2007

Service-oriented design in aspect-oriented and Petri net-based approach

Tae-hyung Kim
Iowa State University

Service-oriented design in aspect-oriented and Petri net-based approach

by

Tae-hyung Kim

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Carl K. Chang, Major Professor
Johnny S. Wong
Ying Cai
Dan Zhu
Morris Chang

Iowa State University

Ames, Iowa

2007

TABLE OF CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

ACKNOWLEDGEMENTS ix

ABSTRACT x

CHAPTER 1. INTRODUCTION 1

1.1 Overview 1

1.2 Objectives 2

1.3 Outline of Approach 3

1.4 Contributions 4

1.5 Dissertation Organization 5

CHAPTER 2. BACKGROUND OF RESEARCH 6

2.1 Service-oriented Computing 6

2.1.1 Services 6

2.1.2 Service-oriented Architecture 7

2.2 Aspect-oriented Approach 8

2.3 Petri Net 9

2.3.1 Basic Petri Net 9

2.3.2 Time Extended Petri Nets 11

2.3.3 High-Level Petri Nets 12

2.3.4 Petri Net Markup Language (PNML) 12

CHAPTER 3. SOFTWARE SYSTEM DECOMPSITON WITH ASPECTS 14

3.1 Introduction....14
3.2 Function-Class Decomposition with Aspect....16
3.2.1 Aspects....16
3.2.2 Process of FCD-A....17
3.3 Example....18
3.4 Related Work....25
3.5 Summary and Discussion....28

CHAPTER 4. SERVICE-ORINETED DESIGN WITH ASPECTS....30
4.1 Introduction....30
4.2 Services and Aspects....31
4.2.1 Services....32
4.2.2 Aspects....35
4.3 Graphical Representation....38
4.3.1 Service Entity....40
4.3.2 Service Chain....40
4.3.3 Service Net....42
4.3.4 Aspect....43
4.3.5 Crosscut Relationship....44
4.4 XML-Based Representation....47
4.4.1 Service Markup Language (SvML)....48
4.4.2 Aspect Markup Language (AsML)....49
4.5 Weaving Process....50
4.5.1 Weaving....51
4.5.2 Overlapped Crosscut Points....54
4.5.3 Restricted Instantiation of Aspects....55

4.6 Example 57
4.6.1 Weaving 59
4.6.2 Analysis 61
4.7 Related Work 62
4.8 Summary and Discussion 66

CHAPTER 5. EXTENSION OF SERVICE-ORIENTED DESINGS 69
5.1 Introduction 69
5.2 Overview of Aspect-oriented Extension Mechanism 70
5.3 Extension Mechanism for Design-specific Data 74
5.3.1 Petri Net Extension Markup Language (PeML) 75
5.3.2 Extension Process 76
5.3.3 Example 77
5.4 Extension Mechanism for Resource-related Data 79
5.4.1 Resource Extension Markup Language (ReML) 79
5.4.2 Resource Definition 81
5.4.3 Resource Interference 82
5.4.4 Extension Process 83
5.4.5 Example 87
5.5 Related Work 93
5.6 Summary and Discussion 94

CHAPTER 6. CONCLUSIONS AND FUTURE WORK 97
6.1 Conclusions 97
6.2 Future Work 99

APPENDIX. ALGORITHMS 102

BIBLIOGRAPHY 107

LIST OF TABLES

Table 1: The crosscutting methods 45

Table 2: The weaving results of the sequential server model..... 89

LIST OF FIGURES

Figure 1: A basic Petri net 10
Figure 2: The meta-model of PNML 13
Figure 3: The process model of FCD-A 15
Figure 4: Process of FCD-A 18
Figure 5: The initial classes with the system-level functional module and the key aspects to be considered 19
Figure 6: Allocating aspect links to the initially identified classes 21
Figure 7: Analysis of aspect links after initial grouping decisions 23
Figure 8: Function-class and aspect views of the retailer system prior to identifying additional classes for the next level decomposition 24
Figure 9: A sequential service with three atomic services 33
Figure 10: A service-oriented system with a service entity and two service chains plus aspects 38
Figure 11: The graphical representation of the crosscut relationship in the tagged value “pointcut” 46
Figure 12: The meta-model and structure of the SvML 48
Figure 13: The meta-model and structure of the AsML 49
Figure 14: The weaving process for generating an intergrate Petri net for a service 51
Figure 15: The Petri net semantics for five crosscutting methods 52
Figure 16: An overlapped crosscut point with three aspects 54
Figure 17: The aspect with the tagged value “sync” 56
Figure 18: The order service in the retailer service net with four aspects 57
Figure 19: The integrated Petri nets for the different versions of the order service 60
Figure 20: The meta-model of PeML including ReML 72

Figure 21: The weaving process for extending a Petri net for a service 73
Figure 22: A PeML File 75
Figure 23: The simulation results of integrated order services
extended with the PeML file 78
Figure 24: A ReML file 80
Figure 25: Weaving instantiated resource aspects 84
Figure 26: The augmented Petri net with two instantiated resource aspects 85
Figure 27: Multiple crosscutting of a service by a resource
with the tagged value “limit” 86
Figure 28: Two order services with the customer modeling 88
Figure 29: The replaced weaving rules of the ReML file in Figure 24 89
Figure 30: The augmented Petri nets for two order services
after weaving the Resource_2.reml file 90
Figure 31: The simulation results of the sequential order service
extended with two ReML files 91
Figure 32: The simulation results of the sequential and the parallel order service
with varying resource availability 92

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me to complete this work.

First and foremost, I would like to thank my advisor, Prof. Carl K. Chang, for his guidance, patience and support throughout this research and the writing of this thesis. His broad and critical insights had helped me more than he knows.

I also would like to thank Dr. Johnny S. Wong, Dr. Ying Cai, Dr. Dan Zhu, and Dr. Morris Chang for their contributions as my committee members.

I especially would like to thank my friends and lab colleagues, in particular Minoh Kang, Jinchun Xia, and Hsin-yi Jiang, for every kind of support they brought to me during my work.

I am profoundly thankful for the endless love, trust and support so unselfishly given to me from my mother, Neung-ja Oh, and my father, Woo-koo Kim.

Finally, I cannot express how much I love my wife, Jiyon Im, and my daughter, Michelle Caen Kim. I am always happy with them.

ABSTRACT

Service-oriented computing (SOC) is an emerging paradigm utilizing services as core elements in software development. However, the service design of SOC oftentimes fails to capture various service-specific concerns required for delivering high-quality and user-friendly services. This is because those concerns are intrinsically tangled within a service. If such concerns, often crosscutting the system, are not satisfactorily treated, a service design result will be inadequate to reflect all facets of a service. The objective of this research, thus, is to provide a systematic, comprehensive, but generic and formal service-oriented design approach that helps to effectively and efficiently develop services in service-oriented systems in the design process.

Our approach is to provide a service-oriented design approach integrated with the concept of aspects and supported by Petri net formalism. Recently, the aspect-oriented programming (AOP) has gained growing interest and its "aspect" concept has received attention as the newly found solution to the early stages of software development as an abstraction and encapsulation mechanism with the purpose of enhancing separation of concerns. We integrate the concept of aspect into the service-oriented design process that consists of the decomposition of a service-oriented system, the structural and behavioral representations of services, and the extension of the semantic annotations on services for the analysis purpose.

In our approach, a service-oriented system is decomposed as a set of primitive services that contain only essential functional features and aspects methodically defined and used to capture pertinent service-specific or domain-specific concerns. Using primitive services and aspects, our method delineates a service-oriented system in two views, the structural view and the behavioral view, based on an extended UML2 and Petri net representations, respectively. In particular, our method supports an automatic weaving

process to generate an integrated Petri net for each distinct service from the behavioral perspective of both a primitive service and a set of aspects related to it, and the relationships between them. As a result, the integrated Petri nets obtained through the weaving process facilitate the verification and evaluation of service design results. To exploit these integrated Petri nets that correspond to composed services, our method supports an aspect-oriented extension mechanism to help comparative evaluation of the service design results, for example, in terms of performance or platform-specific resource interferences.

Finally, such a formal representation and extension method with a standardized description in XML makes it possible to evolve and analyze service-oriented designs in existence or construct varying versions of a service design with reduced development effort by replacing or reusing existing design elements, especially aspects. Simulation results provide convincing data as proof although further experiments with real-life system development are still desirable, as our future work.

CHAPTER 1. INTRODUCTION

Service-oriented computing (SOC) as an emerging paradigm in the software engineering community utilizes services as core elements in software development. Due to the multi-facets of a service, the service design of SOC oftentimes has difficulty to cover all important service-specific concerns required for delivering high-quality and user-friendly services. This is because those concerns are intrinsically tangled within a service [82]. If such concerns, often crosscutting the system, are not satisfactorily treated, a service design result would be inadequate to reflect all facets of a service. The primary objective of this dissertation is to provide a systematic, comprehensive, but generic and formal service-oriented design approach that helps developers to effectively and efficiently develop services in the design process. In this chapter, we present the important goals and an outline of the approach presented in this dissertation.

1.1 Overview

Generally speaking, a service-oriented system consists of a set of services rendered to different classes of customers. As the services belonging to a service-oriented system need to be developed independently, some of them quite often share identical functional and non-functional features. From a bird's eye view of whole service-oriented systems, some services may be under the control of system-wide functional and non-functional features. Certain services need to have varying versions in order to satisfy customer's preferences. Furthermore, services need to be updated or evolved in order to survive in the extremely competitive e-service marketplace by following the rapid shifting e-commerce trends or demands. This kind of issues related to reusability, customizability, manageability, and maintainability of the service development needs to be considered at the early stage of service-system development. To deal with those critical issues in an efficient and effective manner, we develop a service-oriented design method based on an aspect-oriented approach

and Petri net formalism. Recently, the aspect-oriented approach has gained growing interest and its "aspect" concept has been extensively applied to the early stages of software development, such as aspect-oriented software development (AOSD), as an abstraction and encapsulation mechanism for crosscutting concerns with the purpose of enhancing separation of concerns. In our approach, this aspect-oriented mechanism is comprehensively applied to decompose a service-oriented system, and to describe, compose and extend its services in the design process. In particular, the representation, fabrication and extension of a service from its behavioral perspective are supported by a formal method based on the Petri net-based semantics.

1.2 Objectives

The development of services in a service-oriented system is required to cover significant concerns without causing unexpected side-effects such as redundancy. Some services may call for variations in order to support a wide spectrum of customer's preferences or contexts. After services are developed, they need to be continuously updated or evolved to catch up with changing e-service market trends or demands as well as the rapidly shifting e-service technologies. To meets these requirements, a novel service-oriented design method is necessary to support an affective way to identify services in a service-oriented system, represent a service with all critical concerns, generate its variations, facilitate its evolution and support service analysis process. The major goal of the design approach in this dissertation is to make the development of services in the design process effective and efficient by providing a generic and formal service-oriented design method. Accordingly, the important purposes of the design method are addressed as follows:

- To support a decomposition process of service-oriented systems.
- To provide a standardized way to represent services in a decomposed service-oriented system from both the structural perspective and the behavioral perspective.

- To provide an automatic technique to generate a service or its variation from the decomposed elements in terms of formal operational semantics.
- To provide an extension mechanism for assisting the design analysis process of the generated services.

1.3 Outline of Approach

To achieve the objectives in the previous section, an aspect-oriented mechanism is integrated to the entire procedure of the design method from the decomposition of a service-oriented system to the extension of the composed services to be analyzed. The approaches in the dissertation are summarized as follows:

- **A generic software system decomposition method with aspect-oriented approach:**

  A generic decomposition method integrated with the concept of aspect can be applied to decompose any software system, including service-oriented systems, and organize the design elements that can be functional or non-functional into two distinct views in order to provide a high-level blueprint of a software system at the initial design stage.

- **An aspect-oriented representation method for the structure and behavior of services in a service-oriented system using an extended UML2 and a set of XML-based description languages:**

  The graphical and XML-based representation methods provide a way of treating every design element in service-oriented design separately in either structural or behavioral viewpoint. Such methods help enhance the reusability, manageability and traceability of the service-oriented design results and the corresponding service-oriented system. The XML-based representation method not only increases interchangeability and interoperability of service-oriented design results, but also

facilitates the implementation of the weaving program independent of any specific development tool.

- **A Petri net-based semantics to support the formal definitions of service behavior and service composition:**

  The Petri net-based semantics supports the description of any behavior of a design element and composition of a service using the design elements in terms of their behavior and relationships between them.

- **An automatic weaving process to generate services composed of design elements or their variations:**

  Various versions of a service design can be generated through an automatic weaving process by weaving auxiliary behaviors of aspects into the behavior of a primitive service.

- **A systematic extension mechanism to validate and analyze the service design results:**

  The service design results can be extended with the information related to its behavior such as temporal values or resource interferences at the later design stage, which is useful in estimating or predicting the behavioral characteristics of an generated service design, such as its performance, or to comparatively evaluate candidate service designs.

1.4 Contributions

The contributions of this research can be summarized as follows:

- **An extended hybrid decomposition method for a software system.**

  This outcome supports a more flexible decomposition method to help understand a developing system by means of identifying, separating and categorizing non-primitive features as well as classifying and grouping primitive features.

- **A generic service-oriented design method strongly integrated with an aspect-oriented approach to represent, generate and extend design results.**

  This outcome contributes to software engineering research by providing an effective and efficient design method of service-oriented systems and successfully integrating an aspect-oriented process into this design method.

- **Useful design results in a standardized representation and in a harmonious fashion of the structural and the behavioral perspectives**.

  The structure of a service-oriented system and the behavior of its services can be represented in graphical and documental form. In particular, the documental form in XML enhances interchangeability and interoperability of the design results.

- **A formal and mechanical process to the generation of service design results and their extension with behavior-specific information by providing an aspect-oriented weaving mechanism.**

  A concrete weaving mechanism in the design stage constitutes the step stone of a completely automated design process.

1.5 Dissertation Organization

The rest of this dissertation is organized as follows.

In Chapter 2, we introduce important background concepts used in this dissertation.

In Chapter 3, we discuss a way to decompose software systems extended from the function-class decomposition (FCD) method by means of integrating the concept of aspect.

In Chapter 4, we present the aspect-oriented and Petri net-based design method for service-oriented systems named service-oriented design with aspect, abbreviated as SODA.

In Chapter 5, we explain an extension method with aspect-oriented mechanism for the design results obtained through SODA in order to assist the design analysis process.

In Chapter 6, we conclude this dissertation with future work.

CHAPTER 2. BACKGROUND OF RESEARCH

The service-oriented design method discussed in this dissertation is principally dependent on two fundamental concepts: aspect-oriented approach and Petri net. This chapter briefly introduces these two concepts after the meaning of services is explained first.

2.1 Service-oriented Computing

Service oriented computing (SOC) as an emerging cross-disciplinary paradigm for distributed computing utilizes services as core elements in software development and requires a new approach for architecting, designing, publishing and consuming software elements. In this section, the meaning of services in SOC is discussed as a starting point. Then, service-oriented architecture (SOA) is briefly introduced in terms of web services that are for the present the most prominent technology based on the concept of SOC.

2.1.1 Services

In SOC, services are defined as self-describing and platform-agnostic functional elements that support rapid construction of distributed systems at low cost [26]. A service with a fundamental function with an interface that can be published and discovered entails a variety of functional and non-functional features perceived and received by customers, in order to complete a prescribed mission. To encapsulate all the necessary features, the concept of service has been used as a basic abstraction unit in the service model developments [37][38][39].

The concept of service becomes clear when compared with the one of component that is regarded as a functional black-box object that delivers specific predefined features. Compared with components in the strict sense, services provide coarse-grained granularity of features varied from system to system. In a narrow sense, the component-based software development (CBSD) could be regarded as a service-oriented software development with a

restricted concept of the service. Some researches assume that service is another term for the component [27]. Many researches for CBSD, however, are aware that the component concept as an encapsulation unit is not sufficient to separate a broad range of issues related to a service such as adaptability, security, availability, and performance. To overcome this limitation, several CBSD methods incorporate with aspect-oriented approach for developing service-oriented systems [16][46].

The more specific definition of services tends to be quite diverse according to platforms, languages, or perspectives where the term "service" is applied. Moreover, the concept of functionality is even more abstract and may not be realized via using only a set of implementable units like classes or components. As a useful reference, the definition of service in the development of Web services and its tool support is well discussed in [82].

2.1.2 Service-oriented Architecture

In SOC, service-oriented architecture (SOA) builds the service model that organizes a set of software components into services. The infrastructure of SOA presumes loosely coupled heterogeneous platforms with message-oriented peer-to-peer communication via the wired and wireless Internet.

In particular, the SOA for web services based on the *Publish-find-interact* (or *find-bind-execute*) paradigm needs to support a standard means of interoperating between peer web services or communicating among three different roles: service provider, service register and service requester. The interoperability in the web service architecture is achieved using XML-based lingua francas, such as SOAP [77], UDDI [78]and WSDL [79], which lets a web service cooperate with other web services across heterogeneous platforms or communicate with other applications in a standardized way. In addition to those three basic web services standards, the second-generation web service specifications have been developed to support evolution of the service-oriented enterprise (SOE). Among them, BPEL4WS (WS-BPEL

from 2.0) [34] is part of the second-generation specifications, but tends to be considered as an essential one to express the behavior of a business process for a SOE. Although web services technology currently provides the extended infrastructure for implementing SOA, they are not sufficient to deliver a qualified service without supporting other standards such as WS-Policy [80].

2.2 Aspect-oriented Approach

In terms of *separation of concerns* [1], object-oriented (OO) methods show limitations to satisfy requirements related to a spectrum of important concerns that are important, or critical, properties pertaining to a specific domain. Upon decomposing a system, most OO methods support only a single dimension of concerns based on objects or data. This problem is called the "tyranny of dominant decomposition" [2]. Moreover, this single-threaded decomposition scatters concerns over multiple functional modules or classes and hinders developers from injecting different concerns upon analyzing the system. As a result, the codes relating these concerns are spread across or get tangled in several classes or modules in the implementation phase. This phenomenon is called the "code-tangling problem" [3]. In legacy implementations, these scattered concerns exist in tangled codes, which decrease modularity, and make it hard to develop and maintain software systems.

To avoid these problems, many researchers have made contributions to achieve separation of concerns [2][3][5]. In particular, Aspect-Oriented Programming (AOP) [3] provided the mechanism to encapsulate crosscutting concerns into the abstraction known as *aspect* and combine (weave) them with functional elements in the implementation phase. This isolation improves modularity and traceability of a system. Aspects derived from concerns conceptually involve functional and non-functional elements of the entire system.

