Foundation of Aspect Oriented Business Process Management

Amin Jalali
Foundation of Aspect Oriented Business Process Management

Amin Jalali

Abstract

Reducing the complexity in information systems is a main concern on which researchers work. Separation of concerns, also known as the principle of ‘divide and conquer’, has long time been a strategy for dealing with complexity. Two examples of the application of this principle in the area of information system design are the break out the data management into Database Management Systems (DBMSs) and the separation of the business logic from the application logic into Business Process Management Systems (BPMSs). However, separation of cross-cutting concerns from the core-concern of a business process is not yet supported in the Business Process Management (BPM) area. Aspect Oriented principle recommends such a separation. When looking into the business process, several concerns, such as security and privacy, can be identified. Therefore, a formal model that provides a foundation for enabling BPMSs to support separation of concerns in BPM area is needed.

This thesis provides a formal model for dealing with separation of concerns in the BPM area. Implementing this model in BPMSs would facilitate the design and implementation of business processes with a lower level of complexity, which in turn would reduce the costs associated with BPM projects. The thesis starts with a literature review on aspect orientation both in programming and in the BPM areas. Based on this study, a list of requirements for an Aspect Oriented Service for BPMSs is compiled. Then a formal model for such a service, fulfilling a set of these requirements, is designed using Coloured Petri Nets and implemented in CPN Tools. The model is evaluated through the execution of a number of scenarios. The solution is also validated through an industrial case study. The results of the case study are presented the direction for future work outlined. The case study demonstrates that separation of concerns through aspect orientation does indeed reduce the complexity of business process models.

Keywords

Acknowledgements

This thesis would not have been possible without the guidance, help and support of my supervisor, Petia Wohed, PhD, the Deputy Head of the Department of Computer and Systems Sciences of Stockholm University. Therefore, I would like to express my deepest gratitude to her for all her assists. Moreover, I would like to express my warmest appreciation to one of my best friends, Mr. Saman Mesgari. The validation part of this thesis would not be possible without his help and guidance. I would also like to express my deepest gratitude to Anne Håkansson, Associate professor, Ph. D., Senior Lecturer, Director of studies, Department of Computer and Systems Sciences, for her valuable comments, which make the quality of this thesis better. I also thank Professor Paul Johannesson for his feedback on the method section. Last but not least, I like to acknowledge my parents for their constant support and unconditional love.
# Table of Contents

1. **Introduction** .................................................................................... 1  
   1.1. Background ............................................................................................. 1  
   1.2. Problem Statement ................................................................................... 2  
   1.3. Purpose and Goal ..................................................................................... 3  
   1.4. Audience ................................................................................................. 3  
   1.5. Demarcation ............................................................................................ 3  
   1.6. Disposition .............................................................................................. 3  

2. **Research Method** ............................................................................ 5  
   2.1. Introduction ............................................................................................ 5  
   2.2. Research Paradigm ................................................................................... 5  
   2.3. Research Framework ................................................................................ 6  
   2.4. Research Process ..................................................................................... 7  
      2.4.1. Problem Identification ......................................................................... 7  
      2.4.2. Objectives’ Definition .......................................................................... 8  
      2.4.3. Model Design ..................................................................................... 8  
      2.4.4. Model Implementation ........................................................................ 9  
      2.4.5. Evaluation ......................................................................................... 9  
      2.4.6. Validation ........................................................................................ 10  
      2.4.7. Conclusion ...................................................................................... 10  
   2.5. Design Science Research Guidelines ....................................................... 10  
      2.5.1. Guideline 1: Design as an Artifact ...................................................... 10  
      2.5.2. Guideline 2: Problem Relevance ........................................................ 10  
      2.5.3. Guideline 3: Design Evaluation ........................................................... 11  
      2.5.4. Guideline 4: Research Contributions ................................................... 11  
      2.5.5. Guideline 5: Research Rigor ............................................................... 11  
      2.5.6. Guideline 6: Design as a Search Process .............................................. 11  
      2.5.7. Guideline 7: Communication of Research ............................................. 12  

3. **Extended Background** ................................................................... 13  
   3.2. Coloured Petri Nets ................................................................................. 14  
   3.3. Aspect Oriented Programming ................................................................. 14  
   3.4. Aspect Orientation in BPM ...................................................................... 19  
      3.4.1. Aspect Oriented Extension to Business Process Modeling Notation (AO4BPMN) ........................................................................................................................................................................... 19  
      3.4.2. Aspect Oriented for Business Process Execution Language (AO4BPEL) ...................................................................................................................................................................... 21  
   3.5. Flexibility in BPM .................................................................................... 22  
      3.5.1. Process flexibility .............................................................................. 22
Scenario 2 ................................................................................................... 79
Scenario 3 ................................................................................................... 83
List of Figures

Figure 1 scattered cross-cutting concerns in business processes................................. 1
Figure 2 An excerpt from Change asset deal process of a bank........................................ 2
Figure 3 the relation between design science and behavioral science [3 p. 11].................. 5
Figure 4 "Design Science Research Cycles" [3 p. 16][4 p. “80].......................................... 6
Figure 5 Design Science Research Framework reprinted from [3 p. 27]............................ 7
Figure 6 The research process of the thesis...................................................................... 8
Figure 7 Business process lifecycle.................................................................................. 13
Figure 8 code tangling problem....................................................................................... 15
Figure 9 code scattering problem reprinted from [10 p. 7]......................................... 15
Figure 10 addressing cross-cutting concerns in AOP reprinted from [11 p. 8]............... 16
Figure 11 AOP system components reprinted from [10 p. 13]...................................... 17
Figure 12 join points, signature patterns, and pointcut in AspectJ reprinted form [10 p. 55] 18
Figure 13 the execution order of advices reprinted from [10 p. 162]............................... 18
Figure 14 An excerpt from change asset deal process of a bank modeled using BPMN........ 19
Figure 15 An excerpt from change asset deal process of a bank modeled using AO4BPMN .... 20
Figure 16 Archive and Security aspects modeled in AO4BPMN...................................... 20
Figure 17 All possible BPMN models which are derived from the sample AO4BPMN model 21
Figure 18 the execution order of elements upon using around advice reprinted from [14 p. 325] 22
Figure 19 "Flexibility type spectrum" reprinted from [16 p. 12]...................................... 23
Figure 20 "External Architecture of the Worklet Service" reprinted from [17 p. 100]......... 24
Figure 21 Ripple Down Rules Example......................................................................... 25
Figure 22 "Exception handling primitives" reprinted from [20 p. 298].............................. 26
Figure 23 Pointcut language for Business Process Aspect Oriented Service............... 28
Figure 24 Same precedence advices.............................................................................. 30
Figure 25 Flattened version of same order advices......................................................... 30
Figure 26 Different precedence advices.......................................................................... 30
Figure 27 Flattened version of different order advices.................................................... 30
Figure 28 Required Aspect Interface............................................................................ 33
Figure 29 Aspect service starting point........................................................................... 33
Figure 30 Flattened version of the Aspect service starting point example...................... 34
Figure 31 CPN: Aspect service ...................................................................................... 35
Figure 32 CPN: Select Pointcut Service [22 p. 101]....................................................... 36
Figure 33 CPN: WeaveAspect....................................................................................... 37
Figure 34 CPN: Launch Aspect...................................................................................... 38
Figure 35 Merging Proceed tasks.................................................................................. 39
Figure 36 CPN: Proceed Advised Join Point................................................................. 40
Figure 37 Resuming advices......................................................................................... 42
Figure 38 CPN: Resume Advice..................................................................................... 43
Figure 39 Continuing the core-concern.......................................................................... 44
Figure 40 CPN: Finish Aspect....................................................................................... 45
Figure 41 CPN: Core..................................................................................................... 46
Figure 42 Scenario 1-a, Initializing aspect service......................................................... 48
Figure 43 Scenario 1-b, facing Proceed commands....................................................... 48
Figure 44 Scenario 1-c, Finishing execution of Advised join point.................................. 48
Figure 45 Scenario 1-d, Finishing advices execution..................................................... 48
Figure 46 Scenario 2-a, Initializing aspect service.......................................................... 49
Figure 47 Scenario 2-b, continuing advised join point.................................................... 49
Figure 48 Scenario 2-c, Finishing execution of Advised join point.................................. 49
Figure 49 Scenario 2-d, Finishing aspect execution......................................................... 49
Figure 50 Scenario 3-a, Initializing aspect service.......................................................... 50
Figure 51 Scenario 3-b, continuing advised join point.................................................... 50
Figure 52 Change asset deal......................................................................................... 52
1. Introduction

1.1. Background

Business processes refer to the collaboration between people, enterprise resources and Information Systems (IS) within an organization for achieving business goals [1 p. 4]. Effective and efficient business processes are hence vital for modern organizations [1 p. 4]. Business processes are described through business process models. These are often graphical and should be simple enough to be understood by people but sufficiently detailed to capture the complexity of the modeled processes. The level of complexity should be minimized as long as possible during the business process modeling. This issue is also one of the guidelines for business process modeling [2 p. 93]. According to [2 p. 8], it is more desirable to have a set of simpler business process models than a few complex ones.

One way to deal with the complexity of process models is through hierarchical decomposition. This decomposition is a divide-and-conquer strategy that breaks business processes into smaller sub-processes. According to the information hiding principle, the complexity is reduced because the sub-processes do not add to the complexity of the parent process [1 p. 220]. Furthermore, the decomposition also allows reusability of the sub-processes [2 p. 48], which means that one and the same sub-process can be invoked several times.

A type of complexity is rooted in the requirements imposed on a business process. For an instance, Figure 1 illustrates four business processes in a bank, i.e. Change asset deal, Deal for speculation, Open a Letter of Credit and Issue a bank draft. The Change asset deal process requires to be logged. It also establishes some security policies that should be followed in the business process. These logging and security policies are considered as different concerns in this business process. Moreover, each concern may appear in various business processes. For example, the security concern in Figure 1 should be fulfilled by all business processes. Thus, these concerns cross over the business processes. Therefore, these concerns are called cross-cutting concerns.

Furthermore, a business process might contain both cross-cutting concerns activities and business activities, which shape the whole business process with the help of each other. The business activities are called the core-concern of the business process.

![Figure 1 scattered cross-cutting concerns in business processes](image_url)
In addition, cross-cutting concerns are typically scattered through a business process. For an instance, the Archive task which represents the implementation of the logging concern is repeated in the Change asset deal process of the bank (See Figure 2).

The Change asset deal process starts when a back office employee fills in a position sheet. Next, the general manager either confirms the sheet or denies it. Despite the general manager’s decision, the case should be archived. The Archive task is classified as belonging to the logging concern, which is indicated in the model by coloring the task. If the sheet is denied, the process is terminated. If the position sheet is approved, the junior dealer makes the deal and fills up a deal slip. For security purposes, the chief dealer and the general manager sign the deal-slip in the specified sequence. The Sign tasks can be considered as part of a security concern, which is indicated in the model with coloring these tasks. Finally, the deal slip is archived; this Archive task is also classified to belong to the logging concern.

This example demonstrates how the security and logging concerns are scattered through the whole business process. As the number of tasks is increased, the complexity of the process model is also increased. Note that in this case, the complexity cannot be reduced by a heretical decomposition, as it does not make sense to represent the task Archive and Sign as individual sub-processes.

To address the complexity issue arising from capturing different concerns in a process, Edsger Dijkstra proposes the aspect oriented principle, i.e. separation of the cross-cutting concerns from the core-concern [1 p. 24]. Separation of concerns could be performed in two levels; task and process. The task level separation of concerns refers to the design of activities with proper granularity [1 p. 218]. The model in Figure 2 is already at this level. Nevertheless, it does not solve the problem of scattered different types of activities through each business process, which is the process level of dealing with separation of concerns. The process level of separation of concerns refers to the representation of each concern in a separate process model, which wherever needed is associated to the process model of the core-concern. The model in Figure 2 does not reach the process level of separation of concern.

1.2. Problem Statement

Despite it has been postulated that Aspect Orientation would be beneficial for reducing process model complexity, the aspect oriented principle is not supported in current Business Process Management Systems (BPMSs). The lack of a foundation of aspect orientation in the Business Process Management (BPM) area disables separation of scattered through cross-cutting concerns from the core-concern of business processes. Therefore, business processes become more complex and less reusable. The complexity makes the negotiation of business processes harder, which adds cost to the development. Moreover, business process with less reusability adds to the cost of design, development, management and maintenance of business processes, because cross-cutting concerns are implemented in several places simultaneously.
1.3. **Purpose and Goal**

The purpose of this thesis is to describe a foundation of aspect oriented business process management. The goal of the work is to design a formal model for separation of concerns in BPM area. This model will provide the foundation of Aspect Orientated Business Process Management that can be utilized to extend the functionality of BPMSs to support the aspect oriented principle.

1.4. **Audience**

This thesis is written for different category of the audience. Business Process Management Systems’ producers are the main audience of this thesis. This group could extend their BPMS functionality to support aspect oriented business process design and management by reading this thesis. Moreover, business process analysts and designers could grasp how the separation of cross-cutting concerns could help them to model more reusable business processes with less complexity. Business process managers could also get a comprehension of how the aspect oriented service could reduce their business process complexity and increase their reusability by reading this thesis. The thesis can also be beneficial to students. Through this work the students can learn how aspect orientation can be applied when modeling business processes. This would help them to deal with complexity of process models.

1.5. **Demarcation**

The scope of this thesis is to address cross-cutting concerns to reduce the complexity in business processes. However, increasing the flexibility of business processes through this service is out of the boundary of this thesis.

1.6. **Disposition**

Apart from the introduction that has been given in this chapter, the rest of this thesis is composed as follows:

- Chapter 2, Research Method, describes the process and methodology which are used in this research to build the artifact. It also provides detailed information regarding guidelines that are used to direct the research process.

- Chapter 3, Extended Background, provides information and relevant ideas about how the cross-cutting issue is addressed in related works. This information is used to define service requirements in the Requirements Definition chapter. It also covers the flexibility issue in business processes at the end.

- Chapter 4, Requirements Definition, investigates needs, which are required to design a service to address cross-cutting requirements in the realm of BPM. It also frames the requirements that are going to be fulfilled in Service Design chapter.

- Chapter 5, Service Design, provides the architecture of the service. It also explains the designed conceptual model and describes its functionality. This design proofs the applicability and completeness of gathered requirements. The rigorousness of the result is also evaluated in this section.
• Chapter 6, Validation, validates the value of the result using practical examples. Deal processes of a bank are analyzed to validate the application of the aspect oriented service for BPM i.e. ‘change asset deal’ and ‘deal for speculation’.

• Chapter 7, Analysis, discusses the contribution of the evaluated artifact and the ways it attains the problem statement.

• Chapter 8, Concluding Remarks, finalizes this thesis by introducing future work and existing problems.
2. Research Method

2.1. Introduction
This chapter provides a justification for this thesis. It starts by introducing the scientific paradigm to which this thesis contributes. The research framework that leads the process of this thesis is introduced afterwards. The process of conducting this thesis is discussed, and its steps are introduced. For each step, the applicable methods are introduced, and the utilized methods are reasoned out. Finally, the application of design science guidelines in this research is investigated.

2.2. Research Paradigm
There are two paradigms that produce knowledge in Information Systems’ area, behavioral science and design science [3 p. 5]. Design science aims to produce utilities for needs using information technology; while behavioral science tries to complement the design science through developing and justifying theories to explain or predict the truth around the needs (see Figure 3)[3 pp. 5, 10, 11][4 pp. 76, 79].

This thesis belongs to design science paradigm and aims to extend the capabilities of BPMSs by creating an innovative artifact. According to the design science, the artifact could be a kind of construct, model, method, instantiation [4 p. 78] or a better design theory [3 p. 6]. This thesis aims to produce a model to support separation of cross-cutting concerns.

