of the collateral:

If the collateral is in stocks, warrants, or options, then the small business manager contacts the investment brokerage office and receives the required information, such as the most current investment activity statement. Then, the branch manager decides on the case, i.e., rejection or acceptance. If s/he rejects the case, the customer will be notified; otherwise, the requested collateral will be released. Then, the congratulations letter will be sent to the customer.

If the collateral is in shares, then the small business manager contacts the investment broker and the lawyer who originally published the share certificates to the customer. When s/he receives the notarized share certificates, the high-risk fraud manager should validate the originality of the document, due to the security policy. If the high-risk fraud manager detects that the document is a fake, then the branch manager will declare the case to be criminal. Otherwise, the business manager accepts the case, as happens in most of the cases. However, if s/he rejects the case, the customer will be notified. In case of acceptance, the collateral in shares will be released, and the certificate and congratulations letter will be sent to the customer.

The business manager might be interested in being notified if someone works on collateral that has a very high value. The high value is subjective and can be varied at times, so it should be determined by the business manager. Currently, monitoring points are defined for all activities to capture such events. However, with the implemented artifact, this kind of alarm can be defined through a single rule.

The rule is defined as:
The modeling phase of this case study is finished, and the implementation phase is still in progress. We also found that if tasks can be categorized, it would be highly beneficial for defining the monitoring points. For example, a bank manager might be interested in monitoring all payment tasks, or all financial tasks. If the tasks’ types can be defined in process models, this can be also used when tracking them.
5. Conclusion

Separation of cross-cutting concerns is an important issue in all Information Systems disciplines, which is investigated in many areas like Programming. In Business Process Management, although different researchers have investigated aspect oriented business process modeling, such models are not executable in practice due to the lack of formal definition of notation and operational semantics. Thus, several artifacts are developed in this thesis to enable fill this gap and enable enactment of aspect oriented business process models. To extend the support of aspect oriented business process models in the whole BPM life-cycle, this thesis answer three questions, i.e. i) how can an aspect oriented business process modeling technique be formally defined?; ii) how can an aspect oriented business process model be implemented/configured?; and iii) how can aspect oriented business process models be run/adjusted?

The first question is answered through defining formal syntax for aspect oriented business process models. These definitions are evaluated through a case study, where the proposed approach is applied on two banking process models. The result showed how separation of cross-cutting concerns facilitates dealing with complexity through encapsulating the concerns into modules.

The second and third questions are answered through defining the formal semantics for enacting aspect oriented business process models. The semantics is defined using Coloured Petri Nets, and they are implemented in CPN Tools. The models are verified through state space analysis. The semantics is used to develop a service to extend the capability of a WfMS to support enactment of such models. A rule editor is also developed to support definition of rules, which are needed to perform the separation. This study shows a need to investigate how rules can be defined to support separation of cross-cutting concerns based on different business process perspectives. Therefore, a new study is conducted in which formal definitions of rules’ syntax are defined. The syntax is used to define an algorithm to monitor the rules when business process instances are running. The algorithm is developed as a service to observe defined situation when business processes are enacted. The approach is evaluated through a case study. The result shows how the definition of rules can support multi-perspective separation of aspects.

To sum up, the three major contribution in this thesis extended the theory of how aspect oriented business process models should be implemented, con-
figured, run and adjusted. These contributions enable enactment of aspect oriented business process models and extend this area to be investigated through running such models. The executable processes can support the investigation of the benefits of such separation in enactment. It also extends the support for such separation in the whole business process management life-cycle.

This work also shows a direction for future work such as i) an investigation of the possibility to define nested aspects, i.e. an aspect that is related to other aspects. This investigation can extend the area of supporting business process management to address complex circumstances. ii) a definition of semantics for static weaving. This definition can clarify how cross-cutting concerns can be modeled. Indeed, it is already assumed that all aspects can incorporate different workflow patterns. However, this work showed that it is not the case in dynamic weaving. The definition of semantics in static weaving can also specify if there is limitations to design aspects at the design time or it can be modeled as an individual process model. iii) an investigation on how the resource patterns [50], e.g. separation of duties and retain familiar, should be captured in aspect oriented modularization. This study can support human patterns that are needed in the area of business process management. iv) an approach to discover cross-cutting concerns from process logs. Such discovery can support identification of aspects from process log; thus, it can reduce the cost of identifying aspect oriented process models. v) an investigation of how aspects can be separated in knowledge intensive business process management. This investigation can extends the area of supporting separation of cross-cutting concerns to case oriented business process management.
References


Included Papers