Therefore, it is believed that non-functional requirements such as security, availability and testability can be identified and cast into varying aspects. Since AOP concentrates on the

implementation phase of software development cycle, a number of aspect-oriented development approaches have been studied in order to provide a better understanding of a system in its earlier stage of software development [14][15][16][17][18][19][44][63][64].

Aspect-oriented approach has been applied to extend web services as well. An extended aspect-oriented component engineering (AOCE) method to support the development of web services technology [44] proposed AO-WSDL and AO-UDDI that include the annotated details derived from the proposed component aspects separated from several modules, such as user interface, distribution, transaction processing, security, persistency, configuration, etc. Although AO-WSDL and AO-UDDI can be performed with support of web service-oriented systems, their extended AOCE approach illustrated that the aspect-oriented approach can also help the development of web service-oriented systems. AO4BPEL [45] was proposed as an aspect-oriented approach to tackle the lack of modularity and the drawback of static composition of BPEL4WS. It uses aspect as a stand-alone business logic or service in the XML format that can be plugged into or unplugged from the composition process at runtime.

2.3 Petri Net

Since Petri net developed by C. A. Petri is supported by a sound mathematical foundation and a graphical representation, it has been widely used to model software [67] as well as hardware [65]. When Petri net is used to model software architecture or workflow systems, it has the advantage over other diagrammatic representations such as statechart or sequence diagram in UML due to its mathematical soundness and availability of various Petri net tools. In this section, we discuss the fundamentals of Petri net.

2.3.1 Basic Petri Net

Basic Petri net (Place/Transition net or P/T net) is defined using the 5-tuple as follows.

Definition 1: A Petri net $P = (P, T, I, O, M_0)$ where:

- $P = \{P_1, P_2, \ldots, P_m\}$ is a finite set of places.
- $T = \{T_1, T_2, \ldots, T_n\}$ is a finite set of transitions.
- $I \subseteq P \times T$ is a finite set of input arcs directed from places to transitions.
- $O \subseteq T \times P$ is a finite set of output arcs directed from transitions to places.
- $M_0: P \rightarrow \{0, 1, 2 \ldots\}$ is the initial marking.
- $T \neq \varnothing$ and $P \cap T = \varnothing$.

In terms of a transition, the places that can be reversely traced to via the input arcs are called the input places of the transition while the places connected by the output arcs are called the output places of the transition.

The above Petri net definition can have another tuple for a weight function $W_0 : T \rightarrow \{1, 2, 3, \ldots\}$. When a Petri net has a weight function, a transition t is enabled if the number of tokens in each input place p_I is equal to or larger than the weight of the input arc. After the transition t fires, each output place p_O has the same number of tokens as the weight of the output arc. A Petri net is *ordinary* when all of its arc weights are 1's.

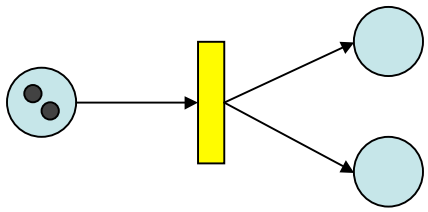


Figure 1: A basic Petri net

In graphical representation of basic Petri nets, places and transitions are drawn as circles and bars (or squares) respectively. Input arcs connect places to transitions whereas output arcs connect transitions to places. If a weight function is defined, its weight values are

attached to arcs. Places may have tokens. The state of a Petri nets called the Petri net marking is defined by the distribution of the number of tokens in each place. In particular, the initial state of a Petri net is specified by an initial marking. Figure 1 shows the graphical representation of a simple Petri net with one transition, three places, one input arcs, two output arcs and two initial tokens. The more detailed introduction of Petri nets can be found in [75] [29].

2.3.2 Time Extended Petri Nets

There are many extensions to Petri nets that consider time due to the necessity to describe temporal behavior of a system to be modeled. A Petri net can be associated with time whose values are either intervals (durations) or delays. The Petri net associated with intervals are called time Petri net while the one with delays called timed Petri net. In the case of timed Petri nets, the delay can be associated with transitions, places or arcs, which produces timed transition Petri nets (TTPN), timed place Petri nets (TPPN) and timed arcs Petri nets (TLPN), respectively. Among them, TTPN is mostly used to model a software/hardware system by interpreting transitions as its operations or tasks. For this purpose, the definition of the TTPN includes a set of delay values associated with its transitions. TTPN can be classified according to the type of the delays introduced. The delays of stochastic Petri nets (SPN) are stochastic and exponentially distributed. The delays of generalized stochastic timed Petri nets (GSPN) can have either zero (immediate) in addition to being exponentially distributed. The delays of deterministic and stochastic Petri nets (DSPN) are either constant (deterministic) or exponentially distributed. The delays of extended stochastic Petri nets (ESPN) are generally distributed. Finally, there is a Petri net of which the delays are zero, constant (deterministic), or generally distributed. Usually, TTPN uses three-phases firing for its transitions that hold the tokens during its associated delays.

2.3.3 High-Level Petri Nets

There are many kinds of high-level Petri nets. In this section, we only discuss two major high-level Petri nets: hierarchical Petri nets (HPN) and coloured Petri nets (CPN).

In HPN [76], the hierarchy construct, called a subnet represented with a rectangular box, is used to encapsulate and hide part of the system at a certain level. This subnet makes it possible to model a large and complex system in a hierarchical structure. Later, a system modeled in HPN can be comparatively comfortable to be reviewed and modified owing to the hierarchical structuring.

In CPN [68], the tokens are extended with color or type. Colored tokens represent objects in the system modeled. Each transition uses the token values in the places connected through its input arcs and produces the value for the places connected through its output arcs. CPN can be used to define a high-level timed Petri net such as dynamically timed Petri nets (DTPN) of which time constraints are defined using colored tokens.

2.3.4 Petri Net Markup Language (PNML)

PNML [57] is a tool interchangeable format in XML. Figure 2 shows the meta-model of PNML explained in [30]. A Petri net tool that supports the PNML file format can edit a Petri net stored in PNML files in a visual way. The high-level Petri nets are currently under standardization process as ISO/IEC 15909 and the transfer format is being developed as Part 2 of this standard (ISO/IEC 15909-2) [74]. Timed Petri nets like GSPN require a time value such as rate for each transition. The PNML specification strongly recommends such information to be placed under the <toolspecific> tag or represented using a syntax proposed for a generic Time Petri net based on the Mathematical Markup Language (MathML) [60].

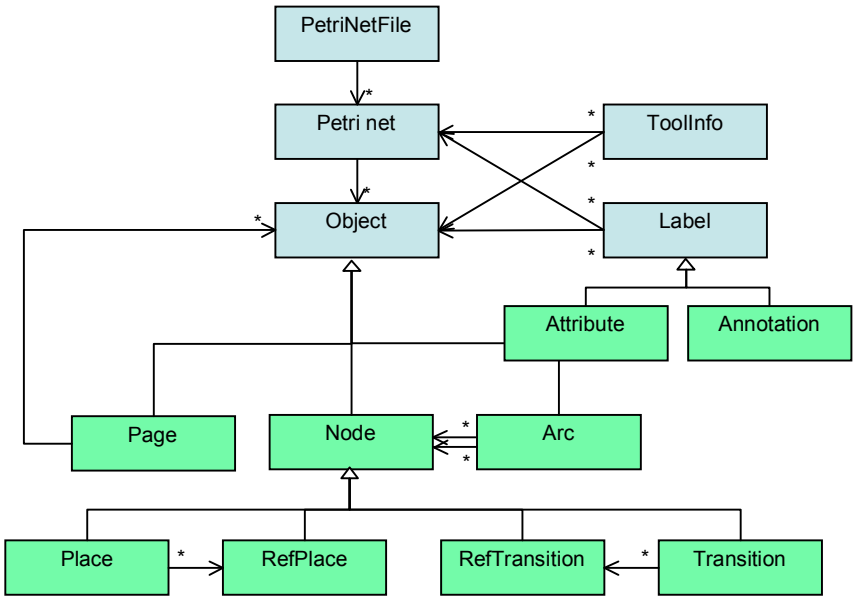


Figure 2: The meta-model of PNML

CHAPTER 3. SOFTWARE SYSTEM DECOMPSITON WITH ASPECTS

3.1 Introduction

Two major problems of OO methods, tyranny of dominant decomposition and code-tangling explained in Section 2.2, make it difficult to model software system architecture with various important concerns that should be carefully considered at the initial stage of the design process, which justifies a new decomposition method to support the separation of concerns principle. We use the Function-Class Decomposition (FCD) [4] as a cornerstone of a new extended decomposition methodology that provides better understanding of the system in the early system design phase by enhancing the principle of separation of concerns during the decomposition process.

FCD, a powerful hybrid decomposition method, was proposed to combine structured analysis methods with object-oriented methods and to provide a supportive architecture for modeling software systems. By generating a hierarchical view that presents the structural model of the software system, FCD can draw a clear alignment for identified classes in terms of functional modules. FCD can alleviate the tyranny of dominant decomposition problem by supporting another dimension based on functional decomposition process to extend the OO approach. However, FCD is not capable of capturing various important concerns required to build a software system and organizing them in a separated dimension. Furthermore, FCD inherently suffers from the code-tangling problem like other traditional decomposition methods. Consequently, the objective requirement for a new extended FCD method is to capture various important concerns during software system modeling without avoidable redundancy and loss of its advantages like simplicity. For this purpose, the concept of aspect is integrated with the original FCD.

To capture and organize various concerns appearing during the system decomposition, an extended FCD method called Function-Class decomposition with Aspects (FCD-A) [72] introduces the integrated concept of aspects and an additional decomposition view. FCD-A partitions a software system iteratively and progressively in two system-wide views: the function-class view and the aspect view. In the function-class view, the classes of a system are identified and grouped into subsystems to create a hierarchal model of a system. In the aspect view, aspects are used to identify, group and categorize the crosscutting concerns of the identified classes or subsystems. They are investigated to recognize subsets of a high-level initial set of aspects or common aspects after the validation procedure for grouping the classes in the function-class view. The two views of FCD-A are tightly associated through two types of links that represent the functional or non-functional relationships of the aspects in the aspect view with the functional modules or classes in the function-class view. When developing these two views repeatedly and alternately, we can specify a software system step by step in a spiral way similar to the Twin Peaks model [22]. The decomposition process model of FCD-A is illustrated in Figure 3.

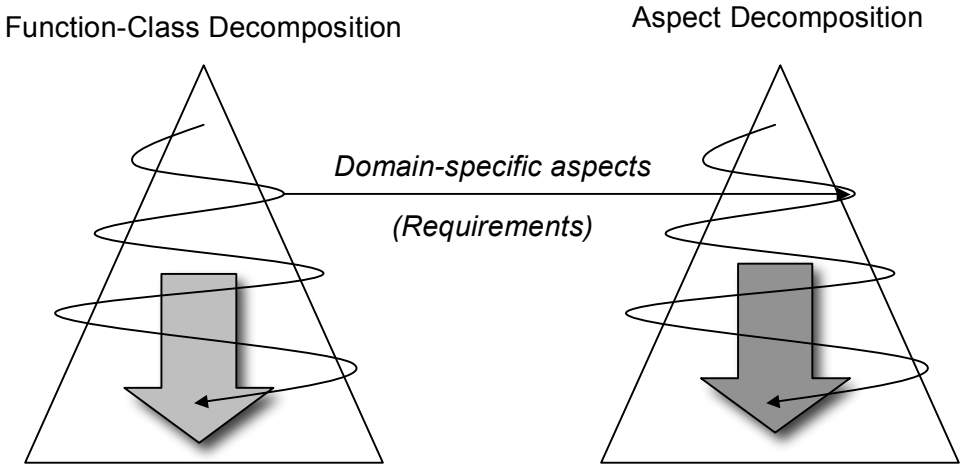


Figure 3: The process model of FCD-A

Since FCD-A is a generic method, it can be straightforwardly adapted to identify core functional features of services and isolate non-core functionality and quality features scattered across a service-oriented system.

3.2 Function-Class Decomposition with Aspect

This section discusses the concept of aspects integrated to FCD-A and explains its process.

3.2.1 Aspects

Concerns for a developing software system are important, or critical, properties that pertain to a specific domain. The concerns scattered across multiple classes in the most traditional OO approaches or tangled into existing codes of legacy implementations decrease the modularity and deteriorate the development and maintenance of software systems [25]. Aspect-oriented programming (AOP) that concentrates on the implementation phase of a software development cycle can encapsulate each concern in an isolated unit called aspect in order to improve the modularity and traceability of a system. This separation based on aspects can be applied to the system design process. In FCD-A, aspect-oriented technique is used to help identify the crosscutting concerns and separate them from the system functionality during the system decomposition process. Thus, we define aspect as follows.

Definition 2: *Aspect* is defined as a domain-specific concern for the system to group common functional features for functional requirements or to specify a property or policy for non-functional requirements.

A common functional feature is a group of operations, a class, or a functional module found in multiple classes, functional modules or a system, respectively. Policy addresses a

guideline on the behaviors of classes or functional modules, which means that an aspect can be a means to control or manage one or several classes or functional modules at the design stage. For example, the "synchronization" aspect identified in a communication module may not enunciate how to synchronize the classes of the communication module or what kind of method should be used, which will be determined in later steps or at the implementation stage. FCD-A can constitute a hierarchy of aspects by allowing each aspect to be further analyzed into child aspects.

3.2.2 Process of FCD-A

FCD provides a hierarchical structure of the system based on the functional modules. As the system analysis goes further, FCD progressively clarifies the function-class view and presents a more concrete system structure. This dominant view, however, cannot fulfill all the considerable facets of the developing system. Furthermore, the class grouping procedure of FCD for creating functional modules used in the next-level process cannot prevent a class or a functional module from repeatedly and redundantly appearing in different branches of the system's hierarchical structure. FCD-A extended from the original FCD method fixes these problems by introducing aspects and providing another new view for maintaining them, called the aspect view, in addition to the function-class view of a system. The aspect view gives a horizontal-cut view of the hierarchy in the function-class view. Each aspect in the aspect view is the abstract container to group all or a subset of requirements identified in multiple classes or functional modules. Consequently, the primary aim of the FCD-A algorithm becomes to generate the aspect view of a system.

Figure 4 shows the outline of the FCD-A process. The detailed algorithm of FCD-A is shown in Algorithm 1 in Appendix. The iterative decomposition process of FCD-A consists of identifying classes, allocating the related requirements to the proper classes, creating links between the classes or functional modules and the aspects, grouping classes

using scenarios, building functional modules using grouped classes, creating child or common aspects, and allocating scenarios to the functional modules. Note that this process preserves the top-down approach in the original FCD. The completion of system decomposition is followed by the integration of functional modules and aspects based on the system development environment such as the programming language adopted.

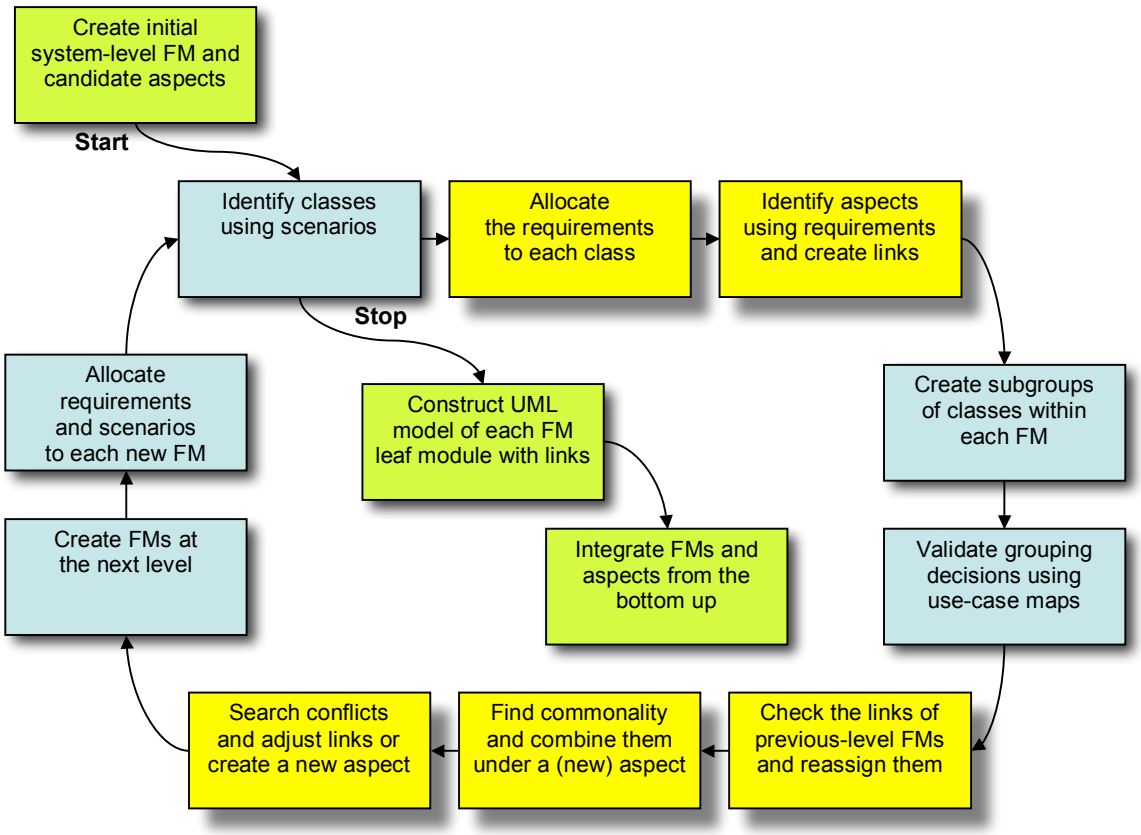


Figure 4: Process of FCD-A

3.3 Example

We will apply the FCD-A method to the retailer system. Another example of the application of the FCD-A method on the M-Net [24], an Internet-based real-time

conferencing system, is presented in [72]. Note that the retailer system will be used as an exemplary service-oriented system in the remainder of this dissertation.