Hevner et al. (2004) provides a conceptual framework to explain processes inside and around a design science research [3 p. 16][4 p. 77]. This framework contains three cyclical processes that are performed in a design science research (see Figure 4)[3 p. 16][4 p. 80].
The environment and knowledge base parts contribute to the design science research with two cyclical processes, relevance and rigor cycles [3 p. 17][4 p. 79].

The relevance cycle helps researchers to frame the research and define the problem statement [4 p. 79]. It also helps researchers to gather requirements, which will be used in the 'Evaluate' step of design science research [3 p. 17].

"The rigor cycle provides past knowledge to the research project to ensure its innovation" [3 p. 18]. The past knowledge could be provided in the form of experience and expertise or existing artifacts and processes [3 p. 18]. This information demonstrates the contribution of the research, and the new knowledge that the research adds to the existing knowledge [4 p. 81]. Indeed, it is a vital part of each research to be based on current theories to guarantee the sufficient brilliance in solving the problems [5 p. 12].

The design cycle is a core process among these cyclical processes; it builds and evaluates artifact based on the inputs from relevance and rigor cycles [3 p. 19].

### 2.3. Research Framework

A research framework helps researchers to conduct their researches based on a tested framework. Design science research framework is proposed by Vaishnavi and Kuechler (2004) for conducting design science researches. This framework describes the general line of reasoning as a process that is adapted for design science researches (see Figure 5). The process is started with the ‘Awareness of the problem’ step during which the problem is identified and defined. Afterwards, a solution is proposed based on existing knowledge in the Suggestion step. The solution is implemented in the Development step as an artifact. The designed artifact should be evaluated using action research, controlled experiment, simulation or scenarios. This is done in the Evaluation step. Finally, the result could be finalized in the Conclusion step. This framework introduces an iterative research process. Iterations are based on a feedback from each step. The feedback is called Circumscription [3 p. 27].
Figure 5 Design Science Research Framework reprinted from [3 p. 27]

The process of conducting this thesis is an application of the design science research framework, which is explained as follows in the next sub-chapters.

2.4. Research Process

The utilized process to conduct this research is an application of the design science research framework. The process contains six steps (see Figure 6). These steps are influenced by the line of reasoning that this thesis follows. There are two types of reasoning to conduct a research, deductive and inductive [6 p. 25]. Deductive reasoning starts with defining a theory, which assists researchers to make a prediction through formulating a hypothesis. Conversely, “Inductive reasoning is generalising from a specific instance, or several new facts, to a more general idea” [7 p. 14]. Deductive reasoning is not applicable to this thesis, because the theory of the aspect orientation is not defined for the BPM area. Therefore, inductive reasoning is employed to gather several facts about current work related to the aspect orientation in both programming and BPM area. These facts are discovered through the first step of the research process that is described below.

2.4.1. Problem Identification

The problem identification is the first step to achieve the ‘awareness of the problem’ in design science research framework. There are several methods that could be utilized such as interview, questionnaires, observation [8 p. 8] and literature study. Literature Study is the only applicable method that could be utilized in this research, because the aspect orientation in BPM area is not implemented; hence, it could not be investigated by other methods. The literature study not only helps the definition of the problem, it also extracted the facts on which the construction of the intended artifact could be based. The facts are qualitative data that are collected from the domains of both BPM and programming. These facts guarantee the uniqueness of the research based on the past knowledge. Therefore, this step is performed through both relevance and rigor cycles of the design science.

Collecting data with one method could impact the result. The result could be limited to previous researches that have been done in a research domain, so the innovation of the produced artifact could be limited to that domain. To deal with this problem, data are gathered from two different domains, i.e. programming and BPM. The aspect oriented principle is mature in the programming area, because it is
Currently implemented. Therefore, the requirement definition of this research will not be limited to current works in BPM area. The problem identification is performed in Introduction and Extended Background chapters.

![Diagram of Design Science Research Cycle](image)

**2.4.2. Objectives’ Definition**

The second step to conduct this research is the objectives’ definition. In this step, the awareness of the problem is achieved by analyzing the collected data. Document analysis method is utilized to define the requirements based on the facts that were gathered in the previous step [8 p. 8]. These requirements are used in the next step to design the formal model. Therefore, this step is performed based on business needs and application knowledge, which covers both relevance and rigor cycles. Quantitative Data Analysis is not applicable in this step, because qualitative data are gathered through the previous step. This step is performed in Requirements Definition chapter.

**2.4.3. Model Design**

The third step to conduct this research is the model design. This step, starts by suggesting a tentative design of the artifact as a model. There are different modeling standards that could be utilized to
model this artifact such as UML, Petri nets and Coloured Petri Nets. UML models could not be investigated in terms of correctness, completeness and consistency, because it is not based on a mathematical definition and semantic. Therefore, it is not selected as modeling standard for designing the artifact. The Petri net models are based on the mathematical definition, so they could be analyzed and evaluated. However, they are unsuitable for designing complex models, because they will become too large and inaccessible[2 p. 41]. Therefore, this modeling standard is not utilized for designing the artifact. Coloured Petri Nets (CPN) models are also formal and based on the mathematical definition and semantic [9 p. 6]. CPN is a high level Petri nets that enables modeling complex situations using structured and accessible models [2 p. 41]. Thus, the aspect oriented service in BPM area is designed using Coloured Petri Nets. The Model Design step is a part of develop/build activity in design science research (See Figure 4), which is a part of Design Cycle. The Model Design step is performed in Service Design chapter.

2.4.4. Model Implementation

The fourth step to conduct this research is the Model Implementation. The designed artifact is developed in this step. The design and implementation of a CPN model are performed in CPN Tools. In fact, Model Design and Model Implementation steps are performed simultaneously in CPN Tools. CPN Tools also check the correctness of the model at the same time. The model implementation step is a part of Design cycle of the design science research (see Figure 4). The Model Implementation step is performed in Service Design chapter.

2.4.5. Evaluation

The fifth step to conduct this research is the Evaluation. The implemented artifact is evaluated in this step. The produced artifact should be evaluated rigorously in the design cycle. The evaluation step of the design science research framework is conducted in two steps, i.e. evaluation and validation. The evaluation step checks the built artifact with rigorous methods. There are different methods for evaluating a CPN model, i.e. state-space analysis and different simulation methods.

State-space analysis checks and verifies the correctness of the model. This method could be used if the CPN model contains finite number of reachable states. However, even a small CPN model could have infinite number of reachable states let, alone a big real-life CPN model. Therefore, it would be very hard and problematic to use this method for verifying a real-life system[9 p. 7]. Thus, this method is not used for verifying the produced artifact.

Simulation methods analyze the CPN models to check models’ functionalities [9 p. 3]. They reveal design errors that a CPN model might contain [9 p. 3]. There are two kinds of simulation, automatic and interactive. Automatic simulations are usually used to test the performance of the CPN model. The time does not have any weight in this research, so applying automatic simulation is not meaningful for the produced artifact. Interactive simulation is used to evaluate the functionality of the artifact, so it is the suitable choice for evaluating the proposed artifact. Scenarios are used to lead the interactive simulation.

The Evaluation method is rigorous because CPN Tools evaluation methods are based on a mathematical foundation. It provides utility to assess the built artifact which is a part of the design cycle. It also provides addition to the knowledge base in terms of a formal model which is a part of the rigor cycle. The results from the Evaluation step are described in Service Design chapter.
2.4.6. Validation
The validation step checks the relevancy of the designed artifact. The relevancy of the artifact should be appraised with its application in an appropriate environment. There are different methods to perform this kind of evaluation such as case study and usability testing. Usability test is not applicable for this thesis because the produced artifact could not be utilized to be tested by specific users for specific goals in a specific context of use. The produced artifact is a kind of formalization with mathematical semantic and definition, so this method of validation is not applicable.

Case study is used to validate this artifact. There are different types of case studies such as snapshot, longitudinal, pre-post, patchwork and comparative [8 p. 27]. Snapshot case study is utilized to validate the artifact. Other case studies are not selected to perform because of time limitation. However, this limitation does not affect the result of this research, because this thesis aims to provide a formal model for aspect oriented service in BPM area. Therefore, a neat case study is enough to check the application of the designed artifact in an appropriate environment.

The information for the case was gathered through an open interview with a business process expert in the banking domain. Two business processes of the bank are analyzed. The application of the artifact on these processes is investigated to check the relevance of the artifact. Therefore, this step plays roles in both relevance and design cycles. The Validation step is performed in Validation chapter.

2.4.7. Conclusion
The last step is the Conclusion in which the result is analyzed, and a conclusion drawn. This process is iterative, so the process could be started again if the conclusion does not cover the selected set of requirements. Future work is also defined at the end of the conclusion based on new knowledge that is discovered through this step. The Conclusion step is performed in Analysis chapter.

2.5. Design Science Research Guidelines
Hevner et. al. provides seven guidelines, which could be used to conduct, evaluate and present design science researches [3 p. 12][4 p. 82]. Each design science research should address each of these guidelines in some manner. This thesis follows these guidelines in order to provide an effective design science research [4 p. 82]. Applications of design science research guidelines are discussed to justify and legitimate the paradigm of this thesis [3 p. 28].

2.5.1. Guideline 1: Design as an Artifact
This guideline states that each design science research should produce an artifact in a form of a construct, a model, a method or an instantiation [4 p. 83]. This thesis produces a model based on Coloured Petri Net formalisms. This model demonstrates a prototype that enables considering the effect of designed solution on the defined problem.

2.5.2. Guideline 2: Problem Relevance
This guideline emphasizes that the result of a design science research should solve relevant business problems using a technology-based artifact [4 p. 84]. The complexity is a main issue in information systems researches. One type of complexity is rooted in cross-cutting functionality, which are scattered through the systems. Cross-cutting functionality could be addressed by aspect oriented
principle. This thesis provides a formalized model that describes how the aspect oriented should be addressed in BPM area. Therefore, it is a very relevant problem that is addressed in this thesis.

2.5.3. Guideline 3: Design Evaluation

This guideline states that “The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well executed evaluation methods” [4 p. 85]. Hevner et al. suggests different methods for evaluating artifacts [4 p. 86]. The produced artifact is evaluated using interactive simulation, structural testing and scenario methods. These methods evaluate the artifact rigorously, because simulation techniques in CPN Tools are based on the mathematical definition.

2.5.4. Guideline 4: Research Contributions

This guideline states that a design science research should contribute at least in one of the following areas: design artifact, foundation and/or methodologies [4 p. 86]. A contribution of a research is considered as the design artifact if the research produces an artifact that solves an unsolved problem [4 p. 87]. This thesis provides a formal model for solving the problem of scattered through cross-cutting functionality in BPM area. There is not any formal model defined for this purpose, so this research contributes in the design artifact area. A contribution of a research is considered as the foundation if the research produces a modeling formalism [4 p. 87]. The produced artifact is a formal model based on Coloured Petri Nets formalism, so this research also contributes in the foundation area. A contribution of a research is considered as the methodology if the research develops and uses a creative evaluation method [4 p. 87]. This thesis does not contribute in this area.

2.5.5. Guideline 5: Research Rigor

This guideline states that a design science research should follow rigorous methods in both constructing and evaluating the artifact [4 p. 88]. This research follows a rigorous method in constructing the artifact, because the requirements are gathered through literature study that is based on application knowledge. The application knowledge is extracted in the rigor cycle (See Figure 4). This thesis also follows a rigor method in evaluating the artifact. The evaluation step of the utilized design process evaluates the artifact using simulation, scenarios and white-box testing (See Figure 6). Therefore, addition to the knowledge base in the form of formalism is provided using utilized rigor method (See Figure 4).

2.5.6. Guideline 6: Design as a Search Process

This guideline defines means, ends and laws as components of a design science research that provides an effective solution for the problem in an iterative approach. Means are defined as a set of actions and resources that could be utilized to reach the goal. Ends are defined as a set of goals and constraints that the research contains. Laws are defined as uncontrollable forces in the environment which could affect the research [4 p. 88].

This research is conducted based on thorough literature study about the aspect oriented paradigm in both programming and BPM areas. Therefore, terminologies and constraints are defined using existing application domain knowledge in Requirements Definition section. The formalization of expected service is designed based on a selected set of requirements. The selection process performed based on the research goal and constraints, which form the ends. The artifact is designed and implemented based on selection of a set of requirements using CPN, which has a mathematical definition in syntax.
and semantics. Therefore, the design could be validated to check correctness and consistency of the model using this means.

All requirements are not selected to be implemented in this research. Therefore, the artifact will not be complete that could address all possible business processes, so some parts of the research to complete formalization model of the aspect oriented service is left for future research.

2.5.7. **Guideline 7: Communication of Research**

This guideline states the importance of presentation of a design science research for both *managerial* and *technical* audiences [4 p. 90]. This research targets two categories of audiences, technical and managerial. The Service Design section is written for the technical audience who has a deep knowledge of CPN.
This chapter starts by giving an introduction about the BPM. Next, it introduces related work on how the separation of concerns is addressed in both programming and BPM areas. Afterwards, it gives an introduction of flexibility in BPM area.

### 3.1. Business Process Management

Business Process Management (BPM) is a set of concepts, methods and technologies, which enable designing, analyzing, configuring, enacting and administrating of business processes [1 p. 5]. Each business process usually has a life cycle, called business process life cycle (See Figure 7). The life cycle is started by designing a business process. In this phase, business process is identified and modeled. Next, the business process should be analyzed. The analysis is started by validating the business process model. The validation could be supported by simulation techniques to investigate unforeseen needs that might be missed in the business process model. Last but not least, the model should be verified, which ensures the correctness of the designed business process.

After designing and analyzing a business process, it is time to configure it. This step consists of system selection, implementation, test and deployment. There is generic software that enables enacting business processes, called Business Process Management System (BPMS) [1 p. 6]. A BPMS should be selected in order to implement the business process. The test and deployment activities are also performed to complete the conjuration phase. The relation between the business process and people in addition with connecting existing systems with BPMS are defined in this phase.

Afterwards, an instance of the business process could be enacted in selected BPMS. BPMS has components for monitoring the current state of a running instance. Maintenance of running instance has also a great matter of importance since the business goal should be achieved by running the business process successfully. Moreover, Valuable information is kept in log files during the enactment phase. This information is used to evaluate the enacted business process. Process mining
applies data mining algorithms on these log files to extract valuable information, which are buried in these files.

### 3.2. Coloured Petri Nets

A model with precise definition and well-defined and unambiguous specification is called formal. Petri nets are formalized modeling techniques with which a business process could be specified in an abstract way [1 p. 149]. These models are independent of execution environment and have a strong mathematical basis [2 p. 41]. Petri nets specify a model using places and transitions. The state of the model could be identified by distribution of tokens within the model [2 p. 37]. However, tokens could not be distinguished from each.

Petri nets models could be very large and inaccessible for describing complex models [2 p. 41]. Therefore, high level Petri nets are introduced to address this problem. Coloured Petri Nets (CPN) addresses this problem by adding data to the nets; the data types are called colour sets [1 pp. 155, 156]. CPN models are executable models with the ability of performing simulation. Thus, different scenarios could be investigated through simulations and the behavior of the system could be validated [9 p. 4]. CPN Tools is a tool that supports designing, executing and simulating CPN models [9 p. 8].

### 3.3. Aspect Oriented Programming

Separation of concerns, also known as the principle of ‘divide and conquer’, has long time been a strategy for dealing with complexity. Two examples of the application of this principle in the area of information system design are the break out the data management into DataBase Management Systems (DBMSs) and the separation of the business logic from the application logic into BPMSs. When looking into the business logic, several concerns, such as security and privacy, can in turn be identified. The idea of separating the different relevant concerns, from each other and presenting them as individual aspects linked to the core-concern was first applied in programming and called Aspect Oriented Programming.