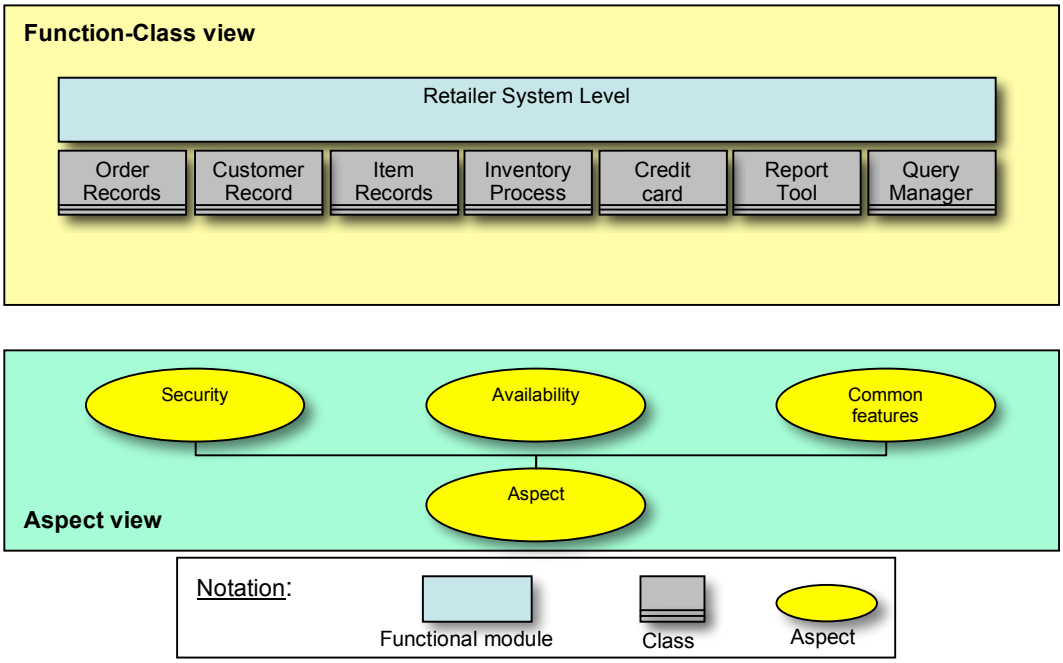


Figure 5: The initial classes with the system-level functional module and the key aspects to be considered

The initial level of analyzing a system starts with a single functional module in the functional view and a few key aspects then need to be considered during decomposition. The top-level key aspects at the initial level can include common functional features like logging or quality features like security, availability, reliability, and so on. They can vary according to the domain or the characteristics of a system under development. Figure 5 shows an initial functional model with the identified classes and the three key aspects of the retailer system. As you can see in the aspect view of Figure 5, we want to identify common features, security and availability of the retailer system during the decomposition process. The "Common

features" aspect is used to combine redundant functional or non-functional features appearing across a retailer system. Those three initial key aspects are placed under the aspect named "aspects" of which role is to provide the root of a hierarchy in the aspect view.

After the functional and non-functional requirements are reallocated to each identified classes, they are used to create *aspect links* from theses classes or functional modules to appropriate aspects. To reflect the functional and non-functional concerns of the system, an aspect link can be either of two types: *operational* or *strategic*. In other words, the type of an aspect link is determined based on the allocated requirement for the classes. The operational link depicted as a sold line is used to indicate an implementable functional feature that is identified as an aspect and required to be embedded into classes in the OO approach. The strategic link depicted as a dotted line is applied to address an abstract method or an annotation that does not need to belong to the associated classes. During the investigation of aspect links, the strategic link can be moved up to a subgroup when all the classes within the subgroup contain the same strategic link, which is impossible for the operational link. Since the strategic link for a functional module affects the behavior of all its classes, it may be implemented independently such as a monitor class. An aspect link belongs to both a class, or a functional module, and an aspect. These characteristics make our method flexible. Each aspect link can be implemented as a method within the classes in the functional view. On the other hand, the aspect links appertaining to an aspect can be implemented as an independent aspect class.

Figure 6 shows the allocation of the aspect links for the initial identified classes. For example, since each customer needs a password to access its customer record based on the allocated requirement for the classes, an operational link are attached from the customer record class to the security aspect. Only the customer who places the corresponding orders or the service provider who deals with them can access the order records. Therefore, a strategic link is applied from the order records to the security aspect. This strategic link is named

"User ID" since we think that a User ID can be available to identify and check who access this class. Similarly, three strategic links are drawn to associate the inventory process class, the credit card process class and the report tool class with the security aspect because their access requires a verification method based on an Order No. Five classes in the function-class view need to access a system database to keep their information, which is represented using the operation links named either "Update DB" or "Search DB" with a key value. If the "Integrity" aspect is included as one of the initial key aspects, it could have been operational links directed from those classes related to the system database.

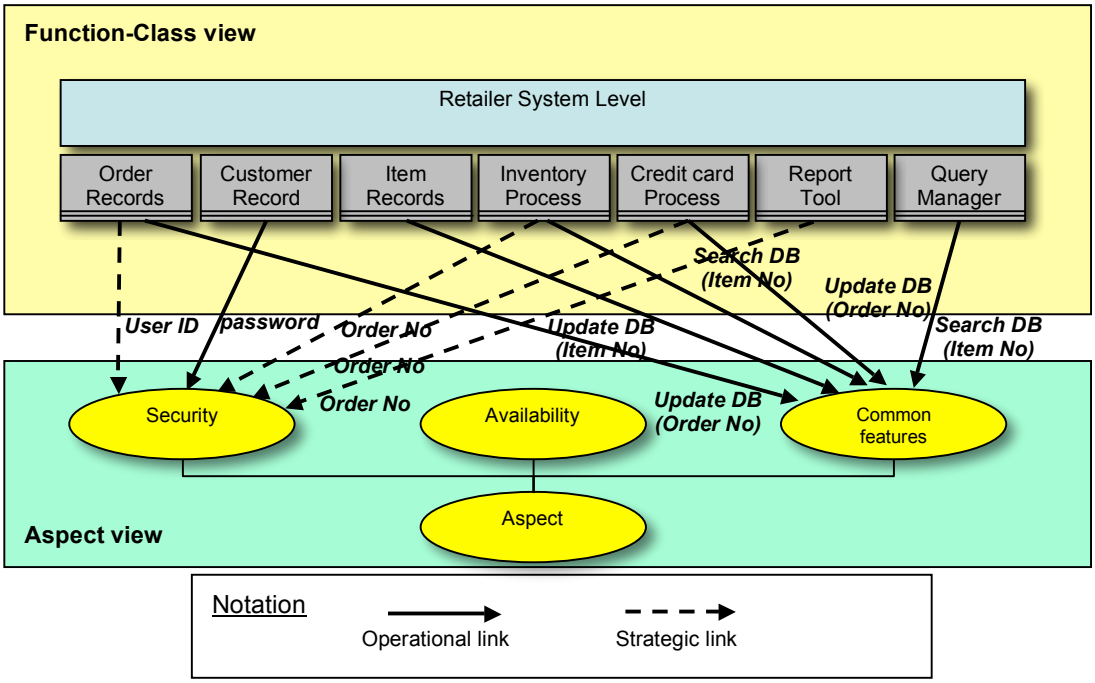


Figure 6: Allocating aspect links to the initially identified classes

After creating aspect links from nodes in the function-class view to aspects in the aspect view, FCD-A performs class grouping to align the classes in a structural model and reveal the relationships among classes or functional modules. FCD-A performs the first class

grouping of the level 1 according to the high-level functionality of two services and customers. The result of the class grouping is well reconciled with the cohesion and coupling criteria for class grouping of later levels. In addition, it helps to develop and categorize child aspects (or sub-aspect) in the next aspect categorization process.

The aspect links created need to be investigated from the aspect view after the grouping of the classes in the function-class view has been done. This step allows us to create a new child aspect by examining commonality and conflict of the names of the aspect links. Here, we will explain using the six classes of the original seven. After the grouping procedure identifies the functional modules named the customer and the order service in the function-class view, we create four child aspects in the aspect view. The "Security" aspect has the "Authentication" aspect derived from the operational link from the customer records and the "Authorization" aspect based on three strategic links from the Inventory process class, the Credit Card process class and the Reporting tool class. Actually, the three common strategic links from those three classes are replaced with a new strategic link from the identified order service functional module. This new strategic link intentionally includes the Item Records class that does not have a strategic link to the security aspect in the previous step, so that it can enhance security over the whole process of the order service. At the later stage of the decomposition, this link will be rearranged if all classes or new subgroups of the order service do not satisfy the requirements of the "Authorization" aspect. In a similar way, the "common features" aspect has the "search DB" aspect and the "Update DB" aspect based on the four operational links. These two child aspects could be replaced with a single child aspect named "Access DB" to decrease the decomposition complexity if necessary. Since the "search DB" and "Update DB" aspects already implies a primary function, the operational links have the remaining part called parameters that were enclosed in parentheses in Figure 6. No conflict among the aspect links can be found at this level. As a result, Figure 7 presents the analysis result for the six classes of two functional modules in the function-class view,

and the related aspects in the aspect view just before new requirements allocation for identifying additional classes for the next level decomposition. The two aspects in the aspect view have child aspects. This figure validates the first level grouping and aspect analysis. There is no undesired cohesion or coupling in its path when we examine a mapped scenario that denotes a customer places a new order to the order service.

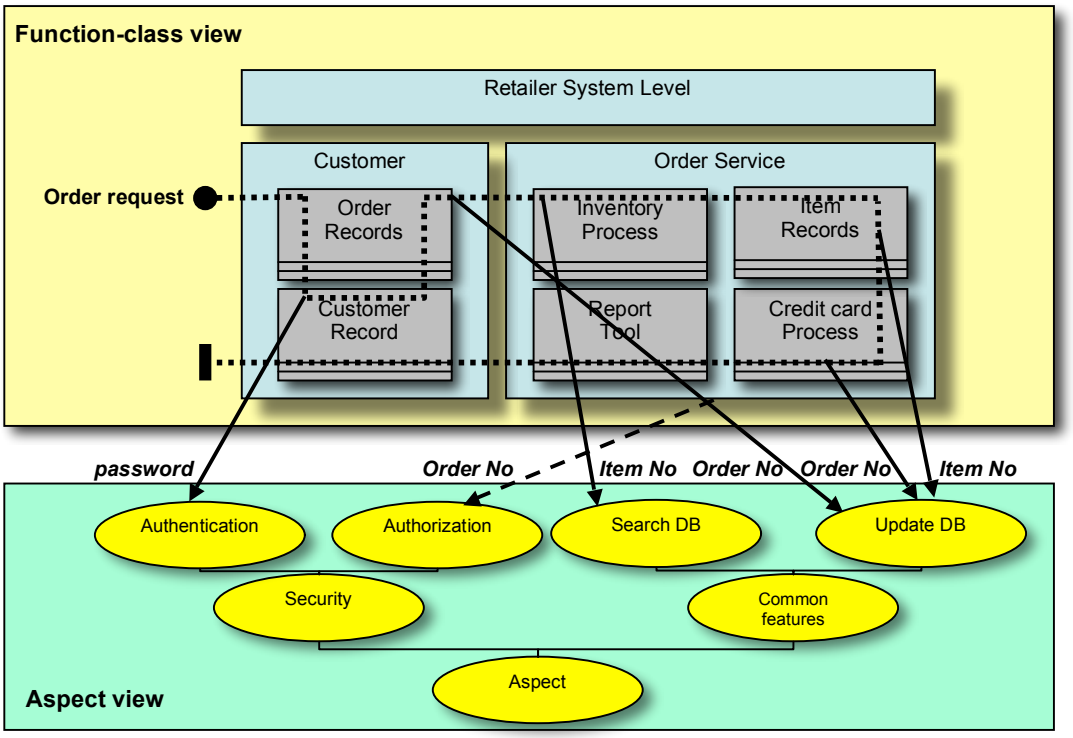


Figure 7: Analysis of aspect links after initial grouping decisions

Figure 8 shows the entire system view after analyzing aspects and the aspect links of which names are omitted for space reasons. The hierarchical aspectual model in the aspect view provides a framework to classify aspects. Note that this hierarchical aspectual model depends on the key aspects selected at the initial step and child aspects created during the decomposition process.

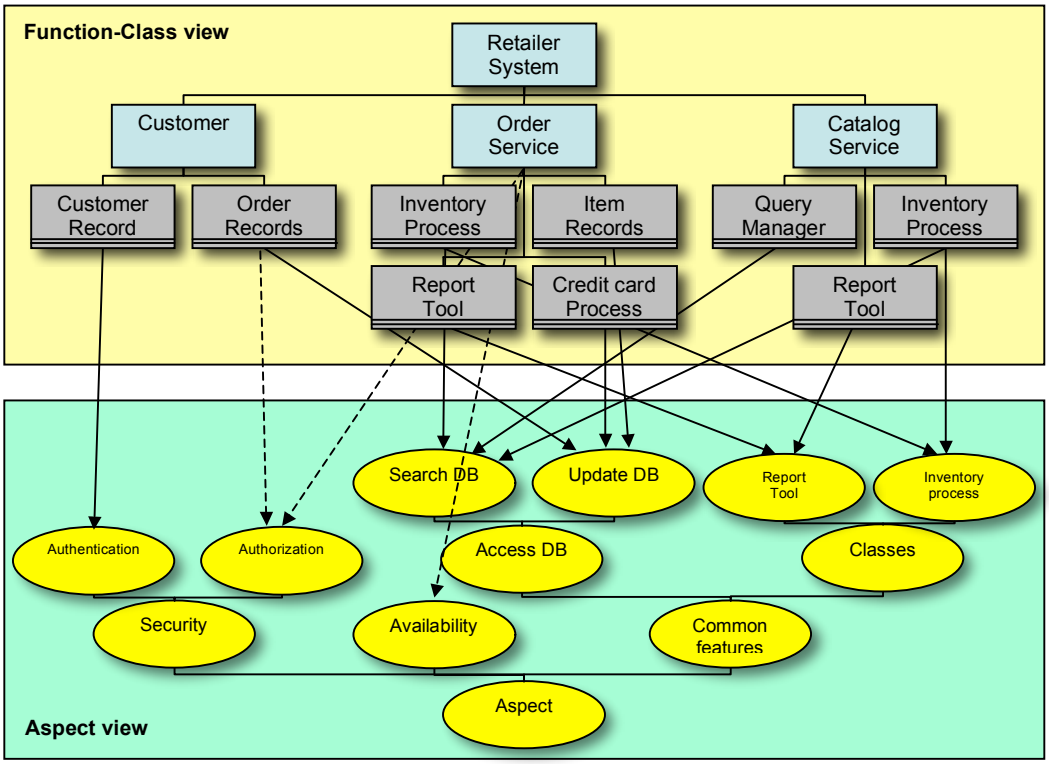


Figure 8: Function-class and aspect views of the retailer system prior to identifying additional classes for the next level decomposition

After the primary classes and aspects are identified, the entire system can be integrated by defining the interfaces between the intermediate-level functional modules, starting from the lowest level. This integration stage is dependent upon the system development environment as well as the preference of the developer. For example, a programming language applied to build the system can determinate how to implement aspect links. General purpose programming languages can reflect them by means of design patterns. OO programming language like Java can realize them in the interface or using abstract classes. An aspect-oriented programming language like AspectJ [7] make it possible to implement them in a straightforward manner. Some of the identified aspects may not appear

at other development stages. For instance, a “testability” aspect used for the whole system development cycle would never appear in the final system ready to deliver to customers.

3.4 Related Work

There have been a number of approaches to extend the AOP paradigm at the earlier stage of the software development process.

Aspect-oriented Component Requirements Engineering (AOCRE) has been proposed in [16]. Functional and non-functional requirements of a component are related to key aspects that other components provide or require. By analyzing general or components requirements, the aspects in the components are identified to distinguish the functionality and interface of the components. These identified aspects at requirements-level can be directly mapped into the design-level software component via interfaces, language reflection or design patterns. However, they do not clearly explain how to identify aspects in components using requirements.

To identify aspects during requirements engineering, a model for Aspect-Oriented Requirements Engineering (AORE) is presented in [18][19]. Once concerns and stakeholders’ requirements are specified, they are evaluated to find candidate aspects using a relationship matrix. These candidate aspects and stakeholders’ requirements are composed by means of the defined composition rules. Then, the contribution between the candidate aspects is used to prioritize and resolve the conflicts between aspects and stakeholders’ requirements. AORE model focuses on identifying aspects from stakeholders’ views and resolving their conflict at requirements engineering level, but it does not treat the mapping with software artifacts of the other stages of the software life cycle.

The Non-functional requirement (NFR) framework is also extended to find links between functional requirements and crosscutting non-functional ones [15]. The NFR framework analyzes non-functional requirements to select appropriate operationalizations

that are detailed solutions for the system. Each operationalization is categorized as an aspect or an architectural decision based on the influence of the software artifacts of later stages. Then, the identified aspects are composed into functional requirements with aspect semantic operators.

Composition patterns evolved from subject-oriented programming [5] are the extension of the UML to increase the reusability of aspects [6]. The composition pattern of a cross-cutting behavior like tracing is defined as a template with parameters and bound with a base design. Again, composition patterns do not specify how to capture these patterns. In a complex software system, the identification of these patterns is not easy.

COSMOS (Concern-Space Modeling Schema) [20] consists of two types of concerns – logical and physical concerns, four types of relationships and predicates in order to support a general-purpose representation for arbitrary concerns. Although types of concerns and relationships are well categorized, predicates that enforce a consistency or a rule on concerns remain as an open area.

For concurrent software architecture, the weakness of OO approaches and the AOP-related issues were well discussed in [8]. To improve concurrent software quality in an OO environment, an aspect-oriented framework (AOF) architecture was suggested in [14]. In the AOF architecture, aspects are regarded as specifications such as the preconditions or postconditions of a concurrent object, which seems to only partially cover the meaning of the aspect for the design level. Also this framework does not describe a method to define the relationship between components and aspects.

The Aspect-Oriented Design (AOD) [17] is based on the concept of software architecture. Aspects are presented in architecture styles. Both functional design and aspect design are performed independently and, then, weaved to deliver the final architecture. However, AOD dose not have a mechanism to weave architectures. In [21], an aspect acts as a building block or a module in the proposed aspect architecture. Each aspect is represented

in a stereotype of a package. A concern diagram, which is a new UML diagram type, shows the dependencies between aspects and overlapping aspects of different concerns. Several aspects can be combined and replaced by a composite aspect to treat a concern. This composition and the changed dependencies between concerns lead to a new concern diagram to show the different architectural view. Thus, an aspect should be well divided or re-modularized into sub-aspects that are the overlapping parts of different concerns. In the example, the concerns to divide the system are based on only low-level functionality, not high-level quality-attributes, although their definition of concerns covers non-functional features.

AOP have been applied in the designing and implementation of an agent system [23]. Here, agent roles and role models correspond to classes and aspects, respectively. The role model catalog at British Telecom Labs is used to design and analyze the agent system. The identified aspects from the role model are implemented in AspectJ [7]. Their main goal is improving the Role Object design for agent system.

An aspect-oriented decomposition for behavior of process-control systems has been proposed in [9]. It uses a state machine notation for specifying system behavior and produces independent components with partial system reliability. This approach is useful for decomposing a system based on some specific system requirements, such as reliability, in terms of dynamic behavior of the system.

The necessity and advantages of identifying aspects in the earlier life cycle of the software development have been deeply discussed in [10]. Here, the main objective is to propose a method to capture and pass aspects in two different phases, requirements engineering and architecture design. Compared with it, our approach is a more integrated and procedural solution to exploit the concept of aspect in a design phase by supporting the linking between requirement aspects and architectural aspects and the mapping between architectural elements in the function-class view and architectural aspects in the aspect view.

The Theme approach [11] for identifying aspects from behaviors in requirements specifications is proposed. It represents aspects and their interactions using an extended UML without supporting an architectural view.

As a specific requirement-level instantiated version of [2], the decomposition of requirements into multiple dimensions discussed in [13] uses multiple concrete concerns derived from abstract concerns in a meta concern space. Then, trade-offs analysis of a specific concern is conducted using the contribution matrix in which each cell shows the relationship between two concrete concerns in terms of the other concerns affecting to them. Although this approach may be useful to evaluate candidate architectures, it is not helpful to construct an architecture since it does not support a way to categorize concrete concerns that coexist in a flat system space and extract some of them to be essentially realized.

The aspectual software architecture analysis method (ASAAM) [12] provides a systematic scenario-based architecture analysis method to identify crosscutting concerns across architectural components using a set of heuristic rules. A scenario scattered across components is considered as an aspectual scenario that is used to identify whether a component is either tangled or ill-defined and to be the aspectual aspects derived from a domain analysis process. This approach, however, represents all architectural components and aspectual aspects in a single flat view.

3.5 Summary and Discussion

In this chapter, we present the FCD-A method that supports the fundamental concept of the aspect at the design process. The FCD-A method decomposes a software system with multiple concerns as a top-down approach using the two views. While a system is gradually broken down into functional modules with the corresponding classes in the function-class view, the aspects for domain-specific concerns like various system properties are identified, separated and categorized simultaneously in the aspect view. The two views of FCD-A are

connected via two types of aspect links that enhance the traceability of a system. When system requirements are changed, the scope of changes can be determined using either function-class view or aspect view. For example, if a password mechanism to a security requirement in Figure 7 needs to be replaced with a smart card, it is obvious that the scope of changes is restricted to the single class. This traceability can be identified by checking the aspect links from the aspect view, not from the function-class view, which addresses how to treat the traceability of the requirements and determine the scope of changes against the change requests of requirements.

The searching process for commonalities and conflicts among aspect links may not be easy as the decomposition goes on. Although both function-class and aspect hierarchies help limit the checking scope of the aspect links, it is not sufficient especially for the complex system. Therefore, a dictionary feature could be useful to help name the aspect link and discover the related or opposite meaning in the names of the aspect links.

In particular, service-oriented systems tend to be extremely sensitive to quality attributes, not only functionality. Therefore, the aspects related to non-core functional and non-functional requirements play an important role in the delivery of user-friendly and high-quality services. It also happens quite often that some features are overlapped in multiple services in a service-oriented system due to the partial similarity of the processes of these services. Some significant quality attributes of services, such as availability, adaptability and security, need to be effectively identified to build flexible and quality-guaranteed service-oriented systems without redundancy. We believe that the FCD-A method is capable to be straightforwardly applied to any service-oriented design process in the field of service-oriented computing.

CHAPTER 4. SERVICE-ORINETED DESIGN WITH ASPECTS

4.1 Introduction

Services as autonomous abstract units can be structurally composed into a complex service and grouped in a service-oriented system. Each service tends to be treated as a process in SOC. Hence, designing a service involving two different views, the structural view and the behavioral view, is a tricky and intricate process. Services should satisfy customers' taste with supplementary functional and non-functional features. Moreover, services should evolve according to the rapid changing e-service market trends and demands. Even a current working service has to be adapted in order to support a wide spectrum of customers. This requires not only an agile development of services, but also support of different versions of a service for a service provider to satisfy various context and preferences of customers. Practically speaking, the concurrent development of different versions of a service, usually required to provide differential services, is considerably challenging. In order to support these requirements for service development in the design process and to keep pace with this shifting environment of SOC, the service design process for SOA needs to be efficient to quickly build a service, extensible to modify or update specific features in services, and generic to make possible "design in the large" like WS-BPEL that does "programming in the large" by treating each web service as a process. In addition, service design results can be reusable and interchangeable for service developers to share common features appearing across varying services or multiple service-oriented systems.

In this chapter, we explain a generic service-oriented design approach with the concept of aspect called the Service-Oriented Design with Aspects (SODA) [73]. Services are structurally decomposed into primitive services that have only essential functional features and aspects that include other supplementary functional or non-functional features

for services. Aspects are used to capture and encapsulate service-specific concerns required for delivering high-quality and user-friendly services.

The structure of a decomposed service-oriented system is represented using an extended UML2 [28] whereas the behaviors of the functional features within services or aspects are described using the Petri net. The interchangeability of all the design results is guaranteed by means of a set of XML-based description languages called service-oriented definition markup language (SODML) that includes the Petri net markup language (PNML) explained in Section 2.3.4. We can efficiently modify a service just by replacing primitive services, aspects, or their relationships in a service design. The different versions of variations of a service also can be generated by means of changing or modifying aspects or relationships. The aspect deserves as a reusable unit, which means that aspects applied to a service design can be reused in the development of other service designs. Most of all, weaving process can generate the behavior description of the composed service by weaving aspects with a primitive service in the terms of the Petri net-based semantics. The Petri net generated via weaving process is stored in PNML can be examined and analyzed using the Petri net tools that support PNML as their input file format. Consequently, the generated Petri nets for the composed service facilitates the analysis of service design results or the selection of the appropriate service design among several candidate designs.

4.2 Services and Aspects

The SODA approach uses two abstraction mechanisms to decompose a service-oriented system in the basis of two abstraction mechanisms, *service* and *aspect*. Due to these two abstraction mechanisms, our design approach structurally arranges design elements for services in two views, the service view for representing the basic architecture of a service-oriented system and the aspect view for grouping the aspects separated from it. In particular, the services in the service view are called *primitive* services that represent basic functional

features and interfaces bound with it. Supplementary functional or quality features related to the primitive services are encapsulated into aspects in the aspect view.

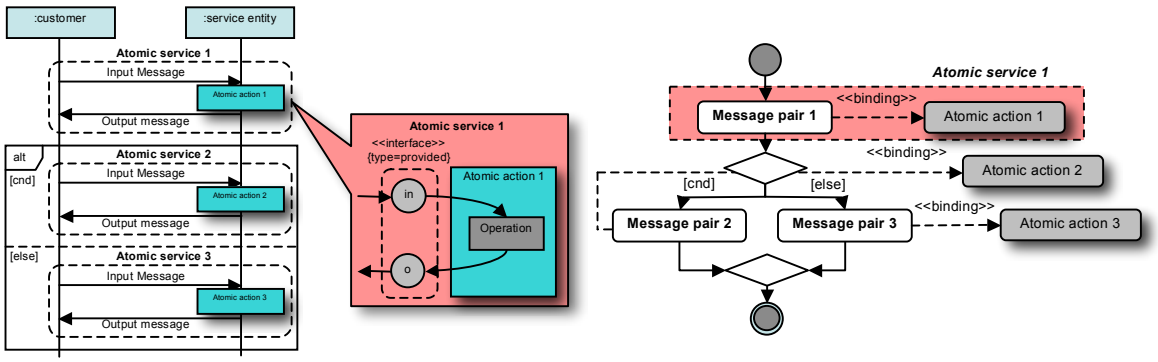
4.2.1 Services

As a self-description unit, a service can deliver a specific output of performing an action or a task to customers. The type of a service is either *atomic* or *sequential* depending on how many interactions are required with its customer. Similar distinction for services can be found in OWL-S [58] that uses the terms atomic (simple) and composite (complex) for this purpose.

Each interaction requires *a message pair* that consists of an input message for requesting the invocation of a service and an output message for delivering the result of its processing. A message pair works as a basic element to define the interface bound with a service. An interface becomes either *provided* or *required* according to the intention of its binding with a service – to allow the service to serve others or to want it to be assisted by other services. In particular, a service is said to be *autonomous* if it does not have any required interface, i.e., an autonomous service can deliver a single service by itself and be deployed independently.

Although an interface can be defined independently from the service, it is identified during the analysis process of a service. The UML use case diagram seemingly is sufficient in order to identify a message pair for an atomic service during a single interaction whilst the UML sequence diagram is necessary to identify multiple message pairs for a sequential service. Then, the interface of each service is defined in terms of message pairs required during the interaction with customers. Accordingly, the number of its message pairs becomes equal to the one of the interactions required for a service. A service must be bound with a provided interface to provide the gateway of a consistent service access to its customers.

From the viewpoint of the customer, a service seems to perform a transactional process called the *atomic action* between a message pair for a single interaction. In the SODA process, the atomic action is regarded as a basic descriptive unit for a transactional behavior in a primitive service or an aspect. Just like the class in object-oriented design, an atomic action is treated as the reusable design element in our approach so that it can be developed and managed independently of other atomic actions as well as interfaces. An atomic action consists of operations and messages that comprise a procedural flow required for processing a service in the behavioral (or process) view. For example, Figure 9-(a) illustrates the sequence diagram of a primitive service that performs two atomic actions (either 1-2 or 1-3). In proportion to the granularity of the service in the design phase, operations as transactions that represent single logical units of work [83] can be mapped to procedures, tasks, or stand-alone services while messages can be simple values or complex XML documents. This mapping can be deferred to the later stages of the service development process.



(a) Sequential diagram **(b) Activity diagram**

Figure 9: A sequential service with three atomic services

To describe the atomic action graphically and formally, the Petri net is used. The Petri net as a high-level design tool is suitable to draw and verify dynamic characteristic of service

behavior. Compared with other methods like the state machine for a service description [46], the Petri net can effectively express the parallel or concurrent operations in atomic actions. It is possible for the basic Petri net for an atomic action to be extended to a high-level Petri net like CPN because of a special necessity during the service development process. However, the use of a high-level Petri net leads to aggravating the complexity of a design process. Our design approach sticks to the basic Petri net. Hence, the formal definition of the atomic action is presented in the following paragraph.

Definition 3: An *atomic action* represents a procedural flow during a single interaction with the customer. It is comprised of one or more operations and messages among them. The atomic action is modeled using the Petri net shown in Definition 1 such that the Petri net for an atomic action $A = (P, T, I, O, M_0)$ where P and T are mapped to messages and operations in a service description, respectively. Tokens in a place denote instantiations of the message. Generally, the Petri net for an atomic action is ordinary.

By regarding the message pairs required for the provided interface of a sequential service as abstract actions, an activity diagram can represent the provided interface of a sequential service. This activity diagram is useful for the customers who want to access the sequential service since they are not interested in the concrete atomic actions to be bound. For example, the activity diagram in Figure 9-(b) outlines the interaction order of the sequential service in Figure 9-(a). In our design approach, the description of an interface can be orthogonal to the one of an atomic action. To make a service deliverable in any case, a binding mechanism needs to erect a bridge between the interface and a set of atomic actions. The binding mechanism makes it possible that a service can be derived via different provided interfaces. If a platform or a framework supports a dynamic binding mechanism, it can deliver different services thorough the same provided interface. As a result, a service can be

constructed by binding a provided interface with a set of atomic action. During the binding process, each atomic action bound with a message pair becomes an atomic service. For example, the "atomic service 1" in Figure 9-(a) is constructed by associating the single operation for the "atomic action 1" with the "in" and "o" places added for a message pair in the provided interface. Formally, the atomic service built by binding of a message pair in a provided interface with an atomic action is specified as follows.

Definition 4: An *atomic service* is an atomic action accessible from the customer via a provided interface. From the viewpoint of the service provider, an atomic action is initiated by receiving an input message from the customer and terminated by sending an output message to it. To be precise, an atomic action A becomes an atomic service S by binding with a message pair mp m $mp = (p_i, p_o)$ where p_i and p_o are an input and an output message in a provided interface definition that is published to customers. Let $T_I, T_O \subseteq T$ be the set of transitions bound to p_i and p_o respectively. Then, an atomic service S is expressed using the binding operator $\oplus$ such that $S = A \underset{(T_I, T_O)}{\oplus} mp$. The operator $\oplus$ constructs the Petri net for S by performing Algorithm 2 (the binding algorithm) in Appendix.

The binding algorithm does not allow the binding process to create a nondeterministic choice or unsynchronized termination using a special operation "fork" or "join" that makes several operations simultaneously executed from the input message or synchronously terminated before an output message is created, respectively.

4.2.2 Aspects

In addition to atomic services that describe the basic functionality of a service, it is important to consider many extra functional and non-functional features that help increase the quality or value of the service in the process of service development. From a bird's eye

view of an entire service-oriented system, the same feature tends to appear across multiple services. It is possible that a feature appears several times in a single service. These scattered features make it hard to abstract a service into a single unit and maintain a service-oriented system. By encapsulating these extrinsic or supplementary features scattered across a service-oriented system using the concept of the aspect, we can enhance the modularity and traceability of a service-oriented system. For the service-oriented design, we suggest three generic types of high-level aspects: *common*, *policy (or property)* and *value-added*. The first two aspects are derived based on the aspect definition in Definition 2. Compared with the aspect in FCD-A, the aspect in SODA contains a more concrete feature such as a behavioral description or a detailed specification.

The common aspect is used to denote a structural or functional redundancy in a service-oriented system. Its goal is to keep the same version or copy of a design element during a service-oriented system development. Especially, when several services for a system are designed in parallel, it happens that different services have the identical design elements varying from a message to a service itself. For example, the same atomic action could be found in several services. Note that the common aspect is not used to separate extra functional features from a primitive service. Usually, a common aspect would be realized using shared classes, components, or libraries, but an AOP language might be useful to implement it.

The policy aspect is used to classify the quality attributes for a service-oriented system such as security, performance, availability and so on. Its objective is to provide a behavioral guideline on service or its element or a means to control or manage one or several services. Some policy aspects with a concrete behavior intervene in a primitive service behavior. For example, a policy aspect for security enforces an access control to services. This security aspect could be implemented in a monitor class or a component that uses an authentication mechanism or probes all the messages produced inside a service if an aspect-

oriented language is not available at the implementation phase. Other policy aspects that do not interrupt the execution of a service could not be materialized at the implementation phase. For instance, a performance aspect cannot be implemented as a software module and may be used for validation or testing in the later phases.

The value-added aspect is used to customize the basic functions of services according to customer's context or preference. The context means any information that can be used to characterize the situation of a customer or a service [31]. The primary goal of value-added aspects is to satisfy customers' preference and improve their convenience by altering the messages or operations in a service. Recently, non-intrinsic functional features play a very critical role in the survival of a service in a competitive e-service marketplace. For example, the functions for context-awareness or adaptation become essential for a service to be offered in a ubiquitous environment. Without them, customers who want to use a service in their device are compelled to type extra information or transform the service output into a device-specific format. The value-added aspect makes it possible to group these non-intrinsic functions in terms of the set of customers, not services. Moreover, the value-added aspect deserves as a reusable unit. Suppose that a value-added aspect that converts graphic files helps a map service adapt to the display type of customer's device. It can be reused in other services that need to reformat their output message into a graphic file format. The difference between value-added aspects and policy aspects relies on how much an aspect is related to the customer's context. On this basis, a performance-related aspect cannot be value-added, but a security aspect may be policy-type if it supports extra functions over the existing security mechanism or it is an option that requires an additional cost of customers. An AOP language would be very useful to modularize value-added functions in the implementation stage, although they could be realized using classes, components, or libraries.

4.3 Graphical Representation

Our approach graphically represents a service-oriented system by means of an extended UML2 diagram and Petri net. Precisely, the conceptual structure of a service-oriented system designed is represented using an extended UML while the behavior of each service delivered to the customer is draw using the Petri net. The support of a corresponding XML-based representation method can increase interchangeability or interoperability of service-oriented design results and make easier the implementation of a weaving program independent of any specific development tool, which is explained in the next section.

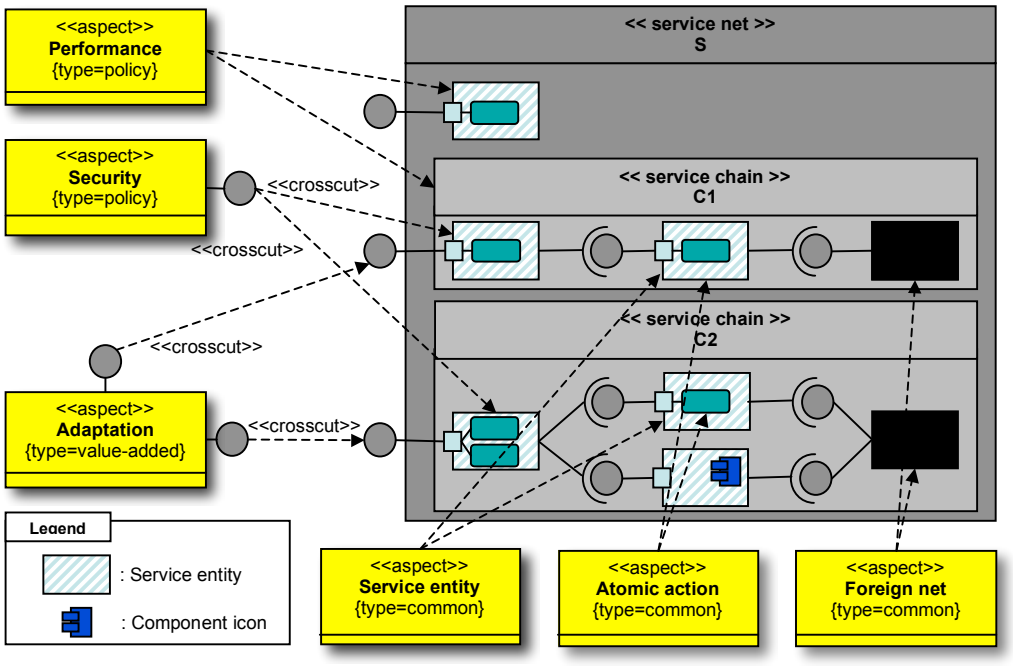


Figure 10: A service-oriented system with a service entity and two service chains plus aspects

To draw the conceptual structure of a service-oriented system, we can use three types of service elements – the service entity, the service chain and the service net, aspects and the

crosscut relationships between them. The service elements (i.e., service entities, service chains, and service nets) and aspects are represented using the extended UML2 composite structure. The crosscut relationships are denoted by the arrows with the stereotype <<crosscut>>. The provided and the required interfaces bound to the atomic action in a service entity are denoted using the ball-and-socket notation.