Aspect Oriented Programming (AOP) is a paradigm for addressing the 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 scattered code along the application program [10 p. 3]. It also solves two problems of traditional programming approach that are described as follows.

Different aspects of a program are written in a module of code traditionally (See Figure 8). Therefore, each module contains different concerns of a program each which handles one aspect such as transaction management, security or exception handling. **Code tangling** refers to this 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 separated modules, but each module should call this functionality to implement relevant concerns (See Figure 9). Code scattering refers to this phenomenon of implementing the same cross-cutting concerns within different program modules or applications [10 p. 7]. An example of code scattering is the implementation of security concerns for authentication and authorization in the ATM, Accounting and Database modules of an application (See Figure 9) [10 p. 7].

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 in all around the program. This means spending lots of resources and time to apply a change in a concern. Therefore, changes with be more costly. Second, if a piece of code in 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 codes should be repeated in new modules if they should implement relevant concerns. Hence, these methods of programming do not support a good degree of reusability, 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 faces less traceability, reusability, quality and productivity, which make the evolution of software more difficult [11 p. 7].
AOP proposes implementing concerns as individual program modules and linking them to the core concern to solve code tangling and code scattering problems (See Figure 10). To enable separation of cross-cutting concerns, AOP implements a number of constructs (14, pp. 11-12.). A class diagram of them is shown in Figure 11 and a description of these is given below.

- **Joint points:** Join points are defined as identifiable points in the running application [10 p. 11]. These points enable the addition of one or several cross-cutting concerns to a core-concern. There are three different types of joint points through which functionality can be added to a program module such as field, method and type. These types are also called signatures. They will be described shortly.

- **Pointcut:** A language that enables definition and selection of relevant join points in the running 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 to the core-concern; these constructs are cross-cutting concerns like security concern. Each advice could be executed before, after or around the advised join point. Advised join point nominates the join point for which the advice should be executed.

- **Weaving:** A mechanism to alter the static structure of core-concern using execution of cross-cutting concerns. 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. Aspect contains pointcuts and advices. In ATM example, the security concern should be implemented in a module, called security aspect. This aspect contains a pointcut and an advice. The pointcut specifies 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.

Figure 11 illustrates the relation between constructs in aspect oriented programming. As mentioned earlier, aspect is a module which contains cross-cutting concern. Each aspect consists of cross-cutting elements. Cross-cutting elements could be a form of dynamic crosscutting element, static crosscutting element and pointcut. The dynamic crosscutting element is known as advice. Each advice could be defined at the run-time. The static crosscutting concern could be a form of Inter-type declarations or weaving time declarations. These elements could be defined at the design-time. The pointcut, which is the other kind of crosscutting element, could select a set of join points that are exposed by the application.
As mentioned earlier, join point could be a kind of field, method and type signature. Figure 12 illustrates different join point signatures in an example. The Account class is a kind of type signature. The balance attribute of the Account class is a kind of field signature. Each attribute of a class could be set or got based on which pointcut could perform the selection. As a case in point, pointcut could specify that the join point is when the account balance is read. Method signature is the other type of join point that represents methods of a class, e.g. the credit method could be caught based on the method signature.

Figure 12 shows a special kind of method signature, ‘public * Account.*(…)’. This expression selects any method inside the Account class which is public regardless of the type that it returns. The number of arguments and their types does also not matter. The star symbol represents any in that expression. Moreover, each pointcut expression which selects the relevant join points could be a combination of explained pointcuts. There are two sorts of operators that enable this combination, unary and binary [10 p. 65].

AOP is implemented in AspectJ, which is a mature language for supporting aspect oriented programming in Java. AspectJ supports the negation union operator (!) and two binary operators (|| and &&). These represent the selection of ‘either’ and ‘both’ pointcuts accompanying the operator [10 p. 65].
Figure 12: Join points, signature patterns, and pointcut in AspectJ reprinted from [10 p. 55]

As it can be seen from the class diagram in Figure 11, several advices can execute in the same or different aspects for a specific join point. In such a case, the order of the execution of the advices is significant and needs to be determined. For this, both the type of the advice and an advice precedence order specified during design time are considered. If an advice should execute before a join point, a higher precedence advice should be executed first. However, if an advice should execute after a join point, a higher precedence advice should be executed last. The execution order for around scenarios is a combination of both. The higher precedence advice should be executed first, but the lowest precedence advice should be resumed after the join point is proceeded [10 pp. 161, 162] (See Figure 13). The execution order of multiple advices will be further described in Chapter 4.

Figure 13: The execution order of advices reprinted from [10 p. 162]
3.4. Aspect Orientation in BPM

Inspired by the ideas of AOP, cross-cutting concerns have also gained some interest in the BPM community. Recently, some work has been carried out for implementing the AOP ideas into BPM domain. Two bodies of work are worth mentioning in this context: Aspect oriented extension to Business Process Modeling Notation (AO4BPMN) and Aspect Orientation for the Business Process Execution Language (AO4BPEL). These are described in the following sub-sections.

3.4.1. Aspect Oriented Extension to Business Process Modeling Notation (AO4BPMN)

Aspect Oriented Extension to Business Process Modeling Notation (AO4BPMN) approach by Charfi [12] suggests an extension to Business Process Modeling Notation (BPMN) industry standard with concepts for dealing with aspect orientation. The terminology of AOP is utilized. Charfi defines join point as activities and events [12 p. 53]. In addition, he introduces three kinds of pointcut languages [12 p. 54]:

- Connecting pointcuts with related join points at design time, which requires having aspects and the main model in the same view; the scalability of the model is the main problem of this approach.
- Using a query based pointcut language such as OCL or QVT to specify the relation between each pointcut with relevant join point; the main problem is readability of such a model for user because they do not know such query languages.
- Defining textual annotation for each join point to relate them to appropriate pointcut is the other way that is proposed for illustrating pointcuts.

An advice is defined as a sub-process which contains the functionality of cross-cutting concern [12 p. 54]. A special kind of activity is introduced called Proceed activity. The Proceed activity is proposed to capture the point that caught join point should be resumed. It also determines the type of the advice [12 pp. 54, 55]. Aspects are defined as modules that contain the pointcuts and advices which include the cross-cutting modularity [12 p. 55].

The same excerpt from the change asset deal process, which is introduced in Figure 2, is illustrated in BPMN notation (See Figure 14). As mentioned earlier, there are two concerns in this piece of business process, i.e. security and logging.

![Figure 14 An excerpt from change asset deal process of a bank modeled using BPMN](image)

Figure 15 shows the same process which is modeled using AO4BPMN. The BPMN annotations in this figure represent the aspects that should be considered for each activity. The Confirm activity requires a logging policy, and the fill deal slip activity requires both logging and security policies. These policies are modeled in aspects which are modeled separately (See Figure 16). As mentioned earlier, each
aspect contains a pointcut and one or several advices. Pointcuts are defined using the BPMN annotation. The pointcut that is defined for the archive aspect is labeled as the logging policy pointcut. The pointcut that is defined for the security aspect is labeled as the security policy pointcut. Each aspect has one advice in this example.

![Figure 15 An excerpt from change asset deal process of a bank modeled using AO4BPMN](image)

The PROCEED activity represents the place that the advised activity should be executed. Therefore, AO4BPMN considers the PROCEED activity as an indicator which shows the type of advices and the execution order of activities of advices[12 p. 55]. However, it is not defined how the precedence should be modeled. If the PROCEED command is only used to determine the execution order of activities, it could work when there is only one advice from each type, i.e. before, after or around advice. However, this notation could not be used to model advices that contain more than one advice from the same type. For instance, the above example that is modeled with AO4BPMN has two after advices, i.e. archive and security. If we want to produce a BPMN model from this AO4BPMN model, several alternatives could be derived (See Figure 17). These variations could be drawn because the orders of these advices are not specified, so all possible orders could be assumed. Therefore, the precedence of advices should be defined in the model.

![Figure 16 Archive and Security aspects modeled in AO4BPMN](image)
Furthermore, The AO4BPMN defines join points from the control-flow perspective, yet it does not consider the definition of join points from the data and resource perspectives. As a case in point, consider a situation in which a logging concern should be executed when the account balance is changed. Account balance might be changed using different activities in various business processes. AO4BPMN could relate aspects to activities. However, the mentioned scenario requires a relation between aspect and data elements, account balance. Therefore, the pointcut language in AO4BPMN does not cover the other perspectives elements, data and resource perspectives. This matter reduces the flexibility of AO4BPMN and the degree of complexity that it could solve. Moreover, AO4BPMN is not formal, so it could not be validated and verified using mathematical notations.

3.4.2. Aspect Oriented for Business Process Execution Language (AO4BPEL)

AO4BPEL is an extension of Business Process Execution Language for Web Services (BPEL4WS) to support cross-cutting concerns through aspect oriented modularization [13 p. 169]. It also provides a mechanism to activate or deactivate aspects by supporting dynamic weaving [13 p. 169]. Activities in Business Process Execution Language (BPEL) are considered as join points[13 p. 175]. Relevant join points are selected by XPath based pointcut language[13 p. 175]. Pointcut elements are defined using <pointcut> element in AO4BPEL [13 p. 175]. Charfi A. and Mezini M. define an advice as “a BPEL activity that executes before, after, or instead of a join point”[14 p. 324]. They also define the precedence and dependencies to determine the execution order of advices when they share the same join point [14 p. 337].
The execution order is determined using the type property of each advice with the value of before, after or around. The advised join point would be replaced by an advice if the advice type is around. The join point will be executed using Proceed element, which could be defined in the advice (See Figure 18).

Figure 18 the execution order of elements upon using around advice reprinted from [14 p. 325]

There are three types of join point in AO4BPEL such as control-flow driven, data-flow driven and participant driven join points[14 p. 324]. Control-flow elements like split and join are supported using control-flow driven join points. Data elements such as initializing a variable are supported using data-flow driven join points. Participant driven join points support the interaction of a business process with partners. [14 p. 324]

Although AO4BPEL provides comprehensive support for join points, it does not consider resource driven join points. Moreover, AO4BPEL does not support any graphical notation for modeling business processes, which is critical for BPM area.

3.5. Flexibility in BPM

Business process complexity has also been addressed in some approaches for dealing with process flexibility. For instance, instead of modeling all possible exceptions in a process, the exceptions can be captured in a separate pool of sub-processes. When needed a sub-process from the pool is then selected and invoked. This is basically the idea behind the Worklet approach [15]. Similarly to Aspect orientation, it proposes a separation of concerns, namely of separation of possible changes of a process (i.e. the exceptions in the example above) from a core-process. Therefore, we included the work on process model flexibility in our study.

3.5.1. Process flexibility

Flexibility is the ability to adapt business processes with the foreseen and unforeseen changes in the environment [16 p. 1]. The flexibility issue in business processes could be discussed based on different business process perspectives such as control-flow perspective, organizational perspective, information perspective and application perspective [16 pp. 2, 18]. Schonenberg M.H. distinguishes four types of flexibility from the control flow perspective: flexibility by design, deviation, underspecification and change [16 p. 2].
Design type flexibility refers to the capability of defining various alternatives in a process specification. Workflow patterns like parallelism, iteration, interleaving, multiple instances and cancellation provide such a capability [16 p. 5].

Deviation flexibility refers to a short time change required in a business process at the run-time [16 p. 6]. Typical behaviors associated with this type of flexibility are undoing, redoing, skipping, invoking tasks, and creating additional instances of a task [16 pp. 6, 7].

Underspecification flexibility refers to the situation in which the designer does not know a part of business process specification, but he or she knows the exact point in a business process that should be defined later [16 p. 8]. Therefore, a placeholder is typically defined to indicate parts of the model to be modeled at a later stage [16 p. 8]. The specification of a placeholder can be performed during the design time (before placeholder execution) or at the run-time (i.e. placeholder execution) [16 p. 9].

Flexibility by change addresses the changes in a business process which could not be known beforehand [16 p. 10]. In contrast to deviation flexibility, it refers to persistent and not to short time changes. In addition it is associated with the introduction of new tasks or the removal of existing task or changing the order of existing tasks [16 p. 10].

Figure 19 illustrates the flexibility types described above and the area that they cover. The flexibility issues for the other three perspectives, organizational, information and application, have not yet been addressed by researches. In [16 p. 18] they are outlined as areas for future research.

![Flexibility type spectrum](image)

Figure 19 "Flexibility type spectrum" reprinted from [16 p. 12]

### 3.5.2. The Worklet Approach

The Worklet approach was introduced to support business process flexibility. It addresses deviation and change flexibility (upper right-hand corner in Figure 19), as well as, underspecification flexibility (lower part in Figure 19). These flexibilities are supported by a service called Worklet. The Worklet service is composed of two services, i.e. Worklet selection service for supporting design time flexibility and Worklet exception handling service for supporting run-time flexibility. The Worklet service is designed based on Service Oriented Architecture (SOA). Therefore, it is general and can be utilized to extend any WfMS to support flexibility. It has been supported by and tested in the YAWL environment. Figure 20 shows the architecture of this service in the YAWL environment.
The Worklet service supports flexibility based on situations that could happen in a running case. These situations are defined using rules in the Worklet service. Rule Editor is the application that enables the definition of these rules using a hierarchical binary tree, called Ripple Down Rules (RDR)[18 p. 48]. An RDR contains a set of rules. Each rule consists of a condition and a conclusion. The Worklet service evaluates an RDR to select the appropriate rule that should be applied for the running case. The selection process is performed based on traversing the three and evaluating the rules. The process starts from the root, which is always contains the Boolean value True as the condition. If a condition evaluates to True, the right-hand side child node is accessed; if it evaluates to False, the left-hand side child node is accessed. When a leaf node is reached, if its condition is satisfied (evaluates to True), the conclusion executed. Otherwise the traversal goes back and the condition of the last parent node that evaluated to True, the conclusion executed. Otherwise the traversal goes back and the condition of the last parent node that evaluated to True is executed.

Figure 21 demonstrates a RDR for an imaginary cash deposit task of a process of a bank. Rule 1 checks if the amount of money is greater than 2,000 EUR. If the condition is not satisfied, the left rule will be evaluated. The left rule checks whether the amount is greater than 500 EUR. If the condition is satisfied it means that the amount is less than or equal to 2000 EUR and more than 500 EUR. In such a case, ‘CheckCustomerHistory’ process will be executed in parallel with the main process. On the other hand, if the left child rule is not satisfied, Rule 5, no repertoire will be selected. Thus, the Worklet service does not execute any repertoire to alter the process.

However, if the rule number two is satisfied the selection process depends on the evaluation of the right child rule. The evaluation of the right child rule would be the same as rule number two which will be done in a recursive way. If the result of evaluation is false, then the repertoire of rule number two will be selected. Otherwise, the repertoire of the selected rule will be executed. The evaluation table of expression in Figure 21 explains the rule number that will be selected upon satisfying related expression.
As mentioned earlier, the Worklet Service contains two services such as Worklet selection service and Worklet exception service. The Worklet selection service supports underspecification flexibility. This service selects a business process to be launched instead of the placeholder [19 p. 215]. The Worklet exception service supports deviation and change flexibility. Therefore, it provides some controls over running case or task. These controls are called primitives. The primitives which are currently available in Worklet exception service are ‘Remove WorkItem’, ‘Remove Case’, ‘Remove All Cases’, ‘Suspend WorkItem’, ‘Suspend Case’, ‘Suspend All Cases’, ‘Continue WorkItem’, ‘Continue Case’, ‘Continue All Cases’, ‘Restart WorkItem’, ‘Force Complete WorkItem’, ‘Force Fail WorkItem’ or ‘Compensate’[17 pp. 102, 103]. These primitives enable the Worklet exception service to support change and deviation flexibility type in business processes. Figure 22 shows these primitives with graphical notations.
As mentioned earlier, the Worklet service supports the flexibility with the help of running a business process or executing primitives. This service continues the execution of the main process, if the service could not find the specified process [19 p. 137]. This is acceptable when the Worklet service handles the flexibilities that could be ignored. However, if the flexibility handles mandatory part of the business process like security, this service could not guarantee the proper execution of the business process. Therefore, this service could not be used to handle flexibilities that must be considered as a part of business processes.