Figure 10 represents the design diagram for a service-oriented system that comprises the service elements responsible for three different services, aspects and their crosscut relationships. We discuss the service design elements shown in Figure 10 in the following subsections. Not only does this design diagram show fundamental design elements required for a service-oriented system in the development phase, but it is also useful at the deployment phase of a service-oriented system across heterogeneous platforms.

As mentioned, an atomic action in a service entity or an aspect is drawn using the Petri net that is likely to be well aligned with the UML graphical representation. Note that the main advantage of using Petri net is that Petri net as a graphical and mathematical modeling method is supported by well-defined formality and many available Petri net tools. In the rigid sense, the activity diagram of UML may be another option to describe an internal behavior of a service or an aspect. This alternative is less attractive since the weak formality of the UML representation hinders the analysis of the service behavior. A UML activity diagram needs to be translated into an input format for a model checker according to a formal semantics in order to verify functional properties of design results such as workflow models [61]. Furthermore, the activity of UML2 is redesigned to use a Petri-like semantics [28]. As a superset of a Petri net with syntactic sugar, an activity diagram is possible to be translated to a variant of Petri net using the defined denotational semantics for the basic features of UML2 activity diagram [62].

4.3.1 Service Entity

A service entity denoted by the stereotype <<service entity>> as a basic design unit in the structural view is responsible to deliver a basic and specific service to the customer. Since a service entity contains atomic services that are basic design unit in the behavioral view, it manifests a primitive service in both the structural view and the behavioral view. The service entity is said to be autonomous, which means that it can deliver a single service by itself and be deployed independently. In practice, the mapping of a service entity varies from system to system according to its granularity. In other words, a service entity can be mapped to any modular unit ranging from a class to a system based on the designer's decision. The composite structure of a service entity could have a component icon if the service entity would be straightforwardly mapped to one or more components at the implementation stage. The following paragraph defines a service entity in terms of the behavioral view as well as the required interactions with customers.

Definition 5: Let $AA = \{A_1, A_2, ..., A_n\}$ be the set of atomic actions and $MP = \{mp_1, mp_2, ..., mp_n\}$ be an set of message pairs in an provided interface definition. A *service entity* E is a set of atomic services such that $E = \{S_1, S_2, ..., S_n\} = \{(A_1 \oplus mp_1), (A_2 \oplus mp_2), ..., (A_n \oplus mp_n)\} = AA \oplus MP$. If $|E| = 1$, the service entity E is *atomic*. Otherwise, E is *sequential*. In particular, an atomic service is said to be *static* when its binding is not changed during the service development process as well as after the service deployment.

4.3.2 Service Chain

A service chain denoted by the stereotype <<service chain>> represents a composite service that delivers a single service with cooperation of multiple service elements. A non-autonomous service entity needs the support of other services via its required interfaces.

This also happens when a service entity is mapped to a COTS (Commercial-Off-The-Shelf) component with its required interface. Occasionally, service designers need to extend functionality of an autonomous service entity. In this case, it is important to explicitly show how to assemble all relevant service elements. A service chain is used to describe the peer-to-peer relationships among those service elements. The service chain must be autonomous, which means that all the exposed required interfaces of service entities within a service chain must not remain unconnected. The provided interface of a service chain comes from a service element in it. Therefore, the service chain may be thought of as the Façade object [32] that provides a single interface for a subsystem or a service proxy that statically links some services. From the behavioral view, the linkage of two service elements is achieved at the level of the atomic services within them. The following definitions present how to link atomic services in different service entities. Then, a service chain is defined.

Definition 6: We extend the binding operation $\oplus$ to support the concatenation of two atomic services in different service entities. An atomic service $S_1 = (P_1, T_1, I_1, O_1, M_{01}) \in E_1$ with a message pair $mp_r = (p_q, p_a)$, where $p_q, p_a \in P_1$ are the query and the answer message respectively, derived from the required interface can be connected to a message pair of the provided interface $mp_2 = (p_{i2}, p_{o2})$ for another atomic service $S_2 = (P_2, T_2, I_2, O_2, M_{02}) \in E_2$. When $E_1 \neq E_2$, an atomic service S_H is expressed using the extended binding operation that concatenates two Petri nets for S_1 and S_2 such that $S_H = (P_H, T_H, I_H, O_H, M_{0H}) = S_1 \underset{mp_r}{\oplus} S_2 = (A_1 \oplus mp_1) \underset{mp_r}{\oplus} (A_2 \oplus mp_2) = A_H \oplus mp_1$. The Petri net for S_H is defined as follows:

- $P_H = P_1 \cup P_2$,
- $T_H = T_1 \cup T_2 \cup \{t_{query}, t_{answer}\}$,
- $I_H = I_1 \cup I_2 \cup \{(p_q, t_{query}), (p_{o2}, t_{answer})\}$,
- $O_H = O_1 \cup O_2 \cup \{(t_{query}, p_{i2}), (t_{answer}, p_a)\}$,
- $M_{0H} = M_{01} \cup M_{02}$.

Definition 7: The extended binding operator ⊕ is left-associative such that $S = S_1 \oplus S_2 \oplus S_3 = (S_1 \oplus S_2) \oplus S_3$ and $S_1 \oplus S_2 \neq S_2 \oplus S_1$. Also, the operator ⊕ can be used to express the case that the right operand is a set of atomic services if an atomic service for the right operand has a set of message pairs $MP_r = \{mp_{r1}, mp_{r2}, ..., mp_{rm}\}$ derived from the required interface such that $S_H = S_0 \underset{MP_r}{\oplus} \{S_1, S_2, ..., S_n\} = S_0 \underset{mp_{r1,}}{\oplus} S_1 \underset{mp_{r2,}}{\oplus} ... \underset{mp_{rm}}{\oplus} S_n = S_0 \underset{mp_{r1,}}{\oplus} (A_1 \oplus mp_1) \underset{mp_{r2,}}{\oplus} ... \underset{mp_{rm}}{\oplus} (A_n \oplus mp_n)$, where if $S_0 \in E_0$ then $S_1, ..., S_n \notin E_0$. Similarly, the binding operator ⊕ in Definition 4 is also left-associate by considering a message pair *mp* as a Petri net that comprises two places such that $S = A \oplus mp_1 \oplus mp_2 = (A \oplus mp_1) \oplus mp_2$. In this case, an atomic service *S* has an atomic action *A* bound to a couple of provided interfaces.

Definition 8: A *service chain H* is a set of composite services such that $H = \{S_{H1}, S_{H2}, ..., S_{Hn}\}$. A composite service $S_{Hi} \in H$ is composed by concatenating atomic services in multiple service entities using the extended binding operator ⊕ in Definition 6 and 7 such that $S_{Hi} = S_1 \oplus ... S_i \oplus ... S_j \oplus ... \oplus S_n$, where $S_i \in E_i$, $S_j \in E_j$, and $E_i \neq E_j$. If $|H| = 1$, a service chain *H* is *atomic*. Otherwise, *H* is *sequential*.

4.3.3 Service Net

A service net denoted by the stereotype <<service net>> represents a collection of assorted services offered by a service-oriented system or a service provider. For example, a travel service-oriented system provides a set of services such as booking and check-in. The service net just depicts a virtual boundary of a service-oriented system using service entities or service chains that are really responsible for delivering real services. A service net pertaining to a service chain or another service net is called the *foreign service net*. A foreign

service net is considered as an entire black box, which implies that we can assume that the services of a foreign service net are provided by autonomous service entities of which atomic actions have a single operation inferred from their provided interfaces. The foreign service net, for example, could be used to illustrate a portal service net that only contains a set of foreign service nets. In this case all the service requests to this portal service net are redirected to the corresponding service net represented using a foreign service net. The definition of the service net is as follows.

Definition 9: Let $EE = \{E_1, E_2, ..., E_n\}$ be the set of service entities and $HH = \{H_1, H_2, ..., H_m\}$ be a set of service chains. A *service net SN* is an union of the above sets such that $SN = EE \cup HH$. Also, a service net SN contains a set of service nets $FSN = \{SN_1, SN_2, ..., SN_k\}$ where $SN \notin FSN$ such that $SN = EE \cup HH \cup FSN$.

4.3.4 Aspect

The aspect denoted by the stereotype <<aspect>> collects various atomic services or annotations in terms of domain-specific or service-related concerns. The type of an aspect is specified in the tagged value "type". In addition to three generic high-level aspects suggested in Section 4.2.2, another high-level aspect or child aspects under them can be created if needed. For instance, a logging aspect with a tagged value "{type=common/action}" indicates that it has a common atomic action shared by a number of service entities. Similarly, an aspect about the response time may have a tagged value "{type=policy/performance}. Based on these type values, aspects can be classified in a hierarchy structure that helps the traceability and maintainability of service design results. Any change related for aspects like creating or renaming must be immediately announce to all designers in order to keep the integrity of to the hierarchal structure. Note that these type

values could be determined based on the aspects names in the aspect view of the FCD-A method.

An aspect only includes autonomous atomic services to represent a functional feature separated. In other words, an aspect cannot have a required interface. This atomic service in an aspect is particularly called *advice* to distinguish it from the atomic service in a service entity to be crosscut. An advice is a concern-specific method that crosscuts any service element, which means that it intervenes in all the atomic services within a service element. An advice should be statically bound with a provided interface called the advice interface indicated using "ad" within the ball notation to provide a prescribed and consistent interface during the development process. An advice does not need to be considered as a black box in the design phases. The advice interface cannot be applied to an aspect with annotations for a quality attribute like the "performance" aspect in Figure 10. The aspect is defined as follows.

Definition 10: An *aspect* is a concern-specific set of annotations in a text-based form, or *static* and *autonomous* atomic services called *advices*. From behavioral viewpoint of a service design, an aspect $C = \{S_1, S_2, \ldots, S_n\}$ is the finite set of advices just like a service entity.

From now on, we use the service entity and the aspect to represent their atomic services and advices respectively except where an obvious distinction is necessary

4.3.5 Crosscut Relationship

The crosscut relationship denoted by the stereotype <<crosscut>> associates an aspect with a service element to be crosscut. In particular, the crosscut relationship from an advice should stipulate how and where it crosscuts an atomic service in a service entity. This crosscut-related information is specified within the "pointcut" tagged values in the crosscut

relationship using one of five crosscutting methods in Table 1. These crosscutting methods are derived from AspectJ [7], the aspect-oriented extension for the Java programming language. The *before* and the *after* method make an aspect performed prior to or posterior to an operation. These two methods can make an aspect work as either a pre-condition or a post-condition of a specific task required for Design by Contract [35]. The *around* method is used to replace a single operation in a service entity with an aspect. The *proceed* method to concurrently execute an aspect with a single operation. The *flow* method as a variation of the proceed method can concurrently execute an aspect by invoking the aspect before an operation and finishing it after another one within an atomic service. If a flow method has the two crosscut points of which operations are identical, it can be superseded with a proceed method. There is no variation version for the around method of which necessity strongly suggests that we should redesign or replace the service entity to be crosscut.

Table 1: The crosscutting methods

Method	Description	Syntax	Crosscut point
Before	Perform extra operations of an aspect before the operation in a service entity	Before(M_{PRE}, O)	(M_{PRE}, O)
After	Perform extra operations of an aspect after the operation in a service entity	After(M_{POST}, O)	(O, M_{POST})
Around	Bypass the single operation in a service entity and perform alternative operations of an aspect instead	Around(M_{PRE}, O, M_{POST})	(M_{PRE}, O) and (O, M_{POST})
Proceed	Perform the single operation in a service entity and the operations of an aspect concurrently	Proceed(M_{PRE}, O, M_{POST})	(M_{PRE}, O) and (O, M_{POST})
Flow	Perform the operations of an aspect along an execution path of a service entity concurrently	Flow(M_{PRE}, O, M_{POST}, O')	(M_{PRE}, O) and (O', M_{POST})

As shown in Figure 11, a crosscut point is specified using the names of messages and operations in the behavioral description of the service entity. Instead of a particular name of a message or an operation, we can use the symbol "*" that stands for *any* messages or *any* operations. If a crosscut relationship has the pointcut tag "{pointcut=before(*, OP1)}", all the input messages for the operation OP1 are interrupted. Only the before method or the after method can have "*" in both the operation and message name at the same time. For example, the pointcut tag "{pointcut=after(*, *)}" can be applied to a crosscut relationship from an audit aspect to log the output messages of all operations in a service entity or chain.

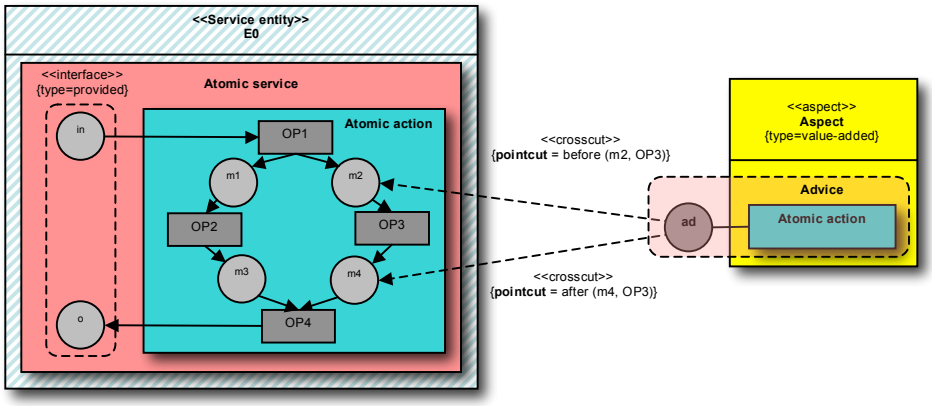


Figure 11: The graphical representation of the crosscut relationship in the tagged value "pointcut"

The following paragraph defines a crosscut relationship more formally in terms of atomic services in a service entity and an advice in an aspect.

Definition 11: A **crosscut relationship** $R_{\Theta(cp)}\,(E(S_E),\ C(S_C,))$ represents an explicit link between two Petri nets $S_E = (P_E, T_E, I_E, O_E, M_{0E}) \in E$ and $S_C = (P_C, T_C, I_C, O_C, M_{0C}) \in C$, which implies that an atomic service S_E in a service entity E is crosscut (intervened) by an advice S_C in an aspect C. A crosscut relationship has a crosscutting method Θ with either one

or two crosscut points shown in Table 1. A **crosscut point** *cp = (m, o) or (o, m)* defined by a message *m* and operation *o* in an atomic service S_E represents an input or output arc, called the **crosscut arc**, such that $(m, o) \in I_E$ or $(o, m) \in O_E$.

4.4 XML-Based Representation

The graphical representation based on the extended UML2 and the Petri net is not a suitable format to be stored and manipulated. Moreover, we do not want our design approach to stick to specific UML2 and Petri net tools. Their tool-specific representation format can undermine the interchangeability of the design results. We want our approach to be free from dependency of any particular tool that quite often dominates the whole design process. To provide a standardized and representation format corresponding to the graphical reorientation in the previous section, we define the service-oriented definition markup language (SODML), a set of service-oriented description languages based on the XML specification. Especially, SODML lets us implement the weaver program required for an automatic processing mechanism independently of tool-specific file formats The main purpose of defining SODML is to increase manageability and maintainability of the service-oriented design results in addition to their interchangeability or interoperability. For example, an automatic way to enforce designers to specify all the necessary information necessary for our design approach can be achieved by providing XML Schema Definition (XSD) that checks the validity of design results. Besides, we may use an XQuery [56] to find out repeated design elements that would be a candidate for a common aspect.

SODML has two basic markup languages, the service markup language (SvML) and the aspect markup language (AsML) over the Petri net markup language (PNML) explained in Section 2.3.4. Note that SODML uses the standard feature of the basic PNML to store Petri net for all the internal behavioral description for service entities and aspects. In addition

to those two markup languages, another markup language used to extend the design results for the analysis purpose is the part of the SODML. This extension markup language is discussed in the next chapter.

4.4.1 Service Markup Language (SvML)

Figure 12 shows the meta-model and the structure of the SvML. The names of the nodes in the meta-model are mapped to XML tags in the XML format.