Figure 22 "Exception handling primitives" reprinted from [20 p. 298]
4. Requirements Definition

The requirements for designing an aspect oriented service for the BPM area are defined and discussed in this section. Each aspect oriented service should define these concepts: join point, pointcut, advice, aspect and weaving process. Therefore, the requirements are structured along these concepts.

4.1. Join point

Join points are identifiable points in a running instance of a business process. These points could be identified from three business process perspectives, i.e. control-flow, data and resource perspectives. The definition of a join point from each business process perspective is exemplified as follows.

R1. Join points shall be possible to define for control-flow, data and resource perspectives.

As an example of a control-flow perspective join point, assume that a bank is interested to define security policies when the withdrawal is occurred in a specific business process. These policies should therefore be defined when the ‘withdraw’ task is performed. The ‘withdraw’ task is a control-flow perspective join point in this example. The join point in business process control flow perspective could be defined as an individual activity or task in a business process. This kind of join point is called task signature.

R1.1. Task signature shall be possible to define in the control-flow perspectives.

As an example of a data perspective join point, assume that the bank is interested to define security policies when the amount of an account balance is changed. These policies should therefore be defined when the balance attribute of the account is changed. The ‘balance’ attribute is a data perspective join point in this example. This join point could be defined using data visibility patterns. The data visibility patterns consist of eight patterns, which define the visibility of data for communication with the external environment [21 p. 357]. Therefore, they should be considered as join points, which could be defined for business process data perspective. These patterns are ‘Task Data’, ‘Block Data’, ‘Scope Data’, ‘Folder Data’, ‘Multiple Instance Data’, ‘Case Data’, ‘Workflow Data’ and ‘Environment Data’ [21 p. 366]. This kind of join point is called data signature.

R1.2. Data signature shall be possible to define in the data perspective.

As an example of a resource perspective join point, assume that the bank is interested to log activities of its clerks. These activities should therefore be captured when a clerk performs a task. The clerk is a resource perspective join point in this example. The join point in business process resource perspective should be defined for both human and non-human resources. These join points are called resource signatures.

R1.3. Resource signature shall be possible to define in the resource perspective.
4.2. **Pointcut**

Pointcut is a language which specifies a set of join points in business processes. The language should support some operations by which the join points could be expressed in a set. Binary operators such as ‘AND’ and ‘OR’ with the help of a unary operator like ‘NOT’ are good candidates to make such an expression language, like AspectJ.

Figure 56 illustrates an example by which a set of join points are selected using a pointcut expression. This expression defines a condition upon which a security policy could be defined for a banking business process. The condition specifies the situation in which the task name is either ‘Pay’ or ‘Receive’, and the person who performs the task is a ‘Clerk’, and the ‘Amount’ is between 10,000 and 30,000 EUR.

![Figure 56](image)

Therefore, the requirements regarding the pointcut language is defined as follows.

**R2.** Pointcut language shall support task, data, and resource signatures.

**R2.1.** Pointcut language shall support task signature.

**R2.2.** Pointcut language shall support data signature.

**R2.3.** Pointcut language shall support resource signature.

**R2.4.** Pointcut language shall support the operators AND, OR, and NOT.

4.3. **Advice**

An advice is a business process which models a cross-cutting concern. Each advice could be executed before, after or around the advised join point. The around advice should call the advised join point using `Proceed` task. `Proceed` task has an underspecification flexibility, which should be defined in the advice later. An advice could only meet `Proceed` tasks one time during its execution, because the functionality of the core-concerns should not be changed in the cross-cutting concerns. The core-concern process and related advices shall also support cancellation patterns.

**R3.** Concerns shall be defined as before, after or around advises.
### R3.1.
An advice shall be able to execute before an advised join point.

### R3.2.
An advice shall be able to execute after an advised join point.

### R3.3.
An advice shall be able to execute around an advised join point.

### R3.4.
The advised join point shall be invoked using a dummy task (called \textit{Proceed}).

### R3.5.
The cancellation patterns shall be supported by advices and core-concern process.

An advice contains the functionality of a cross-cutting concern, so it is a mandatory part of the core-concern. Therefore, the core-concern process could not continue if related advices are not executed. Advices shall be executed in parallel if they have the same order. It is also possible that advices have a different order.

### R3.6.
The core-concern process shall be suspended until related advices are executed.

### R3.7.
An advice shall have an order number.

### 4.4. Aspect

An aspect is a cross-cutting module which consists of one or several pointcuts which relates to one or several advices.

<table>
<thead>
<tr>
<th>R4.</th>
<th>Aspect shall be able to be defined to contain different cross-cutting concerns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4.1.</td>
<td>Aspect could have one or several pointcuts.</td>
</tr>
<tr>
<td>R4.2.</td>
<td>Aspect could have one or several advices.</td>
</tr>
<tr>
<td>R4.3.</td>
<td>Each advice shall be related to one or several pointcuts.</td>
</tr>
</tbody>
</table>

### 4.5. Weaving service

An old version of a process which contains several aspects will be designed using different processes in the aspect oriented BPM. One process represents the core-concern, while others represent cross-cutting concerns. Therefore, a mechanism should be defined to join these concerns with each other. This mechanism is called weaving service. Weaving service could perform at the design-time and at the run-time. The run-time weaving service add more value to the aspect oriented business process, because it allows more flexibility in defining and assigning concerns to a core-concern. The weaving service should define a mechanism to apply the appropriate ordering between different concerns. The same ordering mechanism as AOP could be applied (See Figure 13).

The precedence of merging different advices at the run-time is explained using two examples. Figure 24 demonstrates an advised task, called A. An aspect is related to the join point by a colored circle. This aspect contains three advices with different types such as around, after and before. All advices have the same order. The \textit{Proceed} task shows the place that the advised join point should be launched. The \textit{Proceed} task is a placeholder, so the whole process has underspecification flexibility at the run-time (See Figure 19). Aspect oriented business process supports two kinds of flexibility, change and deviation. The execution of this process will be equivalent to the process in Figure 25.
Figure 24 Same precedence advices
Figure 25 Flattened version of same order advices

Figure 26 illustrates a process like the previous example, but with different ordering number for each advice. The task F shall be executed first, but the task B and E should execute after task F, in parallel. Afterwards, the advised join point should be launched. Next, tasks C and D should be executed in parallel. Finally, Task G should execute to finish the process (See Figure 27).

Figure 26 Different precedence advices
Figure 27 Flattened version of different order advices

Therefore, the following requirements are considered for weaving service.

**R5.** A weaving service shall be defined to execute an aspect oriented business process.

**R5.1.** The weaving service shall allow assignment of aspects to core-concerns at run-time.

**R5.2.** The weaving service shall support passing of data between core- and cross-cutting concerns.

**R5.3.** The weaving service shall support execution of advices with different precedence.

**R5.4.** The weaving service shall support execution of advices with the same precedence.
5. Service Design

5.1. Introduction

In this chapter, a prototype model for aspect oriented service is designed and presented. YAWL Worklet service is chosen as a framework which is extended to design the new service. Worklet is an open-source service which is designed based on Service Oriented Architecture (SOA). Thus, it could be extended to support other WfMS as well. It also designed using CPN, so it has a formalized foundation. Therefore, the new service which is designed to be added to Worklet service could be extended for other WfMS as well. The same architecture which is used by Worklet service is utilized to find the aspect service.

5.2. Requirements Selection

A set of requirements are selected to design a CPN model prototype for aspect oriented service in YAWL. This model helps to realize this set of requirements and demonstrate how the aspect oriented service works. Selected requirements are discussed for different requirements’ categories separately.

5.2.1. Join Point

The Worklet service is based on capturing case or item constraints. Therefore, the task signature requirement, R1.1, could be defined using Worklet service. It is also possible to capture data through the process, but it could be done after defining task signature. Therefore, a lower level of data signature is supported, R1.2. Resource join points are not considered in designing this model, since the Worklet service does not support such a signature, R1.3.

5.2.2. Pointcut

Special approach is used by Worklet service to select join points. First, a task signature is selected using specified control-flow join point. Then, the selection could be done using the data signature. Although this approach applies some limits, it is a good prototype for building aspect oriented business process service. The data signatures could be expressed in combination with each other using AND, OR or NOT operations, R2.1, R2.2, R2.4. This point cut language does not also consider the resource signature in defining the selection process, because there is no resource join point there, R2.3.

5.2.3. Advices

Advices are considered as a process that may have or may not have a Proceed command. Therefore, it is possible to design around, before and after scenarios, R3.1, R3.2, R3.3, R3.4 and 3.6. The cancellation patterns, R3.5, and advice orders, R3.7, will not be considered in the designed model.

5.2.4. Aspect

The aspect requirements are fully supported using the Worklet service, so all requirements including R4.1, R4.2 and R4.3 are selected to be implemented in the artifact.
5.2.5. Weaving Service

Dynamic weaving that supports passing data through advices is modeled as an additional service in Worklet. Multiple advices could be run in parallel, R5.1, R5.2 and R5.4. The precedence of advices is not modeled since order numbers are not modeled in advices, R5.3.

5.3. Aspect Interface

The required set of events and methods which are used by aspect service are described as follows (See Figure 28).

- Work item constraints are used to capture required events to handle advised join points and the Proceed commands. These constraints enable definition of pointcuts for control-flow signatures. The isJP parameter shall be defined as an extension to a task to help distinguishing of an advised join point task from the Proceed task. There is also another parameter that specifies whether the raised event is a kind of preItemConstraint or postItemConstraint.

- CaseConstraint event enables the aspect service to trace cases. It determines the moment that each advice is finished. Therefore, aspect service could resume the core-concern to continue the process.

- SuspendWorkitem is necessary because the advised join point and the Proceed tasks should be suspended to fulfill the aspect oriented goal.

- UnsuspendWorkitem is used to continue suspended workitem to complete the process.

- ‘Launch’ is used to launch a new advice.

- GetInputParameters is used to get the parameters of a new advice. The parameters should be read before launching the aspect, so this method enables the service to get the parameters list.

- ForceCompleteWorkitem is a method that is used to skip the Proceed task. The Proceed task should be executed. Indeed, it is a placeholder for advised join point. Therefore, it should be forced to complete using this method.

- UpdateWorkitemData is used to update the workitem data both for advised task and for Proceed task. The data shall be transmitted through the core-concern and cross-cutting concerns. This matter performed using this method.
5.4. Formalization

The aspect service is described in this section with the help of a sample process (See Figure 29). Imagine the core-concern as a process containing three sequential tasks, A, B and C. Four advices are defined for Task B using a RDR. The first advice is executed around the advised join point, which contains two tasks. Task D is going to execute before the advised join point, while task E should be done after it. The second advice is a kind of after advice that has a task, named F. The third advice which is a kind of before advice has a task, called G. The last advice does not have any Proceed task that represents the execution of advised join point. Therefore, it should be executed before the advised join point, which represents an implicit before advice.

Figure 28 Required Aspect Interface

Figure 29 Aspect service starting point

Figure 30 shows the flattened version of the above example.
Aspect service CPN model is shown in Figure 31. The service is launched when an ItemConstraint event is raised. It means that the aspect service will be launched for each ItemPreConstraint in the core-concern process (See Figure 29). ItemConstraint place represents the occurrence of this event (See Figure 31). The ‘initial aspect’ transition would be enabled if the raised constraint does not belong to a Proceed task. This transition produces a token in reachedJoinPoint place, which contains necessary information for retrieving relevant RDR information. It also produces a token in saveAspectInfo place, which contains caseid and itemid. This separation is considered because the aspect service reuses the evaluation mechanism that is used by the Worklet service. The caseid is not used for evaluation, so it is saved in saveAspectInfo place to be merged with the evaluated token later. getPointcutInfo checks the availability of an RDR for the task and produces a token with the pre-defined related tree in pointcutRetrieved place.

If the token has not any tree, noTree consumes tokens in both pointcutRetrieved and saveAspectInfo places. If the raised constraint is a kind of itemPreConstraint then selectPointcut transition will be enabled, which is equivalent to evalTree transition that is defined by Adams M. in[22].
selectPointcut is a hierarchical transition. Figure 32 illustrates this sub-net. This sub-net evaluates a RDR and retrieves evaluated tree and actions that are defined to perform upon meeting defined criterion.
selectPointcut transition produces a token in handler place (See Figure 29). If there were no set of evaluated advices, then notFulfilled will be enabled. This transition consumes the token, and the process will finish. Otherwise, startAspect merge tokens in both handler and saveAspectInfo places and produce a token in AspectInfoWithData.

Figure 33 illustrates the weave aspect sub-net. This subnet contains five sub-nets which are connected with each other through ICore place. The colour set of ICore place contains two parts. First part determines the message that is passed through the nets. The second part constrains a list of message’s attributes and their values. All sub-nets except the Core are initialized upon receiving a token in their input places. As mentioned above, a token is produced in AspectInfo place, so launchAspect sub-net will be enabled.
Figure 34 shows launchAspect sub-net. launchAspect transition consumes a token from AspectInfo place and produces a token in Aspect place and five tokens in ICore place. The Aspect place contains tokens, which have information about aspects. The requests are passed through ICore place, and they contain the following messages: suspendWorkItem, getAdviceNumber, initAdvice and parentIDFlag. suspendWorkItem, getAdviceNumber and initAdvice are consumed by Core sub-net (See Figure 41). The engine transition consumes tokens that contain engine commands such as suspendWorkItem, launchCase, updateWorkItemData, unsuspendWorkItem, getInputParams and forceCompleteWorkItem. This transition also produces a token in ICore, which represents the response of YAWL Engine. It returns tokens with following messages for above commands: ackSuspendWorkItem, ackLaunchCase, ackUpdateWorkItemData, ackUnsuspendWorkItem and rspGetInputParams. It does not return any response for tokens, which contains forceCompleteWorkItem message (See Appendix A. Aspect Service CPN Declarations for detail information).

The token with initAdvice message is consumed by initAdvice transition of Core sub-net (See Figure 41). This transition produces a token in AdviceNumber and numJP places. These tokens specify how many advices exist in each aspect. Moreover, the token with getAdviceNumber message is consumed by getAdviceNumber transition of Core sub-net. This transition consumes a token from AdviceNumber and ICore places and produces a token in ICore place. Indeed, it responses to the requested command and retrieves the number of advices that an advice has. The produced token contains rspGetAdviceNumber message, which represents the response of such a query.

The token with the parentIDFlag message is not consumed by Core sub-net; instead, it will be consumed by finishAspect sub-net to finalize the process. enableLauncher transition of launchAspect sub-net will be enabled if the Core sub-net retrieves the number of advices in the aspect and suspends the workitem. This means that the advised join point is suspended. It produces ‘strToInt(#2param2)’ numbers of workitem id in AdvisedJP place. ‘strToInt(#2param2)’ represents the number of advices inside the aspect.

Figure 33 CPN: WeaveAspect
Figure 34 CPN: Launch Aspect

`initAdvice` transition will be enabled when advices are extracted from the aspect, and the advised join point is suspended, and the number of advice is retrieved. Afterwards, this transition could launch advices one by one. It consumes the token inside `ExclFlag` place and produces the token after it launched the advice, milestone pattern. It first produces a token with the `getInputParams` message. It also produces a token in `joinPointID` place that saves the workitem id. This identifier could be merged with launched case at the end of the process. It also produces a token in `adviceInfo` place, which will be consumed by `launchAdvice` transition upon receiving an `rspGetInputParams` message from Core sub-net. `launchAdvice` transition launches the advice as a new case by consuming a token from
adviceInfo and ICore places and producing a token in ICore place. The produced token will have launchCase message.