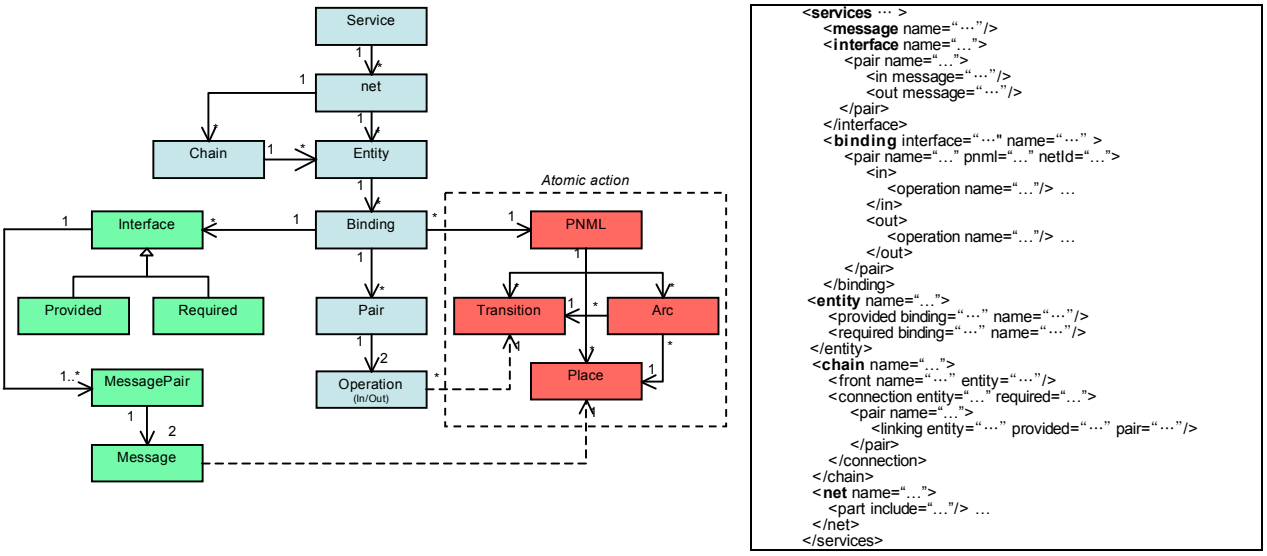


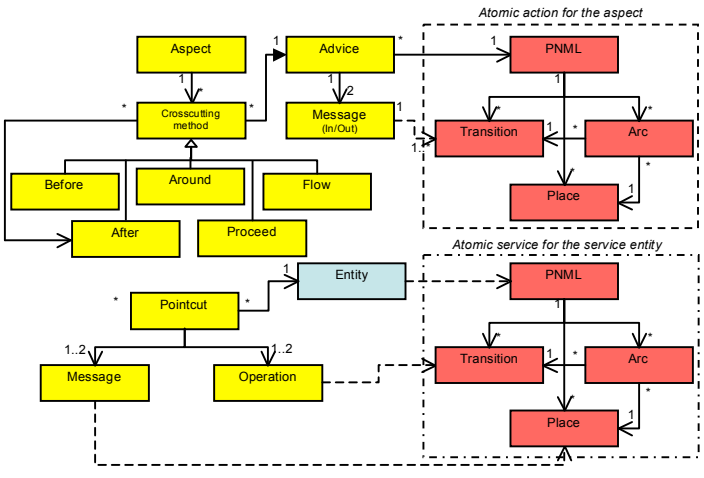
Figure 12: The meta-model and structure of the SvML

The *interface* tag is defined based on the message pair that consists of an input and an output message. The *binding* tag addresses how to build the Petri net for an atomic service by binding an interface definition in the interface tag with the Petri net in a PNML file designated by the *netId* and the *pnml* attribute in the *pair* tag under it. The interface definition does not need to be modified to replace the binding information with a PNML file appearing under the <binding> tag, which means that our approach allows service designers to change the behavior of a service without touching the interface definition published to customers.

The *entity* tag for a service entity contains necessary binding tags and decides whether their interface type is either provided or required. If a Petri net appears in multiple binding tags appertaining to a service entity, it is treated as a unique one within that service entity. Suppose that the binding tag *B1* and *B2* under the entity tag *E* indicates the same Petri net *N1* and they are defined as "provided" and "required" respectively. In this case, the service entity *E* has only a single Petri net *N1* with the provided interfaces *B1* and the required interface *B2*. The *chain* tag for a service chain states which message pair of the required interface for a service entity is linked to which one of the provided interfaces for another service entity. It also specifies which service entity offers a provided interface of a service chain. The *net* tag for a service net should contain at least a service entity or a service chain or a foreign service net.

4.4.2 Aspect Markup Language (AsML)

Figure 13 shows the meta-model and the structure of the AsML. In AsML, a crosscutting method is represented using one of five tags: *before*, *after*, *around*, *proceed* and *flow*.




```
<aspect …>
  <message name="…"/>
    <advice name="…" pnml="…" netId="…" sync="…">
      <in message="…">
        <operation name="…"/> …
      </in>
      <out message="…">
        <operation name="…"/> …
      </out>
    </advice>
    <before name="…" adviceBinding="…" crosscut="…">
      <pointcut message="…" operation="…"/> …
    </before>
    <after name="…" adviceBinding="…" crosscut="…">
      <pointcut message="…" operation="…"/> …
    </after>
    <around name="…" adviceBinding="…" crosscut="…">
      <pointcut preMessage="…" operation="…"
          postMessage="…" /> …
    </around>
    <proceed name="…" adviceBinding="…" crosscut="…">
      <pointcut preMessage="…" operation="…"
          postMessage="…" /> …
    </proceed>
    <flow name="…" adviceBinding="…" crosscut="…">
      <pointcut preMessage="…" preOperation="…"
          postMessage="…" preOperation="…"/> …
    </flow>
</aspect>
```

Figure 13: The meta-model and structure of the AsML

The *adviceBinding* attribute of a crosscutting method is used to associate an advice to a service entity indicated in its *crosscut* attribute. If the adviceBinding attribute value is not a name of the advice in the AsML file, we consider it as a file name for a non-implementable policy aspect such as a performance aspect. The crosscut attribute has the name of either a service chain or a service net in the sense that a crosscutting method is applied to all of the service entities within it. The *pointcut* tag under a crosscutting method denotes a crosscut point specified using the operation and message names or their IDs in the PNML file for a service entity.

The AsML uses the reserved word "any" instead of the symbol "*" to represent any message or operation. Care needs to be taken to specify a crosscut point of the Petri net that appears multiple times in a service element or the same crosscut point that occurs in different Petri nets in order to prevent from applying the crosscutting method to all matched crosscut points in every Petri net in the service element. An *advice* tag is used to designate a specific Petri net for the behavioral description and specify an advice interface that includes the input and output message. In addition, the advice tag may include the *sync* attribute that restricts the number of activated instantiations of the advice. The details about the sync attribute are explained in Section 4.5.3.

4.5 Weaving Process

The aspect-oriented approach applied in our design process supports an automatic weaving process to obtain an integrated behavioral description using the Petri net representation by weaving aspects into the service entity. The weaver pareses the SODML files for a service-oriented system and generate integrated Petri nets for its services. Each integrated Petri net corresponds to a service modeled. Figure 14 illustrates the weaving process for generating an integrated Petri net for a service in the PNML format.

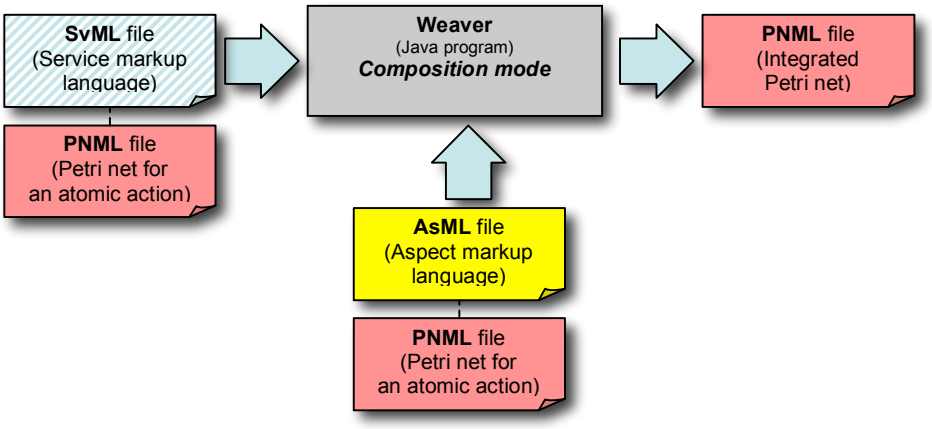
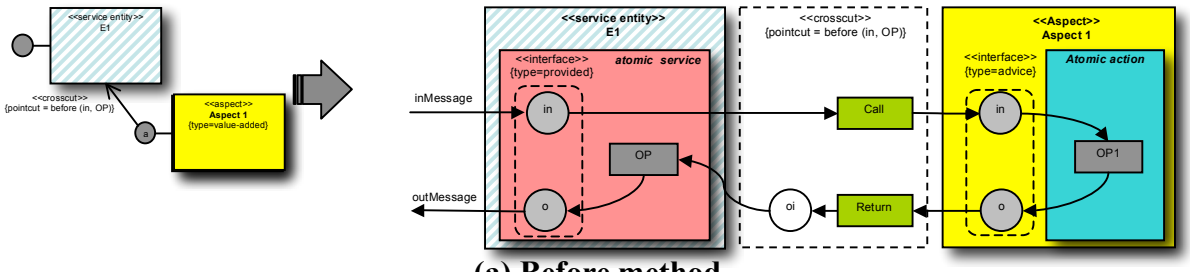


Figure 14: The weaving process for generating an intergrate Petri net for a service

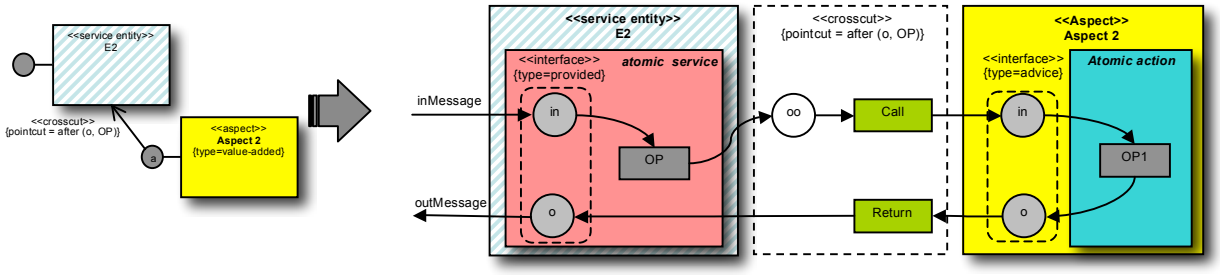
We support this weaving process by means of the Petri net-based semantics explained in the reminder of this section. In addition, we examine two cases of when several aspects with different crosscutting methods are applied to the same crosscut points in a service entity and when the number of instantiations of an aspect needs to be restricted.

4.5.1 Weaving

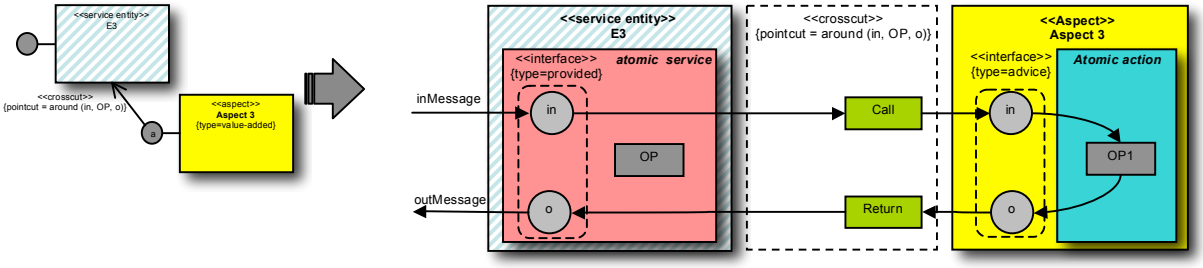
The weaving process produces an integrated Petri net using the Petri net for an atomic service in a service entity and an advice in an aspect based on the crosscutting methods specified in the crosscut relationships between them.



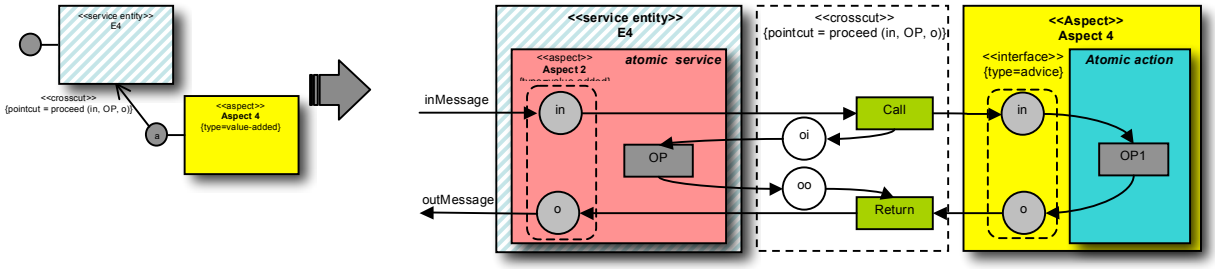
(a) Before method



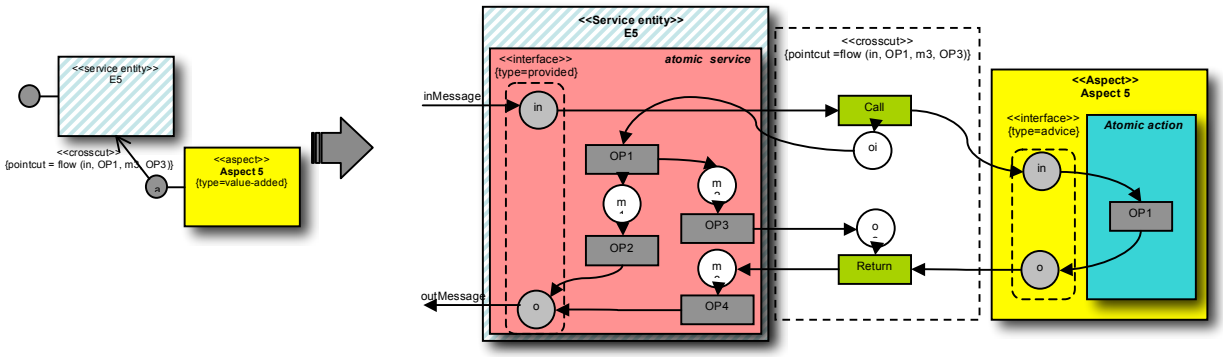
(b) After method



(c) Around method



(d) Proceed method



(e) Flow method

Figure 15: The Petri net semantics for five crosscutting methods

Figure 15 presents the Petri net-based semantics for the five crosscutting methods in the graphical representation. The right side of the figure represents the integrated Petri nets after the weaving process has been done. The first three methods directly modify the flow of the behavior of the primitive services whereas the proceed or flow method makes the advice of the aspect performed concurrently against one or more operations of the service entity. In terms of the Petri net-based semantics represented in Figure 15, the weaving process is formally described as follows.

Definition 12: Weaving process generates an atomic service by combining the atomic service in a service entity and the advice in an aspect according to their crosscut relationship between them. From Definition 11, the atomic service S_W obtained after weaving S_E and S_C is expressed using the weaving operator $\otimes$ such that $S_W = (P_W, T_W, I_W, O_W, M_{0W})$ $= S_E \underset{\theta(cp)}{\otimes} S_C = (A_E \oplus mp_E) \underset{\theta(cp)}{\otimes} (A_C \oplus mp_C) = A_W \oplus mp_E$. The $\otimes$ operator constructs an integrated Petri net for S_W by performing Algorithm 3 (the first weaving algorithm) in Appendix.

Definition 13: The weaving operator $\otimes$ is left-associative such that $S = S_1 \otimes S_2 \otimes S_3 = (S_1 \otimes S_2) \otimes S_3$ and $S_1 \otimes S_2 \neq S_2 \otimes S_1$.

The weaving process introduces *glue* operations (or *glue* transitions) to resolve the mismatch between the messages of a service entity and the input/output message of the advice interface of an aspect. There are two types of glue operations: *call* and *return* (denoted as t_{call} and t_{return} in Algorithm 2). The call operation converts an interrupted message in a service entity into the advice's input message while the return operation reformats the output message of the advice into an original message used in a service entity. At the implementation stage, these glue operations could be realized as glue code [36] that may

extract the specific value of the message, cast its type, or translate a XML document into another. These glue code may be implemented using BPELJ [59] if operations in a service entity (or service entities in a service chain) are declared as Web services and programmed in BPEL4WS.

4.5.2 Overlapped Crosscut Points

It happens quite often that several aspects are weaved into the same crosscut point. In Figure 16, three aspects have the common crosscut points.

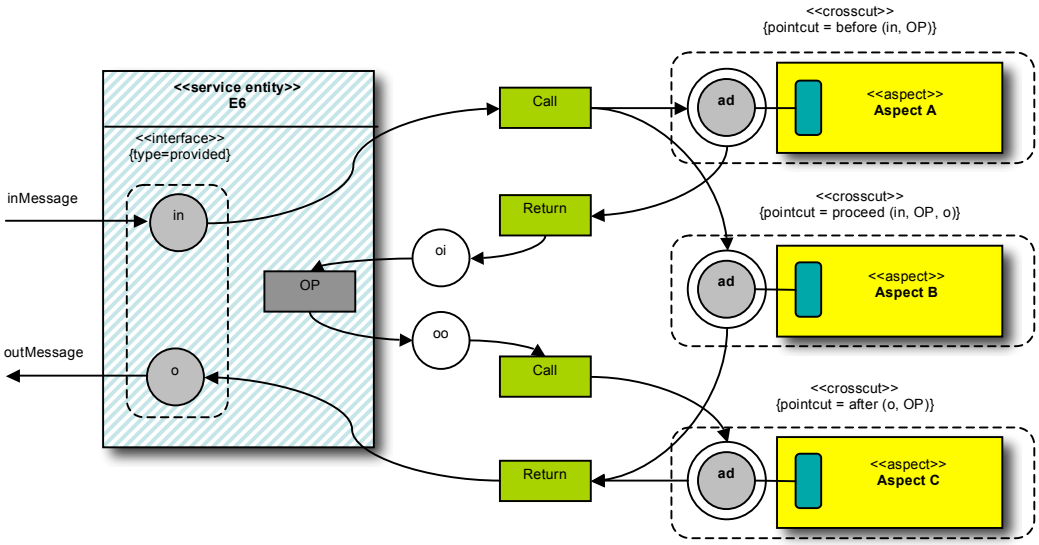


Figure 16: An overlapped crosscut point with three aspects

To deal with multiple aspects with overlapped crosscut points, the glue operations demand to do an additional task. The call operation works as a fork operation that executes attached aspects simultaneously while the return operation does as a join operation that assures their termination. In fact, the glue operations for the process and flow methods already play these role as shown in Figure 15-(d) and (e).

In addition to this role extension of the glue operations, we need a simple precedence rule when the around method is related to overlapped crosscut points. When an around method is applied to an operation, it suppresses the weaving of before, after and proceed methods attached to two crosscut points prior and posterior to the operation. The flow method is exceptional for this suppression because it is not attached to a single operation. For example, when an aspect is attached to the operation OP in Figure 16 using the around method, it invalidates the weaving of three aspects A, B and C.