Finally, the saveAdviceState transition will be enabled when the new case is launched. It consumes a token from joinPointID and ICore places. The consumed token from ICore place is the acknowledged of launching the advice. This transition produces a token in ExclFlag and ICore places. The produced token in ExclFlag place enables sub-net to launch another advice. The produced token in ICore place contains setAdviceInfo message which will be used by setAdviceInfo transition of Core sub-net to save the information of advised join point and caseid of launched advice in adviceInfo place of Core sub-net (See Figure 41).

When all advices are run, the aspect service will not conduct any action. There might be two different scenarios, which could happen. If the launched advice contains Proceed task, the aspect service will wait to catch it (red dashed lines in Figure 35). However, if the advice does not have a Proceed task, or it does not meet a Proceed task during its execution, the aspect service will wait to catch the case complete constraint of it (implicit before advice in Figure 35).

The Proceed task is a kind of underspecification flexibility, so it is just used as a placeholder that shows the point that advised join point should be resumed. The advised join point will be unsuspended by aspect service when all advices are finished or reach to Proceed task.

If the advice does not meet any Proceed command during its enactment, proceedAdvisedJoinPoint and resumeAdvice sub-nets will never be executed (See Figure 33). It is because proceedCommand and AdviceInfo places will not have a token that enables these sub-nets. Therefore, finishAspect sub-net will be enabled when the case of an advice is completed. In this case, a token would appear in caseConstraint place (See Figure 31). remCase transition will consume this token and produce a token in caseComp place (See Figure 31 and Figure 33). Therefore, finishAdvice transition of finishAspect sub-net will be enabled (See Figure 40). finishAspect sub-net will be explain later in this chapter.

If the advice meets any Proceed command during its enactment, it should be suspended. Then, the advised join point should be continued when all related advices met the Proceed task or finished. The Proceed transition in AspectService will be enabled if an advice meets a Proceed task. This transition consumes a token from ItemConstraint place and produces a token in proceededJoinPoint place. proceededJoinPoint place enables the proceededAdvisedJoinPoint sub-net (See Figure 36).
The prepare transition will be enabled when a token is placed in proceedCommand place. This transition produces three tokens in ICore place and a token in caseInfo place. The produced token in proceedCommand place does not contain any information regarding advised join point. Thus, a request to retrieve related advised join point is sent to ICore place with the help of a token with getAdviceInfoByCase message. The getAdviceInfo transition of Core sub-net will be enabled by
consuming this token. It will consume the related token from adviceInfo place; then, it produces a token in adviceInfo place and ICore place. The produced token in adviceInfo place is the same token as the token that this transition consumed, but the produced token in ICore place will have an rspGetAdviceInfoByCase message.

Moreover, the advice case will be suspended by a token that the prepare transition produced in ICore place (See Figure 36). The suspendWorkItem is a kind of engine message, so this token will be consumed by engine transition of Core sub-net. This transition will produce a token with ackSuspendWorkItem which represents that the advice is suspended. The other token that is produced by prepare transition has saveJoinPointID message. This token is consumed by setjoinPoint transition of Core sub-net. This transition produces a token in JoinpointInfo place and the other in CaseID place. The produced token in CaseID place will be consumed by setInitializedAdvice transition (See Figure 41). This transition also consumes a token from adviceInfo place and produces a token in adviceInfo place and in Advice place. Indeed, this transition relates the advice case id with the aspected join point and put them into Advice place to be queried later.

Next, updateData transition in the proceedAdvisedJoinPoint sub-net will be enabled when getAdviceInfoByCase transition of Core sub-net returns the itemid of advised join point. This itemid is used to update the core-concern data. updateData transition updates the data by producing a token in ICore place. This token contains the updateWorkItemData that was described before. This transition also produces a token in adviceInfo place. This token contains the advised join point id and information regarding the suspended advice. Moreover, this transition produces a token in proceededAdvisedJoinPoint place. The setAdviceNum transition consumes tokens, which represent advices of an advised join point from this place and a token from ICore place, which contains rspGetAdviceNumber message and produces a token in adviceNum place. The produced token contains information about the number of advice that an aspect has.

Afterwards, setAdviceInfo transition will be enabled when it could consume two tokens from ICore place that contains ackSuspendWorkItem and ackupdateWorkItem messages. This transition reduces the number of advice in adviceNum place in each execution. When the number of advices in an aspect becomes zero, proceedAdvisedJoinPoint transition will be enabled. This transition produces three tokens in ICore place such as unsuspendWorkItem, resumeFlag and getAdvicenumber. Therefore, the engine transition of Core sub-net could continue the core-concern. resumeFlag will be used in next sub-net, finishAspect.

There is an exceptional condition that an aspect contains implicit before advice. In such a case, setAdviceNum transition of proceedAdvisedJoinPoint sub-net could not be enabled because the number of tokens in proceededAdvisedJoinPoint place is less than the number of advices that are defined for aspect. The proceededAdvisedJoinPoint place represents the number of Proceed tasks that has been met by aspect service. In this case, the implicit advice will be finished, so the finishAspect sub-net will be enabled. This sub-net reduces a number of advices in the aspect by enabling removeCase transition of Core sub-net. Afterwards, the updateAdviceNumber transition of Core sub-net will reduce the number of advice for the aspect. This means that the token that contains rspGetAdviceNumber as a message will be modified. Therefore, setAdviceNum transition will be enabled, and the sub-net could continue.
Figure 37 illustrates the situation in which the advised join point was unsuspended and the workitem is finished. The advised join point should be suspended, and the data for all existing advices should be updated. Next, advices should be unsuspended, and the Proceed task should be forced to complete. The Proceed task should be completed since it is just a placeholder, and the specification is not defined.

Aspect service captures finishing advised join point when initialAspect transition is enabled. This transition is enabled when an item constraint occurs. The process will be exactly like when the workitem has been started, but selectPointcut sub-net will not be enabled. Instead, resumeAspect transition will be enabled that consumes tokens from pointcutRetrieved and saveAspectInfo places. It also produces a token in AdvisedJPComp place (See Figure 31). This place enables resumeAdvice sub-net (See Figure 38).

The finishAspect transition of resumeAdvice sub-net consumes three tokens from ICore place and a token from AdvisedJPComp place. These three tokens are ackUnsuspendWorkItem, rspGetAdviceNumber and resumeFlag. The first mentioned token guarantees the suspension of advised join point. The second one gives information about the number of running advices for the advised join point. The last one generates handling advices which have Proceed command. finishAspect transition produces a token in proceededAdvisedJoinPoint place. This token will be consumed by setSuspension transition when the advised join point is suspended. The finishAspect transition also produces tokens for each running advice in advisedJoinPointInfo. These tokens will be used to update the advices’ data. Moreover, finishAspect transition produces tokens in ICore place with suspendWorkItem and getAdviceInfoByItemID messages. The first token is for suspending advised join point. The last set of tokens with the getAdviceInfoByItemID messages is used by Core sub-net to retrieve advices of the aspect. The getAdvByItemID transition of Core sub-net uses these tokens to retrieve all advices.
The `getAdvices` transition will be enabled when the response of `getAdviceinfoByItemID` is available. This transition consumes the `getAdviceinfoByItemID` token and produces a token in `adviceInfo` and `advisedJP` places. Therefore, `updateAdviceData` transition will be enabled. This transition requests:

- `id` (param `ackUnsuspendWorkItem`, param3::params3)
- `(id,d)` (param `rspGetAdviceNumber`, param::params)
- `(resumeFlag, param2::params2)` (param `updateAdviceData`, ["("case", c)"])
- `(id,d)` (param `getJoinpointID`, ["("case", c)"])
- `id` (param `ackSuspendWorkItem`, param::params)
- `id` (param `unsuspendWorkItem`, param::params)
- `strToInt(#2param)` (param `updateWorkItemData`, ["("item", id), ("data", PLtoS(d))"])
to update the advice data by producing a token with `updateWorkItemData` message in `ICore` place. It also makes a request to get the id of the `Proceed` task by producing a token with `getJoinPointID` message in `ICore` place. This token is consumed by `getJoinPoint` transition of `Core` sub-net. `getJoinPoint` transition gets the advised join point info by consuming a token from `JoinPointInfo` place. It also gets the advice info by consuming a token from `adviceInfo` place (See Figure 41). Finally, this transition produces a token in `adviceInfo` place and a token that has the id of the `Proceed` task in `ICore` place.

The `unsuspendAdvice` transition will be enabled when the `Proceed` task data synchs with advised join point. This transition gets the acknowledgment of this update, so it requests the engine to continue the `Proceed` workitem by producing a token with `unsuspendWorkItem` message in `ICore` place. When the `Proceed` workitem is suspended and the information of `Proceed` workitem is retrieved, `forceCompleteProceedTask` transition will be enabled. This transition requests engine to force complete the `Proceed` workitem by producing a token with `forceCompleteWorkItem` message in `ICore` place.

As a result, the aspect service should wait until all advices are finished. Figure 39 shows the situation in which all advices are finished. The data advised join point should be synched with the data of advices. Next, the aspect service should continue the advised join point.

![Figure 39 Continuing the core-concern](image)

Figure 40 shows `finishAspect` sub-net. This sub-net is responsible for performing described actions. This sub-net will be enabled when a `casePostConstarint` is aroused and `remCase` transition of `aspectService` net produces a token in `caseComp` place (See Figure 31). This place is an input place for `finishAspect` sub-net (See Figure 40). `finishAdvice` transition in `finishAspect` sub-net will be enabled upon receiving a token in `caseComp` place. This transition produces a token in `adviceData`, `adviceCaseID` and `ICore` places. The produced token in `ICore` place contains `getAdviceInfoByCase` message. If the case is a core-concern case, `RemoveAdvisedCaseInfo` transition will consume all of these tokens. `parentIDFlag` that is produced in `launchAspect` sub-net helps this sub-net to recognize the core-concerns cases.

However, if the finished case is an advice case, the Core sub-net will consume the produced token in `ICore` place and produce another token there with `rspGetAdviceInfoByCase` message. The `weaveAdvice` transition will be enabled upon receiving this token, so the `caseid` of advice and the `itemid` of advised join point will be considered in a token which this transition produces in `adviceInfo` place (See Figure 40).
Therefore, \textit{updateAdvisedJoinPointData} transition will be enabled. This transition produces two tokens in \textit{ICore} place, which contains \textit{updateWorkItemData} and \textit{removeAdvice} messages. The token which contains \textit{updateWorkItemData} will be consumed by engine transition of \textit{Core} sub-net (See Figure 41).
This transition produces a token in ICore place, which contains ackUpdateWorkItemData message. Tokens with ackUpdateWorkItemData and removeAdvice messages are consumed by removeCase transition of Core sub-net. This transition also consumes a token from adviceinfo place to decrease the number of advices, which are persisted in adviceNumber place. When the number of advices of an
aspect becomes zero, finalizeAspect transition will be enabled. This transition produces a token in finishedWI place. This token is consumed by one of the finalized transitions to consume all remaining tokens in the net. If the aspect contains advices that has Proceed command, the finalize transition with the \( [n<>0] \) guard will be enabled. Otherwise, the other transition will be enabled to consume tokens.

The finalizeAspect transition also produces a token in ICore place with noMoreAdvice message. This token is consumed by finalizeAspect transition of finishAspect sub-net (See Figure 40). This transition produces a token in ICore place with unsuspendWorkItem message. The engine transition of Core sub-net consumes this token and produces a token with ackUnsuspendWorkItem message. This means that the advised join point is unsuspended and the process could continue. Finally, the token with ackUnsuspendWorkItem message will be consumed by finish transition of finishAspect sub-net.

5.5. Artifact Evaluation

As discussed in the Research Method section, interactive simulation has been selected as a method for evaluating the artifact. To perform interactive simulation, it is required to design scenarios. The scenarios should cover all possible behavior of the model. To fulfill all possible behaviors, it is important to specify the number of advices that a scenario should contain. It is important to have more than one advice in a scenario to check the synchronization features of weaving requirements. However, the process to synchronize advices is not changed by adding more advices to an aspect. Thus, two advices are both necessary and sufficient to evaluate the artifact. Furthermore, each advice could either contain a Proceed command or not, so all possible combinations of this probability should be considered to define scenarios. Therefore, three scenarios are designed to cover all possible cases.

1. The first scenario contains two advices, each of which has a Proceed command.

2. The second scenario contains one advice that has a Proceed command and one advice which has not.

3. The third scenario contains two advices that do not contain a Proceed command.

These scenarios are detailed in the following subsections. For simulating a CPN model, tokens’ distribution in the model and token values are needed to be specified. The values used in our simulation are documented in ‘Appendix B. Evaluation Data CPN Declarations’. In order to be able to repeat our simulation, we have documented the simulation steps, we run in ‘Appendix C. Evaluation Scenarios’. The steps are listed using binding elements, according to the documentation format used in [9 p. 22]. Each binding element contains two parts. The first part represents the name of transition that is executed, and the other part represents the data that shows the token, which should be consumed.

5.5.1. Scenario 1

This scenario contains two advices, each of which has a Proceed command. The first 28th binding elements (See Appendix C) show the initialization of the artifact. During these steps, the service stops the advised join point, itemID1, and launches both advices. After the execution of the 28th steps, the model reaches the state illustrated in Figure 42.

When the two advices are launched, the aspect oriented service waits to get notified about the Proceed commands. Then, the two advices are suspended and the advised join point (in the core-concern model) is unsuspended. Figure 43 illustrates the state after the execution of step 50 (Appendix C).
When the advised join point is finished, it should be suspended. Afterwards, the advices should be unsuspended and the Proceed command should be forced to be completed (see binding elements 51 until 74 in Appendix C). Figure 44 illustrates the state and the required actions in the process at this stage.

When the advices are finished, the core concern should continue. Figure 45 illustrates the state of the model after the execution of step 75 (Appendix C). The scenario is finished successfully, and all tokens are consumed.

5.5.2. Scenario 2

This scenario contains one advice that has a Proceed command and one advice which has not. The first 28th binding elements (See Appendix C) show the initialization of the artifact. During these steps, the service stops the advised join point, itemID10, and launches both advices. After the execution of the 28th steps, the model reaches the state illustrated in Figure 46.
When the two advices are launched, the aspect oriented service waits to get notified about the `Proceed` commands. However, the engine could not get any `Proceed` command for the advice which does not have any `Proceed` command. Therefore, the service will wait to catch the end constraint of this case. Afterwards, the service will continue the advised join point. Figure 47 illustrates the state after the execution of step 49 (Appendix C).

When the advised join point is finished, it should be suspended. Afterwards, the remaining advice should be unsuspended. The `Proceed` command should be forced to be completed. (See binding elements 50 until 64 in Appendix C). Figure 48 illustrates the state and the required actions in the process at this stage.

When the advice is finished, the core concern could continue the work. Figure 49 illustrates the state of the model after the execution of step 65 (Appendix C). The scenario is finished successfully, and all tokens are consumed.

### 5.5.3. **Scenario 3**

This scenario contains two advices that do not contain a `Proceed` command. The first 28th binding elements (See Appendix C) show the initialization of the artifact. During these steps, the service stops the advised join point, `itemId100`, and launches both advices. After the execution of the 28th steps, the model reaches the state illustrated in Figure 50.
When the two advices are launched, the aspect oriented service waits to get notified about the \textit{Proceed} commands. However, the engine could not get any \textit{Proceed} command for the advices. Therefore, the service will wait to catch the end constraint of both advices. The service will continue the advised join point afterwards. (See binding elements 29 until 47 in Appendix C). Figure 51 illustrates the state and the required actions in the process at this stage.

When the advices are finished, the core concern could continue the work (See binding elements 48 and later in Appendix C). All advices are finished, so the main concern should continue to the end. The scenario is finished successfully, and all tokens are consumed.

5.5.4. \textbf{Conclusion of Evaluation}

The produced artifact is evaluated using three design science evaluation methods, i.e. simulation, scenarios and white-box testing. The simulation is performed using CPN Tools interactive simulation. This simulation is led by scenarios that cover all possible situations. This combination provides a means to check the functionality of the designed artifact, which is a white-box testing in the design science. This evaluation justifies the addition to the knowledge of the aspect oriented service in BPM area in the form of formalisms. Therefore, it proofs the rigorousness of this research. This evaluation could not proof the application of the artifact in the appropriate environment, which is required to show the relevancy of the research. Thus, a case study will be performed to evaluate the designed artifact for proofing its relevancy. This evaluation is called validation, which is described in the next chapter.
6. Validation

In this section, the result is validated by applying it on a business case study. The case study contains two business processes from the financial domain namely, change asset deal and deal for speculation. These processes are taken from a bank which has more than 1,000 branches. The processes are described in the subsequent sections. Detailed information about them was derived through an interview with a domain expert.

6.1. Change asset deal

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 the marketable securities. The Change asset deal process handles deals, which aim to change the account balances of the bank. Therefore, this process makes a deal to exchange an amount of money from one currency to another in the banking domain.

Figure 2, in the Introduction section depicted the first part of this business process. The whole process is shown in Figure 52. As mentioned earlier, back office employee fills in a position sheet. Then the general manager confirms or denies it. Then the sheet is archived. If the sheet is denied, the process is terminated. If the position sheet is approved, a junior dealer makes the deal and fills in a deal slip. Next, a chief dealer and the general manager sign the deal slip, and the deal slip is archived.

After the deal slip has been archived, two parallel sets of activities are performed. On one hand, the dealt amount of money 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. In parallel, the dealt amount of money 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 makes 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.

As can be noted, there are two aspects in this business process, logging and security. The impact of the aspect oriented modeling is studied by separating the logging concern. Figure 54 shows the result of this separation. The logging concern is shown as a circle labeled with ‘L’ (for Logging). Basically, the task Archive has been extracted from the model and captured through a separated process (i.e. advice). The dummy task Proceed indicates the place where the advised join-point should be launched. In the example, it will be like a placeholder for the tasks Confirm, Sign, Control, and Register Voucher.

As can be seen in Figure 54, applying the aspect oriented principle reduces the overall size of the model [23] from 25 to 22 tasks (note that the security concern has not yet been separated) Furthermore, it reduces the size of the core-concern model from 25 to 20. Hence, the model becomes simpler.
Figure 5.4 Separation of logging concern from change asset deal process
6.2. Deal for speculation

Deal for speculation is a kind of deal that dealers make in the bank. The process is shown in Figure 53. Although the goal of a dealer is to make a profit, it is possible that he loses money. Therefore, there is a limit on the amount of money that junior and chief dealers can deal. As long as a junior dealer has not exceeded his limit, he does not need permission for doing a deal. Otherwise, he needs permission from the chief dealer, so he sends a request. If the chief dealer denies the request, the process ends.

If the chief dealer approves the deal, the junior dealer can proceed, which he does by opening a position. However, if the amount of the deal exceeds the chief dealer’s limit, permission is also needed from the general manager. If the general manager denies the deal, the process finishes. Conversely, if the general manager approves the request, the junior dealer can proceed by opening a position (See Figure 53). Afterwards, the junior dealer makes the deal. From the task Deal onwards, the process continues in the same as the change asset deal process.

Figure 55 shows the aspect oriented design of this process. Two concerns can be identified: a logging concern and a security concern. In this example, aspect orientation is applied to deal with both concerns. This results in the introduction of two individual processes complementing the core-concern process. The annotations ‘S’ and ‘L’, representing the pointcuts, are used for denoting the security and logging concerns, correspondingly. It should be noted that the logging concern is the same as in the previous process and can therefore be reused from there.

This example demonstrates that an advice can be a complex process model (see the security concern in Figure 55). In this case, a Ripple Down Rule (RDR) is defined to control the initiation of the advice. The advice is launched when the rule is evaluated as true. In the example, the RDR expresses that the advice High Control shall be launched as soon as a junior dealer exceeds his limit (JD.Limit<Amount). Otherwise, the security advice will be not executed, instead the core process will continue as specified.

Note further, that a second RDR has been defined, this time for handling rejections. The rule specifies that if the chief dealer or the general manager denies a request (i.e. NotConfirmed), the Worklet exception service will terminate and remove the case (i.e., removeCase).

As can be seen in Figure 55, applying the aspect oriented principle reduces the overall size of the model [23] from 29 to 25 tasks. It also reduces the size of the core-concern model [23] from 29 to 19. Even in this case, the model becomes simpler. Furthermore, it should be noted that because the logging concern appears in both ‘deal for speculation’ and ‘change asset deal’ processes, it should not be double counted in measuring the overall impact of the application of aspect orientated modeling.

As a matter of the fact, also the security concern appears in both processes. However, due to limitation in the approach, it was not possible to separate it from the ‘change asset deal’ process in parallel with the logging concern, and it was only partially separated from the ‘deal for speculation’ process (see for instance, the tasks Sing in Figure 55 are part of this concern, but remain in the core-concern process). These limitations will be discussed in next section.
Figure 55: Separation of logging and security concerns from the deal for speculation process.
7. Analysis

The artifact presented in this thesis is a formal model that provides a foundation for aspect oriented BPM. The artifact is validated using a real-life problem in the banking domain, and it is discussed to consider how the complexity is reduced based on the given artifact. The artifact was developed based on a selected set of requirements. Due to time and technology limitations, some requirements were left outside the scope of this work. The following table shows a summary of the requirements with an indication of whether they are fulfilled (F), partially fulfilled (P) or not fulfilled (N). For the partially and not fulfilled requirements, the table also shows the reason, i.e. technical (Tech) or time limitation (Time). In this section, the impact of these limitations on the result is discussed. This discussion will also outline directions for future work.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Requirement</th>
<th>Status</th>
<th>Limit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Join points shall be possible to define for control-flow, data and resource perspectives.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.1</td>
<td>Task signature shall be possible to define in the control-flow perspectives.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R1.2</td>
<td>Data signature shall be possible to define in the data perspective.</td>
<td>P</td>
<td>Tech</td>
</tr>
<tr>
<td>R1.3</td>
<td>Resource signature shall be possible to define in the resource perspective.</td>
<td>N</td>
<td>Tech</td>
</tr>
<tr>
<td>R2</td>
<td>Pointcut language shall support task, data, and resource signatures.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.1</td>
<td>Pointcut language shall support task signature.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R2.2</td>
<td>Pointcut language shall support data signature.</td>
<td>P</td>
<td>Tech</td>
</tr>
<tr>
<td>R2.3</td>
<td>Pointcut language shall support resource signature.</td>
<td>N</td>
<td>Tech</td>
</tr>
<tr>
<td>R2.4</td>
<td>Pointcut language shall support the operators AND, OR, and NOT.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>Concerns shall be defined as before, after or around advises.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3.1</td>
<td>An advice shall be able to execute before an advised join point.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3.2</td>
<td>An advice shall be able to execute after an advised join point.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3.3</td>
<td>An advice shall be able to execute around an advised join point.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3.4</td>
<td>The advised join point shall be invoked using a dummy task (called Proceed).</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3.5</td>
<td>The cancellation patterns shall be supported by advices and core-concern process.</td>
<td>N</td>
<td>Time</td>
</tr>
<tr>
<td>R3.6</td>
<td>The core-concern process shall be suspended until related advices are executed.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R3.7</td>
<td>An advice shall have an order number.</td>
<td>N</td>
<td>Time</td>
</tr>
<tr>
<td>R4</td>
<td>Aspect shall be able to be defined to contain different cross-cutting concerns.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4.1</td>
<td>Aspect could have one or several pointcuts.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R4.2</td>
<td>Aspect could have one or several advices.</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R4.3</td>
<td>Each advice shall be related to one or several pointcuts.</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
R5 | A weaving service shall be defined to execute an aspect oriented business process. |
R5.1 | The weaving service shall allow assignment of aspects to core-concerns at run-time. | F -- |
R5.2 | The weaving service shall support passing of data between core- and cross-cutting concerns. | F -- |
R5.3 | The weaving service shall support execution of advices with different precedence. | N Time |
R5.4 | The weaving service shall support execution of advices with the same precedence. | F -- |

Table 1 Requirement Analysis Table

7.1. Impact Analysis

The impacts of the requirements that are not fully satisfied are discussed as follows.

7.1.1. Partial definition of data signature and join point

Requirements R1.2 and R2.2 are identified in order to support definition of data signature in the pointcut language. The artifact presented in this thesis is built based on the Worklet service. The Worklet service does not support direct definition of data signatures. Therefore, these two requirements could not be fulfilled due to technical limitations inherited from the Worklet service. However, constraints on data could be defined based on workitem or case constraints. This implies that the data signatures are mapped to the control flow signatures.

Figure 56 illustrates this mapping using an example. This example shows that the data signature could be defined using task signatures in the Worklet service. This example shows a situation in which a bank needs to define a security policy when the balance of an account becomes more than 10,000 EUR and less than 30,000 EUR. This constraint could not be defined in the Worklet service directly, but it is possible to define this constraint for each task. This means that many task signatures should be defined to fulfill the definition of one data signature, which results in scattering join point definitions over the whole business processes (See Figure 56).

Therefore, it is still possible to extract cross-cutting concerns from the core-concern of a business process by partially fulfilling these requirements, so the goal of this thesis is achieved. However, it makes the development, management and maintenance of business processes harder, because for changing a data signature several task signatures should be altered. The full support of this requirement is proposed as future work to overcome this problem.
7.1.2. Dismissing resource signature and join point

Requirements R1.3 and R2.3 are included to support definition of resource signature in the pointcut language. As the Worklet service does not support the definition of resource signatures, the developed artifact could not fulfill these requirements due to technical limitations. This limitation affects the result in a way that the artifact could not separate cross-cutting concerns, which are related to resource signatures. Further work is required along the lines proposed in [16] so that the approach can deal with this limitation.

7.1.3. Dismissing cancelation feature

Requirement R3.5 is not considered because of time limitation. Implementing this requirement enables the model to handle scenarios in which one of the concerns is cancelled. If one concern is cancelled, the whole related concerns should also be cancelled. However, this artifact does not support this behavior.

7.1.4. Dismissing precedence feature

Requirements R3.7 and R5.3 should be fulfilled so that an aspect oriented service can support advice precedence feature. They are not considered to be implemented in this thesis because of time limitation. The impact of dismissing these requirements on the result is investigated below.

In order to illustrate the limitation, an integrated process of the ‘Change asset deal’ and ‘Deal for speculation’ processes are used. This process is depicted in Figure 57. The variation processes ‘Change asset deal’ and ‘Deal for speculation’ were integrated, because they share many similarities and because the company has long time been interested in considering them as one integrated process. However, until now, the company has not been able to deal with the complex solution, which such integration resulted in. The tasks associated to different aspects are coloured differently. The result of applying aspect orientation is shown in Figure 58. This example demonstrates how the aspect oriented approach is meant to work when all the requirements are fulfilled.
Figure 57 Integrated model of 'Change asset deal' and 'Deal for speculation' processes.
Figure 58 illustrates the application of advice order and weaving precedence requirements (R3.7 and R5.3) on the deal processes. Similarly to the solutions in Figure 54 and Figure 55 the logging concerns (L) and the security concern (S7) are extracted from the core-concern model (see the bottom right hand side corner of Figure 58). In addition, however, the security concern contains six more advices that are extracted from the main process (S1-S6). This is possible to do due to the waving and order requirement have been considered. The order enables the specification of precedence between the advices. This is shown in Figure 58 by arrows under the annotations for the advices. An implementation of the waving would imply that such orders can be interpreted and executed.

Compared to the process in Figure 57 where aspect orientation is not applied and the model contains 32 tasks, the model for the core concern in Figure 58 contains only 13 tasks. Hence the application of aspect oriented modeling did indeed reduce the complexity of the example model. As this example demonstrates, if requirements R3.7 and R5.3 were fulfilled, the complexity of a model could potentially be further reduced, compared to the result we achieve after applying the designed artifact. Therefore, a future extension of the proposed artifact should fulfill these requirements.

Finally, it should be pointed out that in Figure 58 there are three security advices, which are similar to each i.e. S2, S3 and S4. In fact, these advices are exactly the same except the resource who is responsible for performing the task. Therefore, it would be nice to be able to present them through one advice-model. This could be achieved if the resource could be passed from the core-concern to the cross-cutting concern. In fact, this is a new requirement which can be added to the list of requirements specified in Requirements Definition section. The fulfillment of this requirement opens up a new area of research, i.e. the support for resource passing between processes in BPMS.
8. Concluding Remarks

8.1. Conclusions and discussions

In the work presented here, the foundation for Aspect Orientated Business Process Management was developed and described. The result is a formal model defining the semantics of process models for which the cross-cutting concerns have been separated from the core concerns. This formal model can be implemented as a service and extend the functionality of Business Process Management Systems by enabling: (i) the modeling of cross-cutting concerns, e.g. organizational security and privacy policy, independently from the business processes they have impact on, and (ii) when relevant, the execution of the specified concerns inside the related business processes.

The formal model was developed to fulfill a selected set of requirements. It was defined in Coloured Petri Nets (CPN) and implemented in CPN Tools. The model was evaluated using three design science methods, i.e. simulation, scenarios and white-box tests. The results of these evaluations were successful.

It is postulated that Aspect Oriented Business Process Management will provide a means for reducing complexity of business process models. To test this, we applied the model we had developed in a practical case study. The case study investigated two banking business process, which were modeled both in a traditional way and in an aspect oriented way (enabled through the developed formal model). Domain knowledge about the processes was acquired through an open-ended interview with a domain expert. The case study showed that aspect oriented business process modeling can significantly reduce the complexity of a process model.

The work presented in this thesis, extends the related work in the Business Process Management area, i.e. AO4BPMN and AO4BPEL. It extends the results of AO4BPMN by providing a formal model, which when implemented as a service will also provide an execution environment. It extends the results from AO4BPEL by providing both a foundation for aspect oriented business process management based on a graphical notation, and a formal (in addition to executable) semantics.