4.5.3 Restricted Instantiation of Aspects

Basically, the weaving process uses a simple copy mechanism to knit an aspect into service entities. More specifically, the Petri net for an advice is copied into the one for an atomic service in a service entity as many as the number of the crosscut relationships between them. Each copied Petri net is regarded as an *instantiation* of the aspect. During a service execution or analysis, we often need to control the number of instantiations of an aspect. Suppose that an aspect A in Figure 17 has an advice to access a database in an exclusive way. Due to the copy mechanism applied, we can see that the corresponding Petri net appear twice into the Petri net of a service entity. However, these two Petri nets derived from the same aspect A should not be simultaneously executed. To limit the number of the concurrent invocations of an instantiated aspect, the tagged value “sync” can be specified in an aspect. Figure 17 is the weaving result of the aspect with the tagged value “{sync=1}”. Here, two instantiations derived from the aspect A cannot be executed at the same time. Either of them can be executed only when a token is available in the sync place. The sync tagged value is useful to represent a critical section in the behavior of a service entity. If the aspect A in Figure 17 is a *vacant* aspect that has a null operation, its “{sync=1}” prevents to operations, OP1 and OP3, for being executed simultaneously.

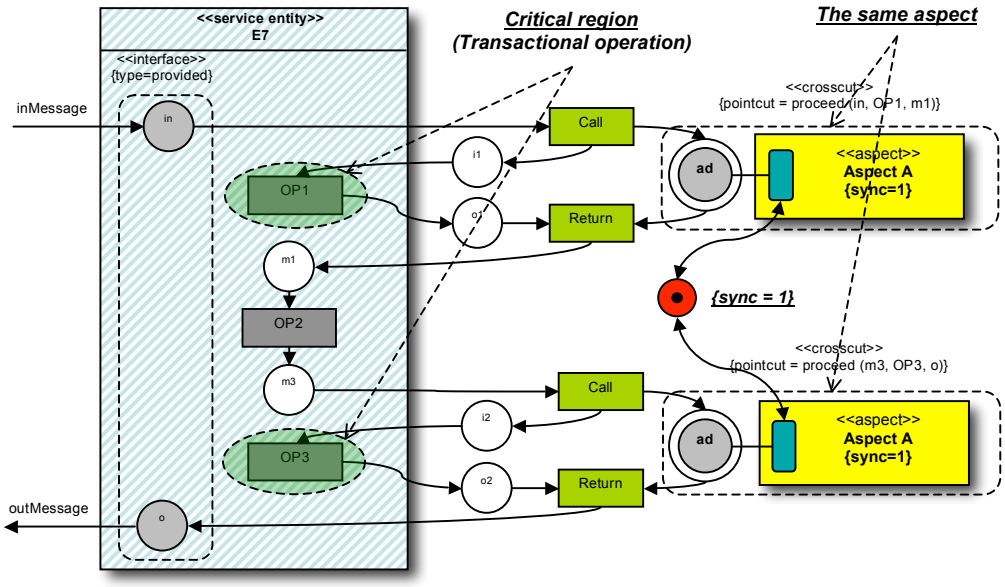


Figure 17: The aspect with the tagged value "sync"

During the weaving process, the weaver adds a special place called the *sync* place similar to a run-place, and connects it to the transitions bound with the input and output place for a message pair of an advice interface. The sync place has the initial tokens as many as the number is specified in the corresponding sync tagged value. The following paragraph defines the meaning of the sync.

Definition 14: A **sync** *sync(k)* for an advice restricts the number of activated instantiations of an advice during a service execution to a nonnegative integer *k*. A *sync(k)* applied to an aspect affects all advices. When an advice $S_C = (A_C \oplus \{p_i, p_o\})$ with *sync(k)* is weaved into S_E , Algorithm 4 in Appendix is performed as additional steps of Algorithm 3 in Appendix.

4.6 Example

We use the order service for a retailer service-oriented system as an exemplary service to demonstrate our design approach.

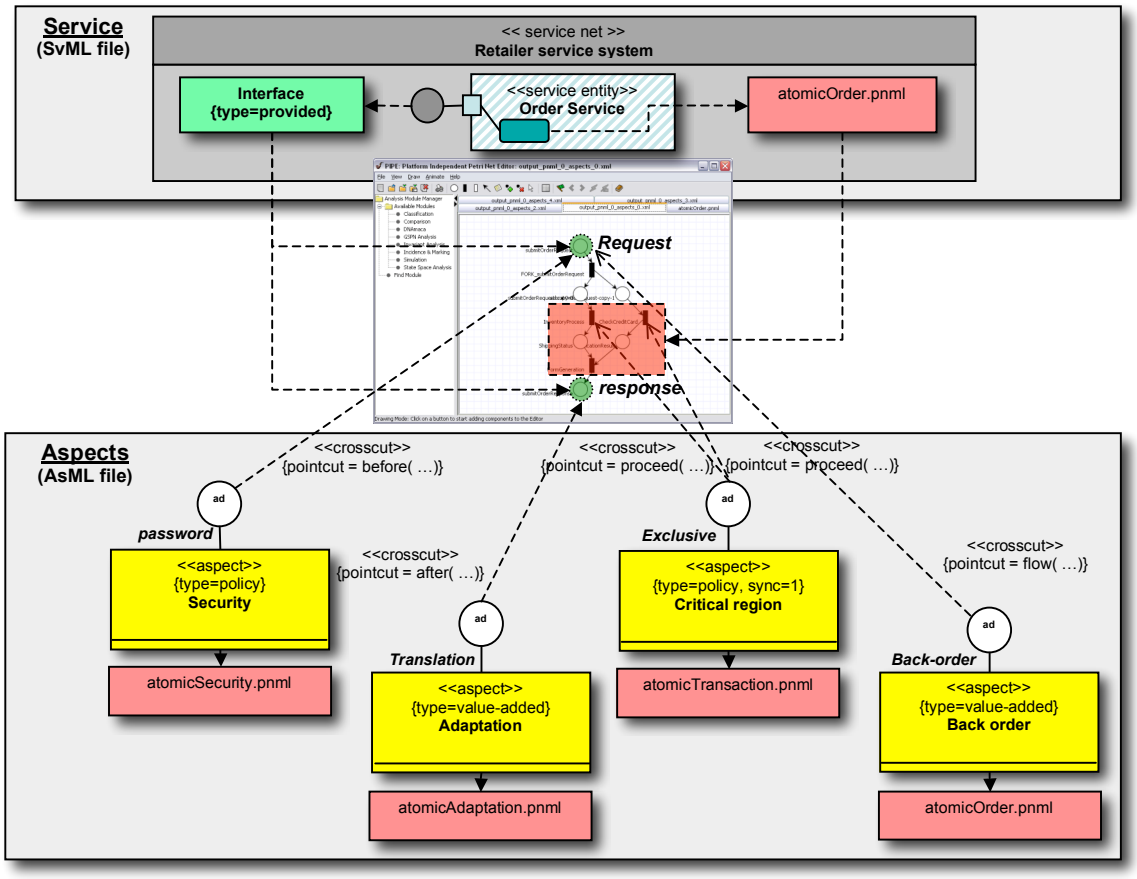


Figure 18: The order service in the retailer service net with four aspects

Figure 18 presents the graphical representation for a primitive order service and four aspects in the retailer service-oriented system. The upper box represents the primitive order service stored in two XML files: the SvML file for the structure of the retailer system including the order service and the PNML file for its behavior description. Upon customer's request, the primitive order service concurrently performs two operations for processing the

inventory-related data and verifying the credit card information, and then sends an order acknowledge back to the customer. These two operations are essential to build an order service and tend to remain unchanged even though the order service evolves.

This primitive order service is possible to evolve according to diverse reasons such as e-market trends or customer's needs. Suppose that a service provider wants to support the following four requirements related to the order service.

- To increase security, check password before the order service processes its request.
- To decrease the system workload, two operations the inventory process and the credit card verification, should not be performed concurrently.
- To send the order acknowledgement to customer's mobile device like a cellular phone, the response of the order service should be reformatted using another markup language such as Wireless Markup Language (WML).
- To place back orders into another retailer system if customer's order request includes items marked as "back-order" or "out-of-stock".

The four aspects in the lower box of Figure 18 are derived from the above four requirements. Each aspect is designed simply to have either one or two operations. The crosscut relationships from those aspects include the crosscut method in the pointcut tagged value of which the exact crosscut points are omitted for the sake of clarity. The aspects and the crosscut relationships are stored in an AsML file and the behavioral description for each aspect is stored in an individual PNML file. For the behavioral description of the back order aspect, the PNML file used for describing the behavior of the primitive order service is used again, which exemplifies the reusability of an atomic action as a basic design unit in the behavioral view.

4.6.1 Weaving

To obtain the integrated Petri net for the behavioral description of the order service with aspects, we implement a weaver program using Java version 1.5 [51] and JDOM [81]. The weaver produces an integrated Petri net for the service in the basic format of PNML by performing two steps: the binding step with the SvML file and the weaving step with the AsML files. Both steps also process the related PNML files. The task of each step is explained using the order service in Figure 18.

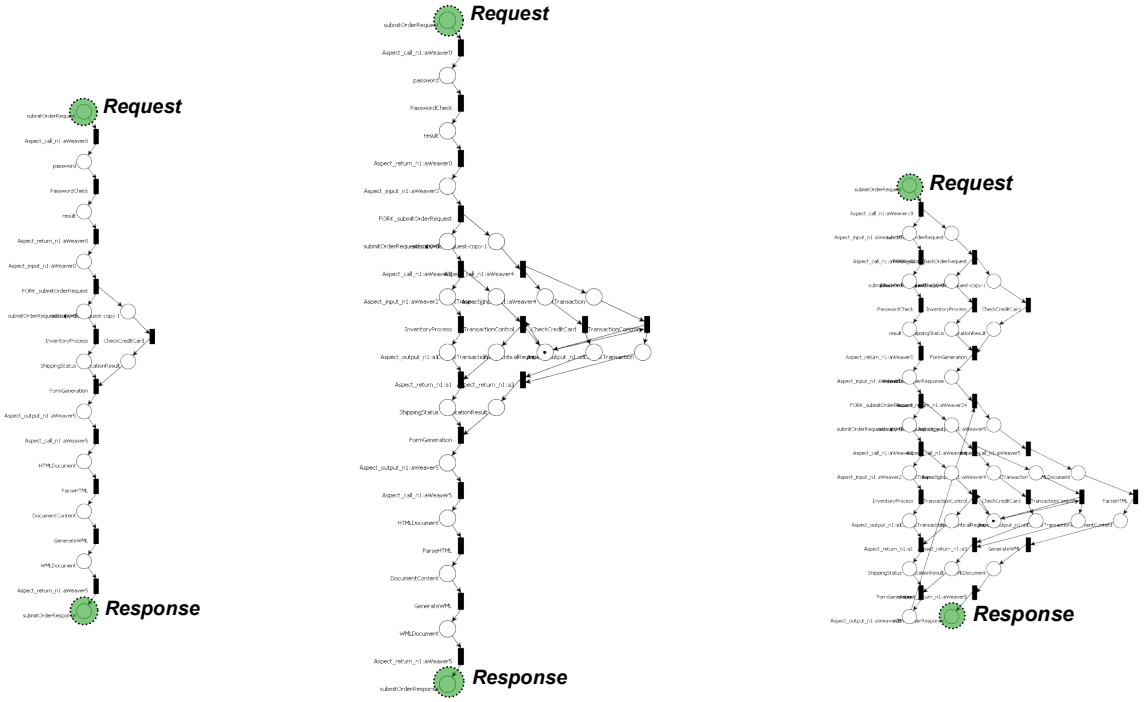
First, the weaver builds an atomic service for the primitive order service using the SvML file and the PNML file. The binding process for the primitive order service can be expressed as *PrimitiveOrderService = atomicOrder.pnml ⊕ (submitOrderRequest, submitOrderResponse)*. The weaver creates the request and response places based on the message pair defined. The input message represented as a request place is defined to be linked with two operations in the Petri net for the atomic service denoted in the dashed box. To prevent a non-deterministic choice between those two operations, the weaver program adds a fork operation that yields two cloned input messages for them. In Figure 18, the Petri net for the primitive order service after the binding step is shown using the PIPE tool [52]. Note that it can be seen by any Petri net tool that supports the PNML file format such as Petri net kernel (PNK) [53] and Renew [54].

Then, the weaver processes an AsML file with aspects and crosscut relationships and generates a corresponding integrated Petri net according to the Petri net-based semantics explained in Section 4.5. In this example, we create three versions of the order service as follows.

- *OrderService2 = PrimitiveOrderService* $\otimes_{before(...)}$ *Secutiry.Password* $\otimes_{after(...)}$ *Adaptation.Translation*
- *OrderService3 = PrimitiveOrderService* $\otimes_{before(...)}$ *Secutiry.Password* $\otimes_{after(...)}$ *Adaptation.Translation* $\otimes_{proceed(...)}$ *CriticalRegion.Exclusive*

- *OrderService4 = PrimitiveOrderService* $\otimes_{before(...)}$ *Secutiry.Password* $\otimes_{after(...)}$ *Adaptation.Translation* $\otimes_{proceed(...)}$ *CriticalRegion.Exlcusive* $\otimes_{flow(...)}$ *BackOrder.Back-order*

The weaving process generates the integrated Petri nets for the above three order services specified. Figure 19 shows them using the PIPE tool. Compared with the Petri net for the PrimitiveOrderService in Figure 18, those integrated Petri nets obtained are more complex owing to the Petri nets from aspects, glue transitions, additional intermediate places and arcs. The integrated Petri nets for the OrderService2 and the OrderService3 have an initial token in the place created based on the tagged value "sync=1" of the critical aspect.



(a) OrderService2 (b) OrderService3 (c) OrderService4

Figure 19: The integrated Petri nets for the different versions of the order service

As mentioned, the back order aspect in Figure 19-(c) uses the same atomic action with the primitive order service. If a back order process needs to be different from the order service entity, a new integrated Petri net for the changed order service can be obtained simply by performing a weaving process with a new back order aspect modified with a changed atomic action.

4.6.2 Analysis

With the integrated Petri nets, we can perform their structural analysis that helps verify the correctness of the service design results. The structural analysis of the integrated Petri nets for the order services in Figure 19 is done with the analysis modules of the PIPE tool. Note that some of its analysis modules require an initial token to be placed in the request place, which represents a service input message from a customer. The classification module identifies the subclasses of the integrated Petri net. The integrated Petri nets always raise a deadlock, at least, at the response place during the analysis process. Therefore, the detection of a real deadlock with a dangling operation that does not contribute the processing of the order services requires a preprocess that makes the integrated Petri nets strongly connected by inserting a transient transition and linking it with the request and response place. The state space analysis results report that the integrated Petri nets are *bounded*, *safe*, and *deadlock free*. In particular, the boundness is reassured by executing the invariant analysis module that performs transition and place invariants (T- and P-invariants). The simulations on the PIPE tool result in the average numbers of tokens for all the places of the integrated Petri nets, which may help to find a bottleneck position in terms of structural view of the order service.

Although the behavior analysis can be performed with these integrated Petri nets, its results do not provide any significant meaning. This is because they have a simple flow of

which behavior can be predictable based on their structural characteristics. More comprehensive and meaningful behavioral analysis could be performed with the integrated Petri net once some extra information is applied. For example, we can do the performance analysis if the integrate Petri net has performance-related values such as temporal values for each operations. Therefore, an automatic mechanism will be discussed in the next chapter to extend the Petri net for a service with extra information related to the behavior of that service.

4.7 Related Work

There are quite a lot of research works to help design service-oriented systems. Even though the definitions or roles of their service seem to vary according to the viewer's perspectives [27], they recognize the service as an abstract design unit and treat it as a significant element for modularization through a service-oriented software development. In addition, the service-oriented design method is believed to support a coarser granularity of abstraction, a higher level of visibility and a wider range of focus on service-oriented systems than other current design practices such as object-oriented or component-oriented design approaches [83].

Aspect-oriented approach is utilized quite often to cover various concerns raised during service development process. For example, the component-based software development (CBSD) could be regarded as a service-oriented software development with a limited concept of the service. Here, a specific service is destined to be performed through a component as a functional black box entity. Since the component concept is insufficient to encapsulate a wide range of scope for the service concept, several CBSD methods incorporate with aspect-oriented approach for developing service-oriented systems [16][46].

In a service-based model proposed in [38], services are incrementally elaborated and mapped to components under consideration to cover the safety and security aspects. This

approach requires a formally defined model for service architecture. They report that the service contexts for context-adaptive services need to be incorporated into service architecture models as future work. In [39], a service itself is defined as a crosscutting architectural aspect. Using an Architecture Definition Language (ADL), services are captured as interaction patterns among the roles of the system. During the weaving process, aspects are used to assign roles to the component classes. The main purpose of services is to separate roles from component configurations and architectures. In our opinion, other issues such as customer-friendliness deserve attention from the very beginning stage of services design.

A service-oriented architecture (SOA) modeling based on the architecture style is presented in [37]. A static and a dynamic part of the architecture style are depicted using class diagrams and graph transformation system rules, respectively. A model checking technique is used to validate consistency of scenarios captured in sequence diagrams with respect to the dynamic behavior for the architectural style. This architecture-level design is useful for business scenario analysis following the *Publish-find-interact* (or *find-bind-execute*) paradigm rather than a generic service-oriented system from service-oriented designers' viewpoint.

Although web services [33] are a specific kind of services, most service-oriented computing researches are more or less related to them. They provide basic and standard technology for service publishing and customer interaction using XML-based languages, SOAP [77], UDDI [78] and WSDL [79] in terms of the Publish-find-interact paradigm. The business process behavior is expressed using BPEL4WS [34]. Although currently technology provides the infrastructure for implementing web service-oriented systems, they are not sufficient to deliver a qualified service. For instance, WSDL that is similar with our service entity concept does not include any quality information of a service. The web service offering language (WSOL) [40] is proposed to formally specify various constraints and multiple classes of service provided by web services in addition to WSDL. Besides, many

service-specific features such as context-awareness or data translation are still an important factor, but are scattered across multiple modules in the design phases. For BPEL4WS, a pattern-based Petri net semantic is developed to translate every BPEL process into a Petri net [49]. This exemplifies that the Petri net is a useful method to express a business logic in a composite service.

In [44], aspect-oriented component engineering (AOCE) is extended to support the development of web services technology. The proposed component aspects such as user interface, distribution, transaction processing, security, persistency, configuration, etc. are separated from several modules. After the details are analyzed from these aspects, they are annotated to WSDL or UDDI. Although these annotated WSDL and UDDL, named AO-WSDL and AO-UDDI, cannot be processed without a common argument among web services systems, they illustrate that the aspect-oriented approach can help the development of web service-oriented systems. Aspect-oriented approach is applied to extend BPEL4WS too. AO4BPEL [45] is proposed to cover two shortcomings of BPEL4WS: the lack of modularity and the static composition. The aspect as a stand-alone business logic or service in the XML format can be plugged or unplugged into the composition process at runtime. To support AO4BPEL, an aspect-aware BPEL orchestration engine is devised. The BPEL runtime, one of its five subsystems, manages most of process-related jobs including invoking and joining aspects via the aspect manger. These works show the application and usefulness of aspect-oriented approach to extend web services technology, rather than a means to support aspect-oriented service design itself.