8.2. Future work

There are some future researches, which should be conduct to continue the result of this research, which is described as follows. There is a need to perform a research on:

1. Defining resource join point.
2. Defining data signatures in pointcut language.
3. Defining resource signatures in pointcut language.
4. Considering precedence in the formalization of the aspect oriented service in BPM.
5. Defining a mechanism to pass resource information among core-concern and cross-cutting concern. This research could also be performed for the same issue in Worklet service.
6. Evaluating resource patterns for aspect oriented service or Worklet service.
7. How to deal with cancellation scenarios, which might happen in an aspect oriented service.
References

Appendixes

Appendix A. Aspect Service CPN Declarations

(* Standard priorities *)
val P_HIGH = 100;
val P_NORMAL = 1000;
val P_LOW = 10000;

(* Standard declarations *)
colset UNIT = unit;
colset INT = int;
colset BOOL = bool;
colset STRING = string;

(* colset declarations *)
colset ID = STRING;
colset SPECID = ID;
colset CASEID = ID;
colset TASKID = ID;
colset ITEMID = ID;
colset CASEIDxITEMID = product CASEID * ITEMID;
colset PARAM = product STRING * STRING;
colset PARAMS = list PARAM;
colset DATALIST = PARAMS;
colset WIR = product SPECID * CASEID * TASKID * ITEMID * DATALIST;
colset evITEMCONSTRAINT = product WIR * DATALIST * BOOL * BOOL;
colset EXLEVEL = with exCase | exItem ;

(* Service-side Declarations *)
colset EXTYPE = with CasePreConstraint | CasePostConstraint |
ItemPreConstraint | ItemPostConstraint |
ConstraintViolation | ItemAbort | ResourceUnavailable |
TimeOut | CaseExternal | ItemExternal | Selection |
Aspect | AspectPreConstraint | AspectPostConstraint;
colset TREEID = product SPECID * TASKID * EXTYPE;
colset TREEPARAMS = product EXLEVEL * ID * TREEID * DATALIST;
colset ACTION = with restart | fail | complete |
suspend | continue | remove |
compensate | select | advise;
colset TARGET = STRING;
colset PRIMITIVE = product ACTION * TARGET ;
colset NODE_ID = INT;
colset COND = STRING;
colset CSTONE = PARAMS;
colset EXLET = list PRIMITIVE;
colset CHILD_NODE_ID = NODE_ID;
colset RNODE = product NODE_ID * COND * EXLET * CSTONE * CHILD_NODE_ID *
CHILD_NODE_ID;
colset TREE = list RNODE;
colset TREEMAP = product TREEID * TREE;
colset TREELIST = list TREEMAP;
colset IDxTREExDATA = product ID * TREE * DATALIST;
colset ASPECTTYPE = with preAspect | postAspect;
colset IDxASPECTTYPE = product ID * ASPECTTYPE;
colset IDxEXLETxDATA = product ID * EXLET * DATALIST;
colset IDxTREExDATAxASPECTTYPE = product ID * TREE * DATALIST * ASPECTTYPE;
colset CASEIDxDATA = product CASEID * DATALIST;
colset ITEMIDxCASEIDxITEMIDxDATA = product ITEMID * CASEID * ITEMID * DATALIST;
colset CASEIDxITEMIDxDATA = product CASEID * ITEMID * DATALIST;
colset CASEIDxITEMIDxEXLETxDATA = product CASEID * ITEMID * EXLET * DATALIST;
colset ITEMIDxEXLETxDATA = product ITEMID * EXLET * DATALIST;
colset ITEMIDxDATA = product ITEMID * DATALIST;
colset evCASECONSTRAINT = product SPECID * CASEID * DATALIST * BOOL;
colset CASExCSPEC = product CASEID * SPECID;
colset CASExDATA = product CASEID * DATALIST;
colset RNODExEVAL = product RNODE * BOOL ;
colset CMD = with none
| initAdvice
| setAdviceInfo           |
| getAdviceInfoByCase    | rspGetAdviceInfoByCase  |
| getAdviceInfoByItemID  | rspGetAdviceInfoByItemID |
| getAdviceNumber        | rspGetAdviceNumber      |
| forceCompleteWorkItem  |
| suspendWorkItem        | ackSuspendWorkItem      |
| launchCase             | ackLaunchCase           |
| updateWorkItemData     | ackUpdateWorkItemData   |
| saveJoinpointID        | getJoinpointID          | rspGetJoinpointID       |
| unsuspendWorkItem      | ackUnsuspendWorkItem    |
| removeAdvice           |
| noMoreAdvice           |
| parentIDFlag           | resumeFlag              |
| getInputParams         | rspGetInputParams       |

colset COREMSG = product CMD * PARAMS;

colset ITEMIDxNUM = product ITEMID * INT;

colset CASEIDxITEMIDxPRIMITIVExDATA = product CASEID * ITEMID * PRIMITIVE * DATALIST;

colset ADVICE = TARGET;

colset ADVICExDATA = product ADVICE * DATALIST;

colset ITEMIDxADVICExDATA = product ITEMID * ADVICE* DATALIST;

colset ITEMIDxADVICE = product ITEMID * ADVICE;

colset IDxPARAMS = product ID * PARAMS;

colset PHIDxCID = product ITEMID * CASEID;

(* variable declarations *)

var str: STRING;

var n: INT;

var s: SPECID;

var c: CASEID;

var t: TASKID;

var i, id: ITEMID;

var d, cd: DATALIST;

var pre, isJP: BOOL;
var xlv : EXLEVEL;
var tid : TREEID;
var tList : TREELIST;
var aType : ASPECTTYPE;
var tree : TREE;
var xlt : EXLET;
var node, root, prevTrue : RNODE;
var e: UNIT;
var nid: CHILD_NODE_ID;
var cmd : CMD;
var params, params2, params3: PARAMS;
var param, param2, param3: PARAM;
var msg: COREMSG;
var pr : PRIMITIVE;
var adv: ADVICE;

(* function declarations *)
fun isEngineCommand(cmd:CMD) =
if cmd=suspendWorkItem then true
else if cmd=launchCase then true
else if cmd=updateWorkItemData then true
else if cmd=forceCompleteWorkItem then true
else if cmd=unsuspendWorkItem then true
else if cmd=getInputParams then true
else false;
fun getRsp(m:CMD) =
if m=suspendWorkItem then ackSuspendWorkItem
else if m=launchCase then ackLaunchCase
else if m=unsuspendWorkItem then ackUnsuspendWorkItem
else if m=updateWorkItemData then ackUpdateWorkItemData
else if m=getInputParams then rspGetInputParams
else none;
fun response(m:CMD, q:PARAMS) =
if getRsp(m)=none then empty
else 1 `(getRsp(m), q);

fun repl(p:PARAM, []:PARAMS) = [] |
repl(p, h::pl) = if #1p = #1h then p::pl else h::repl(p, pl);

fun upData([], q:PARAMS) = q |
upData(p::pl, q) = repl(p, upData(pl, q));

fun PtoStr(p:PARAM) = "(" ^ #1 p ^ "," ^ #2 p ^ ")";
fun PLtoS([], :PARAMS) = "" |
PLtoS(param::params) = PtoStr(param) ^ PLtoS(params);

fun getCond(n:RNODE) = #2n;

fun getNode(id:NODE_ID, []:TREE) = [] |
getNode(id, n::t) = if #1n = id then [n] else getNode(id, t);

fun eval(c:COND, d:DATALIST) = if c="true" then true else false;

fun getExlet(n:RNODE) = #3n;

fun getChildID(n:RNODE, t:BOOL) = if t then #5n else #6n;

fun setAspectConstraint(pre) =
  if pre=true then AspectPreConstraint
  else AspectPostConstraint;

fun getTreeAspect(tid:TREEID, []:TREELIST) = [] |
getTreeAspect(tid, tm::tl) =
  if (#1tid, #2tid, Aspect) = #1tm then #2tm else getTreeAspect(tid, tl);

fun getAspectType(tid:TREEID) =
  if #3tid=AspectPreConstraint then preAspect
  else postAspect

fun getExletCount([]) = 0 |
getExletCount(pr::xlt) = 1 + getExletCount(xlt);

fun getNumOfStr(s:STRING) =
  case s of
  "0" => 0 |
  "1" => 1 |
  "2" => 2 |
  "3" => 3 |
  "4" => 4 |
  "5" => 5
| "6" => 6 |
| "7" => 7 |
| "8" => 8 |
| "9" => 9;

fun getPow(i:INT) =
  if i=1 then 1
  else 10*getPow(i-1);

fun strToInt(s:STRING) =
  if String.size s = 1 then getNumOfStr(s)
  else
    (getNumOfStr( substring(s, 0, 1) )
    *
    getPow(String.size s)
    )
  +
    (strToInt( substring(s, 1, String.size s-1) ))
  );

fun getAction(p:PRIMITIVE) = #1p;
fun getTarget(p:PRIMITIVE) = #2p;
fun getWorklet(p:PRIMITIVE) = getTarget(p);
Appendix B. Evaluation Data CPN Declarations

(* value declarations *)

val testItem =

1 ` ("specID1", "caseID1", "taskID1", "itemID1", [("Attribute1","Value1")], [("Attribute11", "Value11")], true, false) ++

1 ` ("specID1", "caseID1", "taskID1", "itemID1", [("Attribute1","Value1")], [("Attribute11", "Value11")], false, false) ++

1 ` ("specID2", "target2", "taskID2", "itemID2", [("Attribute2","Value2")], [("Attribute22", "Value22")], true, true) ++

1 ` ("specID3", "target3", "taskID3", "itemID3", [("Attribute3","Value3")], [("Attribute33", "Value33")], true, true) ++

1 ` ("specID10", "caseID10", "taskID10", "itemID10", [("Attribute10","Value10")], [("Attribute110", "Value110")], true, false) ++

1 ` ("specID10", "caseID10", "taskID10", "itemID10", [("Attribute10","Value10")], [("Attribute110", "Value110")], false, false) ++

1 ` ("specID20", "target20", "taskID20", "itemID20", [("Attribute20","Value20")], [("Attribute220", "Value220")], true, true) ++

1 ` ("specID100", "caseID100", "taskID100", "itemID100", [("Attribute100","Value100")], [("Attribute1100", "Value1100")], true, false) ++

1 ` ("specID100", "caseID100", "taskID100", "itemID100", [("Attribute100","Value100")], [("Attribute1100", "Value1100")], false, false)

val imTList =

1 ` ("specID1", "taskID1", Aspect),

[(1, true),
 [advise, "target2"], [advise, "target3"],
 [("Attribute", "admin"), ~1, ~1]]
 ) ++

1 ` ("specID10", "taskID10", Aspect),

[(1, true),
 [advise, "target20"], [advise, "target30"],
 [("Attribute", "admin"), ~1, ~1]]
 ) ++

1 ` (}
("specID100", "taskID100", Aspect),
[(1, "true"),
[(advise, "target200"), (advise, "target300")],
[("Attribute", "admin")], ~1, ~1)]
)

val testCase =
1` ("specID1", "caseID1", [("Attribute1", "Value1"), ("Attribute2", "Value2")], false) ++
1` ("specID2", "target2", [("Attribute1", "Value1"), ("Attribute2", "Value2")], false) ++
1` ("specID3", "target3", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID10", "caseID10", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID10", "target1", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID20", "caseID20", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID30", "target30", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID100", "caseID100", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID200", "target200", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false) ++
1` ("specID300", "target300", ["Attribute1", "Value1"], ("Attribute2", "Value2")], false)

val imMTNode = 1` (1, "true", [(advise, "target")], [("a", "a")], 1, 1)
## Appendix C. Evaluation Scenarios

### Scenario 1

<table>
<thead>
<tr>
<th>Step</th>
<th>Net Name</th>
<th>Binding element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AspectService</td>
<td><code>(initialAspect, &lt;c=&quot;caseID1&quot;, pre=true&gt;)</code></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td><code>(getPointcutInfo, &lt;tList=&quot;[({&quot;specID1&quot;, &quot;taskID1&quot;, Aspect}), [{1, &quot;true&quot;, [{(advise, &quot;target2&quot;), (advise, &quot;target3&quot;)}], [{&quot;Attribute&quot;, &quot;admin&quot;}], ~1, ~1})]]&quot;&gt;)</code></td>
</tr>
<tr>
<td>3</td>
<td>selectPointcut</td>
<td><code>(storeArgs, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td><code>(evalNode)</code></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td><code>(resTrue)</code></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td><code>(termNode)</code></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td><code>(retExlet)</code></td>
</tr>
<tr>
<td>8</td>
<td>AspectService</td>
<td><code>(startAspect, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>9</td>
<td>launchAspect</td>
<td><code>(launchAspect, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td><code>(getnextAdvice, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td><code>(getnextAdvice, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>12</td>
<td>Core</td>
<td><code>(initAdvice, param=&quot;(&quot;itemID1&quot;, &quot;2&quot;)&quot;)</code></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td><code>(getAdviceNumber, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td><code>(getAdviceNumber, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td><code>(engine, cmd=&quot;suspendWorkItem&quot;, params=&quot;{&quot;item&quot;, &quot;itemID1&quot;}&quot;)</code></td>
</tr>
<tr>
<td>16</td>
<td>launchAspect</td>
<td><code>(enableLauncher, id=&quot;itemID1&quot;)</code></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td><code>(initAdvice, adv=&quot;target2&quot;)</code></td>
</tr>
<tr>
<td>18</td>
<td>Core</td>
<td><code>(engine, cmd=&quot;getInputParams&quot;, params=&quot;{&quot;spec&quot;, &quot;target2&quot;}&quot;)</code></td>
</tr>
<tr>
<td>19</td>
<td>launchAspect</td>
<td><code>(launchAdvice)</code></td>
</tr>
<tr>
<td>20</td>
<td>Core</td>
<td><code>(engine, cmd=&quot;launchCase&quot;, params=&quot;{&quot;spec&quot;, &quot;target2&quot;}, {&quot;params&quot;, &quot;{&quot;spec&quot;, &quot;target2&quot;}&quot;})&quot;</code></td>
</tr>
<tr>
<td>21</td>
<td>launchAspect</td>
<td><code>(saveAdviceState)</code></td>
</tr>
<tr>
<td>22</td>
<td>Core</td>
<td><code>(setAdviceInfo, param=&quot;(&quot;itemID1&quot;, &quot;target2&quot;)&quot;)</code></td>
</tr>
<tr>
<td>23</td>
<td>launchAspect</td>
<td><code>(initAdvice, adv=&quot;target3&quot;)</code></td>
</tr>
<tr>
<td>24</td>
<td>Core</td>
<td><code>(engine, cmd=&quot;getInputParams&quot;, params=&quot;{&quot;spec&quot;, &quot;target3&quot;}&quot;)</code></td>
</tr>
</tbody>
</table>
launchAspect (launchAdvice)

Core (engine, cmd="launchCase", params="[("spec", "target3"), ("params", ("spec", "target3"))]")

launchAspect (saveAdviceState)

Core (setAdviceInfo, param="(itemID1, target3")")

AspectService (Proceed, c="target2")

(Proceed, c="target3")

proceedAdvisedJoinPoint (prepare, c="target3")

(prepare, c="target2")

Core (getAdviceInfo, c="target2")

(getAdviceInfo, c="target3")

(setJoinpoint, param="(target2", "itemID2")")

(setJoinpoint, param="(target3", "itemID3")")

(setInitializedAdvice, c="target2")

(setInitializedAdvice, c="target3")

(engine, cmd="suspendWorkItem", params="["item", "itemID2"]")

(engine, cmd="suspendWorkItem", params="["item", "itemID3"]")

proceedAdvisedJoinPoint (updateData, c="target2")

(updateData, c="target3")

(setAdviceNum, id="itemID1")

Core (engine, cmd="updateWorkItemData", params="[("item", "itemID1"), ("data", (Attribute2, Value2))]")

(engine, cmd="updateWorkItemData", params="[("item", "itemID1"), ("data", (Attribute3, Value3))]")

proceedAdvisedJoinPoint (setAdviceInfo, c="target2", param2="(data", (Attribute2, Value2))")

(setAdviceInfo, c="target3", param2="(data", (Attribute3, Value3))")

(proceedAdvisedJoinPoint, id="itemID1")

Core (getAdviceNumber, id="itemID1")

(engine, cmd="unsuspendWorkItem", params="[("MainCase", "itemID1")]")

AspectService (initialAspect, <c="caseID1", pre=false>)
(getPointcutInfo, <tList="["specID1", "taskID1", Aspect], [1, "true", [(advise, "target2"), (advise, "target3")], ["Attribute", "admin"], ~1, ~1]]">)

(resumeAspect, <id="itemID1" >)

(resumeAdvice, <id="itemID1" >)

(Core (engine, <cmd="suspendWorkItem", params="["item", "itemID1"]" >))

(getAdvByItemID, <c="target2" >)

(getAdvByItemID, <c="target3" >)

(resumeAdvice (setSuspension, <id="itemID1">)

(getAdvices, <param="("itemID1", "target2")">)

(getAdvices, <param="("itemID1", "target3")">)

(updateAdviceData, <c="target2">)

(updateAdviceData, <c="target3">)

(Core (getJoinPoint, <c="target2">)

(getJoinPoint, <c="target3">)

(engine, <cmd="updateWorkItemData", params="["item", "itemID1"], ("data", "(Attribute1, Value1)")"]">)

(engine, <cmd="updateWorkItemData", params="["item", "itemID1"], ("data", "(Attribute1, Value1)")"]">)

(resumeAdvice (unsuspendAdvice, <id="itemID1">)

(unsuspendAdvice, <id="itemID1">)

(Core (engine, <cmd="unsuspendWorkItem", params="["item", "itemID1"], ("data", "(Attribute1, value1)")"]">)

(engine, <cmd="unsuspendWorkItem", params="["item", "itemID1"], ("data", "(Attribute1, value1)")"]">)

(resumeAdvice (forceCompleteProceedTask, <param2="("itemID1", "itemID2")">)

(forceCompleteProceedTask, <param2="("itemID1", "itemID3")">)

(Core (engine, <cmd="forceComplete", params="["item", "itemID2"]"]">)

(engine, <cmd="forceComplete", params="["item", "itemID3"]"]">)
AspectService (remCase, <c="target2">)

(remCase, <c="target3">)

finishAspect (finish advice, <c="target2">)

(finish advice, <c="target3">)

Core (getAdviceInfo, <c="target2">)

(getAdviceInfo, <c="target3">)

finishAspect (weaveAdvice, <c="target2">)

(weaveAdvice, <c="target3">)

(updateAdvisedJoinPointData, <c="target2">)

(updateAdvisedJoinPointData, <c="target3">)

Core (engine, <cmd="updateWorkItemData", params="[("item", "itemID1"), ("data", "(Attribute1, Value1) (Attribute2, Value2)")]")

engine, <cmd="updateWorkItemData", params="[("item", "itemID1"), ("data", "(Attribute1, Value1) (Attribute2, Value2)")]")

(removeCase, <c="target2">)

(removeCase, <c="target3">)

(finalizeAspect, <id="itemID1">)

(finalize, <id="itemID1">)

finishAspect (finalizeAspect, <param="("item", "itemID1")")

Core (engine, <cmd="unsuspendWorkItem", params="[("item", "itemID1")]")

finishAspect (finish, <id="itemID1">)

AspectService (remCase, <c="caseID1">)

finishAspect (finish advice, <c="caseID1">)

(RemoveAdvisedCaseInfo, <c="caseID1">)
<table>
<thead>
<tr>
<th>Step</th>
<th>Net</th>
<th>Name</th>
<th>Binding element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AspectService</td>
<td>(initialAspect, (&lt;c=\text{caseID10}, \text{pre}=\text{true}&gt;))</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(getPointcutInfo, (&lt;tList={((\text{specID10}, \text{taskID10}, \text{Aspect}), {(1, \text{true}, {{\text{advise, target20}}, {\text{advise, target30}}})), ((\text{Attribute}, \text{admin})], ~1, ~1)})&gt;)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>selectPointcut</td>
<td>(storeArgs, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>(evalNode)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>(resTrue)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>(termNode)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>(retExlet)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>AspectService</td>
<td>(startAspect, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>launchAspect</td>
<td>(launchAspect, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>(getnextAdvice, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>(getnextAdvice, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Core</td>
<td>(initAdvice, param={\text{itemID10}, \text{2}}))</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>(getAdviceNumber, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>(getAdviceNumber, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>(engine, cmd=\text{suspendWorkItem}, params={{\text{item}, \text{itemID10}}})</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>launchAspect</td>
<td>(enableLauncher, id=\text{itemID10}))</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>(initAdvice, adv=\text{target20}))</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Core</td>
<td>(engine, cmd=\text{getInputParams}, params={{\text{spec}, \text{target20}}})</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>launchAspect</td>
<td>(launchAdvice))</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Core</td>
<td>(engine, cmd=\text{launchCase}, params={{\text{spec}, \text{target20}}, {\text{params}, {{\text{spec}, \text{target20}}}}} }</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>launchAspect</td>
<td>(saveAdviceState))</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Core</td>
<td>(setAdviceInfo, param={\text{itemID10}, \text{target20}}))</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>launchAspect</td>
<td>(initAdvice, adv=\text{target30}))</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Core</td>
<td>(engine, cmd=\text{getInputParams}, params={{\text{spec}, \text{target30}}})</td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>launchAspect (launchAdvice)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Core (engine, cmd=&quot;launchCase&quot;, params=[{&quot;spec&quot;, &quot;target30&quot;}, {&quot;params&quot;, &quot;&quot;spec&quot;, &quot;target30&quot;}])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>launchAspect (saveAdviceState)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Core (setAdviceInfo, param=&quot;&quot;itemID10&quot;, &quot;target30&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>AspectService (proceed, &lt;c=&quot;target20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>proceedAdvisedJoinPoint (prepare, &lt;c=&quot;target20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Core (getAdviceInfo, &lt;c=&quot;target20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>(setJoinpoint, &lt;param=&quot;&quot;target20&quot;, &quot;itemID20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>(setInitializedAdvice, &lt;c=&quot;target20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>(engine, &lt;params=[{&quot;item&quot;, &quot;itemID20&quot;}])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>proceedAdvisedJoinPoint (updateData, &lt;c=&quot;target20&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Core (engine, &lt;params=[{&quot;item&quot;, &quot;itemID10&quot;}, {&quot;data&quot;, &quot;(\text{Attribute20}, \text{Value20})}}] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>AspectService (remCase, &lt;c=&quot;target30&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>finishAspect (finish advice, &lt;c=&quot;target30&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Core (getAdviceInfo, &lt;c=&quot;target30&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>finishAspect (weaveAdvice, &lt;c=&quot;target30&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>(updateAdvisedJoinPointData, &lt;c=&quot;target30&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Core (engine, &lt;cmd=updateWorkItemData&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>(removeCase, &lt;params2=[{&quot;data&quot;, &quot;(\text{Attribute20}, \text{value20})}}] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>(updateAdviceNumber, &lt;id=&quot;itemID10&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>proceedAdvisedJoinPoint (setAdviceNum, &lt;id=&quot;itemID10&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>(setAdviceInfo, &lt;id=&quot;itemID10&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>proceedAdvisedJoinPoint, &lt;id=&quot;itemID10&quot;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Core (getAdviceNumber, &lt;id=&quot;itemID10&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>(engine, &lt;cmd=unsuspendWorkItem&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>AspectService (initialAspect, &lt;c=&quot;caseID10&quot;, pre=false)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>(getPointcutInfo, &lt;tList=[{&quot;specID10&quot;, &quot;taskId10&quot;, Aspect}, {&quot;1&quot;, &quot;true&quot;, [{advise, &quot;target20&quot;}], {advise, &quot;target30&quot;}}] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>resumeAspect, &lt;id=&quot;itemID10&quot; &gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>resumeAdvice (finish aspect, &lt;id=&quot;itemID10&quot; &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Core (engine, &lt;cmd=&quot;suspendWorkItem&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>params=&quot;[&quot;item&quot;, &quot;itemID10&quot; ]&quot; &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>(getAdvByItemID, &lt;c=&quot;target2 &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>resumeAdvice (setSuspension, &lt;id=&quot;itemID10&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>(getAdvices, &lt;param=&quot;(&quot;itemID10&quot;, &quot;target20&quot;)&quot;&gt;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 58   |(updateAdviceData, <c="target20">
| 59   |Core (getJoinPoint, <c="target20" >)                               |
| 60   |Core (engine, <cmd="updateWorkItemData",  
|      |params="["item", "itemID10" ],  
|      |"data",  
|      |"{Attribute10, Value10}" ]") >)                                 |
| 61   |resumeAdvice (unsuspendAdvice, <id="itemID10">)                    |
| 62   |Core (forceCompleteProceedTask,  
|      |<param2="("itemID10", "itemID20")">
| 63   |Core (engine, <cmd="forceComplete",  
|      |params="["item", "itemID20" ]") >)                               |
| 64   |AspectService (remCase, <c="target20" >)                            |
| 65   |finishAspect (finish advice, <c="target2">)                        |
| 66   |Core (getAdviceInfo, <c="target20" >)                              |
| 67   |finishAspect (weaveAdvice, <c="target20" >)                        |
| 68   |(updateAdvisedJoinPointData, <c="target20">)                       |
| 69   |Core (engine, <cmd="updateWorkItemData", 
|      |params="["item", "itemID10" ], 
|      |"data", 
|      |"{Attribute10, Value10} (Attribute20, Value20)" ]") >)             |
| 70   |(removeCase, <c="target20">
<p>| 71   |(finilizeAspect, &lt;id=&quot;itemID10&quot; )                                  |
| 72   |(finilize, &lt;id=&quot;itemID10&quot; )                                        |
| 73   |(finishAspect, &lt;id=&quot;itemID10&quot;&gt;)                                    |
| 74   |(finalization, &lt;c=&quot;target10&quot;&gt;)                                     |</p>
<table>
<thead>
<tr>
<th>Line</th>
<th>Source</th>
<th>Command/Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>Core</td>
<td><code>engine, &lt;cmd=&quot;unsuspendWorkItem&quot;, params=&quot;[(&quot;item&quot;, &quot;itemID10&quot;)]]&quot;&gt;</code></td>
</tr>
<tr>
<td>76</td>
<td>finishAspect</td>
<td>(finish, <code>&lt;id=&quot;itemID10&quot;&gt;</code>)</td>
</tr>
<tr>
<td>77</td>
<td>AspectService</td>
<td>(remCase, <code>&lt;c=&quot;caseID10&quot;</code>)</td>
</tr>
<tr>
<td>78</td>
<td>finishAspect</td>
<td>(finish advice, <code>&lt;c=&quot;caseID10&quot;</code>)</td>
</tr>
<tr>
<td>79</td>
<td></td>
<td>(RemoveAdvisedCaseInfo, <code>&lt;c=&quot;caseID10&quot;</code>)</td>
</tr>
</tbody>
</table>
### Scenario 3

<table>
<thead>
<tr>
<th>Step</th>
<th>Net Name</th>
<th>Binding element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AspectService</td>
<td>(initialAspect, (&lt;c=&quot;caseID100&quot;, \pre=true&gt;))</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(getPointcutInfo, (&lt;tlList=&quot;[(&quot;specID100&quot;, &quot;taskID100&quot;, \Aspect), [(1, &quot;true&quot;, [{(\text{advise, &quot;target200&quot;}, \{(\text{advise, &quot;target300&quot;}})}], {&quot;Attribute&quot;, &quot;admin}}, \sim 1, \sim 1])])))</td>
</tr>
<tr>
<td>3</td>
<td>selectPointcut</td>
<td>(storeArgs, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>(evalNode)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>(resTrue)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>(termNode)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>(retExlet)</td>
</tr>
<tr>
<td>8</td>
<td>AspectService</td>
<td>(startAspect, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>9</td>
<td>launchAspect</td>
<td>(launchAspect, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>(getnextAdvice, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>(getnextAdvice, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>12</td>
<td>Core</td>
<td>(initAdvice, param=&quot;(&quot;itemID100&quot;, &quot;2&quot;)&quot;)</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>(getAdviceNumber, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>(getAdviceNumber, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>(engine, (cmd=&quot;suspendWorkItem&quot;, \params=&quot;{&quot;item&quot;, &quot;itemID100&quot;}})</td>
</tr>
<tr>
<td>16</td>
<td>launchAspect</td>
<td>(enableLauncher, id=&quot;itemID100&quot;)</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>(initAdvice, adv=&quot;target200&quot;)</td>
</tr>
<tr>
<td>18</td>
<td>Core</td>
<td>(engine, (cmd=&quot;getInputParams&quot;, \params=&quot;{&quot;spec&quot;, &quot;target200&quot;}})</td>
</tr>
<tr>
<td>19</td>
<td>launchAspect</td>
<td>(launchAdvice)</td>
</tr>
<tr>
<td>20</td>
<td>Core</td>
<td>(engine, (cmd=&quot;launchCase&quot;, \params=&quot;{&quot;spec&quot;, &quot;target200&quot;}, {&quot;params&quot;, {&quot;spec&quot;, &quot;target200&quot;}})})</td>
</tr>
<tr>
<td>21</td>
<td>launchAspect</td>
<td>(saveAdviceState)</td>
</tr>
<tr>
<td>22</td>
<td>Core</td>
<td>(setAdviceInfo, param=&quot;(&quot;itemID100&quot;, &quot;target200&quot;)&quot;)</td>
</tr>
<tr>
<td>23</td>
<td>launchAspect</td>
<td>(initAdvice, adv=&quot;target300&quot;)</td>
</tr>
<tr>
<td>24</td>
<td>Core</td>
<td>(engine, (cmd=&quot;getInputParams&quot;, \params=&quot;{&quot;spec&quot;, &quot;target300&quot;}})</td>
</tr>
<tr>
<td>25</td>
<td>launchAspect</td>
<td>(launchAdvice)</td>
</tr>
<tr>
<td>26</td>
<td>Core</td>
<td>(engine, (cmd=&quot;launchCase&quot;, \params=&quot;{&quot;spec&quot;, &quot;target300&quot;}, {&quot;params&quot;, {&quot;spec&quot;, &quot;target300&quot;}})})</td>
</tr>
<tr>
<td>Line</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td><code>launchAspect (saveAdviceState)</code></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td><code>Core (setAdviceInfo, param=&quot;(&quot;itemID100&quot;, &quot;target300&quot;)&quot;)</code></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td><code>AspectService (remCase, &lt;c=&quot;target200&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td><code>AspectService (remCase, &lt;c=&quot;target300&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td><code>finishAspect (finish advice, &lt;c=&quot;target200&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td><code>Core (finish advice, &lt;c=&quot;target300&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td><code>Core (getAdviceInfo, &lt;c=&quot;target200&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td><code>Core (getAdviceInfo, &lt;c=&quot;target300&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td><code>finishAspect (weaveAdvice, &lt;c=&quot;target200&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td><code>finishAspect (weaveAdvice, &lt;c=&quot;target300&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td><code>Core (updateAdvisedJoinPointData, &lt;c=&quot;target200&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td><code>Core (updateAdvisedJoinPointData, &lt;c=&quot;target300&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td><code>Core (engine, &lt;cmd=updateWorkItemData&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td><code>Core (engine, &lt;cmd=updateWorkItemData&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td><code>Core (removeCase, &lt;c=&quot;200&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td><code>Core (removeCase, &lt;c=&quot;300&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td><code>Core (updateAdviceNumber, &lt;id=&quot;itemID10&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td><code>Core (finalizeAspect, &lt;id=&quot;itemID100&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td><code>finishAspect (finalizeAspect, &lt;param=&quot;(&quot;item&quot;, &quot;itemID100&quot;)&quot;)</code>)</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td><code>Core (engine, &lt;cmd=unsuspendWorkItem, params=&quot;[(&quot;item&quot;, &quot;itemID100&quot;)]&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td><code>finishAspect (finish, &lt;id=&quot;itemID100&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td><code>AspectService (initialAspect, &lt;c=&quot;caseID100&quot;, pre=false&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td><code>AspectService (getPointcutInfo, &lt;tList=&quot;[([&quot;specID100&quot;, &quot;taskID100&quot;, Aspect), [(1, &quot;true&quot;, [(advise, &quot;target200&quot;), (advise, &quot;target300&quot;)], [(&quot;Attribute&quot;, &quot;admin&quot;)), -1, -1)])&quot;]&quot;)</code></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td><code>AspectService (resumeAspect, &lt;id=&quot;itemID100&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td><code>resumeAdvice (finish aspect, &lt;id=&quot;itemID100&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td><code>Core (finalize, &lt;id=&quot;itemID100&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td><code>AspectService (remCase, &lt;c=&quot;caseID100&quot; &gt;)</code></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td><code>finishAspect (finish advice, &lt;c=&quot;caseID100&quot;&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td><code>AspectService (removeAdvisedCaseInfo, &lt;c=&quot;caseID100&quot;&gt;)</code></td>
<td></td>
</tr>
</tbody>
</table>