It is also important to compose existing services in an instantaneous and efficient way. Some researches propose dynamic service composition frameworks in order to give flexibility and adaptability to service-oriented systems. They build a composite service by connecting simple and peer services. In our approach, it can be designed either using a service chain or using functional aspects treated as individual services. Although, our work

does not have a framework to support just-in-time adaptive service composition or aspect weaving, it is worthy of surveying them in terms of how to construct a composite service from basic services. In eFlow [41], a service broker finds necessary services based on the predefined selection rules and constructs a composite service using a simple graph similar to the UML activity diagram. When a service composition is modified, eFlow enforces transactional semantics based on the predefined rules by verifying behavior consistency to confirm that the migration does not result in run-time errors or non-deterministic behaviors. Self-serv [42] uses the state chart to express the business logic of a composite service and invoke component service operations in a procedural sequence. A service coordinate for a composite service uses two routing tables for preconditions and postprocessing actions in order to make an execution path from state chart to state chart. Based on Self-serv, the quality model for web services is proposed in [43]. They suggest five generic quality criteria for elementary services: execution price, duration, reliability, availability, and reputation. Using them, the five corresponding aggregations for the QoS of composite service functions are defined. Then, an execution plan for a composite service is selected by solving an optimization problem using a linear programming method. JAsCo [46] is also a tailored aspect-oriented language based on the Java Beans component model for component-based development. Aspect beans are defined as a regular Java beans with hooks. The connector as the collection of related aspects initializes the hooks of various aspect beans for a specific context and executes their behaviors. It shows how to implement aspects in component-based middleware, but does not explicitly present how to design and analyze them. Later, JAsCo is applied to implement the Web services Management Layer (WSML) [47] that is placed between the application and the world of web services in order to provide a just-in-time integration based on dynamic AOP like eFlow and Self-serv. Just like other middleware, WSML supports selecting and monitoring modules as well as generic management functionalities.

Our model is roughly comparable to four metalevels of the Meta Object Facility (MOF) of the model driven architecture (MDA) [48] as a platform impendent model (PIM). For example, the UML2 composite structure and associations extended with stereotypes and tagged values in SODA could be the constructs of the level M3. With these constructs, two metamodels for services and aspects shown in Figure 12 and 13 are created as the metamodels for the level M2. Although our approach makes service design results reusable across multiple implementations by transforming them into other models owing to the XML-based representation, it omits many important factors in the MDA such as the platform specific model (PSM), the transformation between models or from a model to a source code, and automatic code generator from a more abstract model.

Similar approaches were proposed in [63][64] for modeling secure software based on PrT (Predicate/Transition) nets. These domain-specific approaches are more appropriate to patch developed software since they want to identify formulae mapped to transitions, which hinders developers to sketch diverse software systems at the design stage. Furthermore, their approaches could not be appropriate for evaluating system performance due to their use of PrT nets. To apply their approaches, we could support an extension mechanism that converts the basic Petri net used in SODA to PrT net by means of devising the syntax for those inscription formula to be map to transitions

4.8 Summary and Discussion

In this chapter, we present the generic service-oriented design approach with the concept of aspect named SODA. In SODA, a service belonging to a service-oriented system is considered to be decomposed into the primitive function with an interface that can be published and discovered and the aspects that denote supplementary features or annotate quality attributes.

The conceptual structure of a service-oriented system is drawn in the extended UML2 and represented using service elements, aspe cts, and their relationships. Among service elements, a service entity is the fundamental structural unit that includes the Petri net for describing its behavior. A composite service and a service-oriented system are drawn using a service chain and a service net respectively. Aspects can be categorized under the three generic high-level aspects: *common*, *policy* and *value-added*. The *common* aspect is used to denote structural and functional redundancy in a service-oriented system. The *policy* aspect annotates quality attributes across a service-oriented system and provides behavioral guidelines for services. The *value-added* aspect is used to group extra functions for customizing services. Domain-specific or service-dependent aspects can be defined if needed. Each aspect is associated with multiple service elements using crosscut relationships.

To increase the interchangeability and interoperability of the service-oriented design result, we define the service-oriented definition markup language (SODML) to store it in the XML format. Service elements are stored using the service markup languages (SvML). Aspects and their crosscut relationships with the service elements are specified in the aspect markup languages (AsML). The Petri nets for their internal behavioral descriptions for the functionality of the service or the aspect are stored using the Petri net markup language (PNML). According to the Petri net-based semantics defined for the crosscutting methods in the crosscut relationship, the weaver generates an integrated Petri net from the Petri nets of a primitive service and aspects. As a result, by altering or updating aspects attached to a primitive service, we can generate different versions or variations of a service. The integrated Petri nets obtained through weaving process facilitate the analyses of the service design results in both the structural perspective and the behavioral perspective.

Our aspect-oriented and Petri net-based approach is greatly expected to help increase manageability, reusability and maintainability of service design results. Most of all, the main advantage of our work is that the modification of a service design or the creation of its

variation is allowed to be effectively and efficiently done with low or reduced effort by changing or reusing any design elements, especially aspects. We believe that our design approach helps a service-oriented software system development process and contributes to the service-oriented computing community as well as the aspect-oriented software development community.

CHAPTER 5. EXTENSION OF SERVICE-ORIENTED DESINGS

5.1 Introduction

Services described in Petri nets have many advantages owing to the mathematical soundness and graphical representations of Petri nets, and availability of various Petri net tools. However, the basic Petri net used in our design process is absent from some information necessary to identify the behavioral characteristics of its corresponding service. For example, simulation of a Petri net for a service could find a structural problem such as a deadlock path, but it cannot tell about the elapsed time for the service delivery or point out where a performance bottleneck occurs. To discover this kind of behavioral characteristics of a service, we need to consider the information specific to the service behavior during the design analysis process. It is also very useful to evaluate a service or compare it with others. In terms of aspect-oriented approach, such behavior-specific information can be regarded as particular concerns that are only necessary during the behavior analysis process of the service design result. Therefore, it can be separated from the design result and applied only when necessary. For this purpose, an aspect-oriented extension mechanism is proposed for the Petri net-represented service[1]. Note that the Petri net is used to describe the behavior of a service-oriented design result. We provide a systematic method to extend a service design result with its behavior-specific information. Also, this extension process needs to be independently and automatically performed only in case of necessity to analyze the behavioral characteristics of the service design result. There are two types of the behavior-specific information considered in this chapter and applied for the extension at the service design level. One is the design-specific data like temporal values required for executing the operations and the other is the

[1] This extension mechanism can be applied to not only a Petri net-represented service, but also any design result that can be represented using the basic Petri net and stored in a PNML file.

resource-related data caused by an underlying platform. Accordingly, our extension mechanism supports two different phases: the one for appending the design-specific data to the Petri net for a service and the other for transforming a platform-independent service into an platform-dependent service that takes into account the resources and their interferences in terms of the Petri net-based semantics. Each extension stage can be performed individually.

The behavior-specific information separated from service designs is described using XML-based representation languages that make this extension mechanism aligned with our service-oriented design approach (SODA) presented in the previous chapter. Through the weaving process, the basic Petri net for a service is transformed to an augmented Petri net used for analyzing behavioral characteristics of the service extended with design-specific information and resource interferences. This aspect-oriented extension mechanism facilitates evaluating a service designed, or selecting an optimal or superior one among several service design candidates.

5.2 Overview of Aspect-oriented Extension Mechanism

In this chapter, a service stands for an atomic service in a primitive service after binding process or a service obtained via the weaving process of aspects for the convenience of explanation. In other words, a service denotes an autonomous service represented using a single Petri net with a request and a response place. Thus, a token in the request and response place of the Petri net for a service implies a service invocation and termination respectively. Also, the transitions and the places of the Petri net are interpreted as the operations and messages constituting the corresponding service. We want to prevent a different interpretation of the transitions and the places that can vary according to the granularity of the design in the service development process.

As mentioned, our aspect-oriented extension mechanism consists of two independent extension phases: one with the design-specific data and the other with resource-related data.

The major difference between two phases is that whether an original structure of the Petri net for a service needs to be modified or not. The design-specific data can be appended to the Petri net for a service without causing its modification. The resource-related data, however, requires it to be transformed because resources and their interactions need to be materialized within the Petri net for a service.

The design-specific data is appended to the elements of the Petri net for a service. To group and specify those data in the XML format, the PNML extension markup language (PeML) is devised. Usually, the design-specific data contains extra information that transform a basic Petri net used to represent the service design result to another class of Petri net such as a generalized stochastic Petri net (GSPN) without touching the original structure of the basic Petri net. Although a service seems to be sufficiently outlined with the basic Petri net, its behavioral analysis tends to require some extra information. For example, the performance analysis can be performed when a rate value is assigned to each transition. Actually, the separation of this design-specific data is strongly recommended due to following reasons. Firstly, the design-specific data is not needed until a behavioral analysis is done. It can be managed separately and weaved automatically as the occasion demand. Secondly, The separated design-specific data can be modified without causing any side effect on the developed model. Through weaving process, a modified data leads a new Petri nets for the service of which behavior characteristics would differ. Finally, the representation format for those data also can be isolated. Most of currently available Petri net tools that support the PNML file tend to represent the design-specific data using their own tags that severely decrease the interchangeability of the PNML file. Once all of the data and its relevant tag are removed from a PNML file, it would not happen that another Petri net tool fail to open that PNML file because it cannot understand those PIPE tool-specific tags.

The resource-related data requires a specification about what kind of resources is available and how they interact with the service developed. Since this data is intrinsically

unrelated to the design results, its separation is natural. To provide a standardized specification format for this resource-related data, the resource extension markup language (ReML) is defined and used. In ReML, a resource-related data consists of resource definitions and their weaving rules. A resource is defined as one of three abstract resources that will be explained in the later section. Whenever a defined resource interacts with (crosscuts) a Petri net for a service, it is treated and instantiated as an aspect. The interaction of the defined resource is specified according to the weaving rules based on the Petri net-based semantics. Since the resource-related data specified in ReML can be placed under the PeML, we can conveniently group and store behavior-specific information mentioned in this chapter within a single file. The meta-model for the PeML including ReML is shown in Figure 20 and its details will be discussed in the following sections.

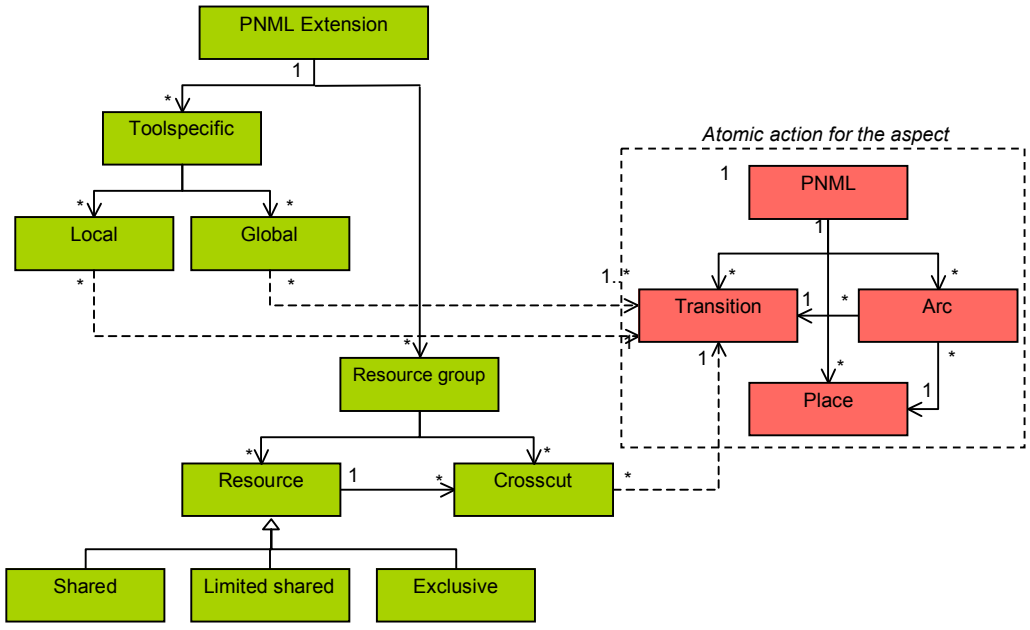


Figure 20: The meta-model of PeML including ReML

Through the weaving process shown in Figure 21, the weaver in the extension mode reads a PNML file for a service and a PeML file for design-specific data and resource-related data, and then generates a PNML file for an augmented Petri net that represents a service extended with design-specific data or various resource interferences.

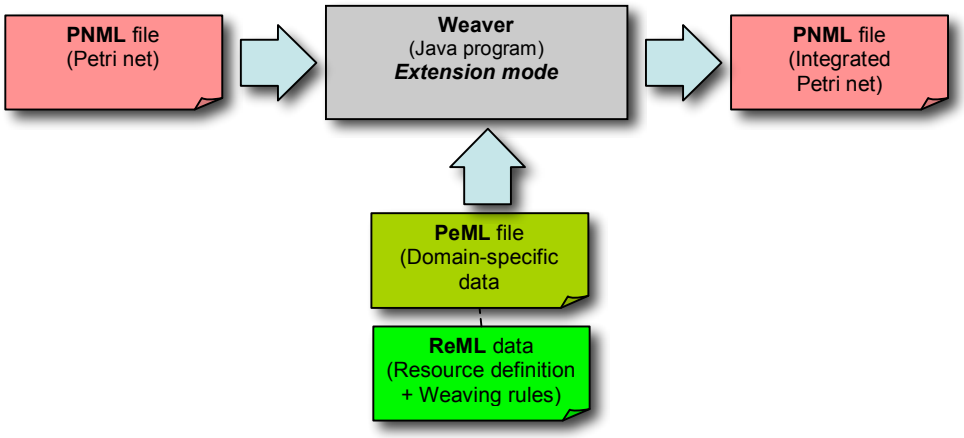


Figure 21: The weaving process for extending a Petri net for a service

If a Petri net tool supports PNML and understands tags for appending design-specific data, an augmented Petri net generated via weaving process could be immediately used to analyze its performance characteristics. Otherwise, a transformation procedure of the PNML file is necessary, for example, using the XML translation language (XSLT) [50], or a translation program implemented.

Although this chapter focuses on the explanation of the extension mechanism in terms of the behavior of a service, we want to mention that the behavior-specific information in PeML or ReML used in the extension mechanism can be specified in the structural representation of the service without causing any side effect. The design-specific data in ReML can be simply specified using a UML note that is depicted as a rectangle with the top-right corner folded over. The resource-related data in PeML can drawn just like the graphical

representation of aspects of SODA so that each resource is represented using an aspect and its interferences are expressed using crosscut relationships of which methods are restricted according to the type of the resource.

5.3 Extension Mechanism for Design-specific Data

According to the PNML specification, the design-specific data should be placed under the *toolspecific* tag. If the design-specific data is temporal values, its representation is recommended to follow a syntax proposed for a generic Time Petri net based on the Mathematical Markup language (MathML) [60]. However, most of Petri net tools that support the PNML are representing those data using their own special tags. For example, the PIPE tool uses the *rate* and *value* tag for specifying a stochastic value for each transition in a PNML file in addition to the basic Petri net representation. As mentioned, those tool-dependent tags decrease the interchangeability of the PNML file. The PeML is designed to group this design-specific information enclosed with tool-dependent tags in a single file in terms of an identifier – a unique name and a version. Through the weaving process, a group of design-specific information can be selectively attached to a basic Petri net for a service. Note that our extension mechanism is generic. Any kind of data described in XML can be applicable to extend a Petri net although we only demonstrate the extension with a delay data in this section.

Based on the Petri net definition in Definition 1, the extension with design specific data can be considered to associate a set of the elements of the Petri net for a service to a set of design-specific data. The extension with the design-specific data can be formally described as follows.

Definition 15: If a Petri net for an atomic service is extended with the design-specific data $D = \{d_1, d_2, d_3, ...\}$, the weaving process for this extension is to add an additional tuple

for a mapping function $X \rightarrow D$ to the definition of the Petri net $P = (P, T, I, O, M_0)$, where $X \subseteq P \cup T \cup I \cup O$.

For example, a mapping function $E_0: T \rightarrow \{d_1, d_2, ..., d_i, ...\}$, where d_i is a delay such as a rate value, transforms a basic Petri net for a service to a timed transition Petri net.

5.3.1 Petri Net Extension Markup Language (PeML)

PeML places a group of a design-specific data under the *toolspecific* tag of which tool and name attributes are used to indicate its unique identifier. For example, the PeML file in Figure 22 has two toolspecific tags for the PIPE tool version 1.5b [52] and the StpnPlay tool version 0.85b [55].

```
<PNMLextension>
    <toolspecific tool="PIPE" version="1.5b" netId="n1">
        <global type="transition" toolSpecificTag="false">
            <value tag="rate">
            <value tag="value">2.0 </value>
            </value>
            <value tag="timed">
                <value tag="value">false</value>
            </value>
        </global>
        <local name="InventoryProcess" toolSpecificTag="false">
            <value tag="rate">
                <value tag="value">0.5 </value>
            </value>
            <value tag="timed">
                <value tag="value">true</value>
            </value>
        </local>
    </toolspecific>

    <toolspecific tool="stpnplay" version="0.85b" netId="n1">
        <global type="transition" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Deterministic</value>
            </value>
            <value tag="value">
                <value tag="value">3.0</value>
            </value>
        </global>
        <local name="CheckCreditCard" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Exponential</value>
            </value>
            <value tag="value">
                <value tag="value">20.0</value>
            </value>
        </local>
```

```
        <local name="PasswordCheck" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Exponential</value>
            </value>
            <value tag="value">
                <value tag="value">10.0</value>
            </value>
        </local>
        <local name="TransactionControl" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Exponential</value>
            </value>
            <value tag="value">
                <value tag="value">25.0</value>
            </value>
        </local>
        <local name="GenerateWML" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Exponential</value>
            </value>
            <value tag="value">
                <value tag="value">10.0</value>
            </value>
        </local>
        <!-- Extra communication delay for the Back Order
        <local name="FORK_SubmitBackOrderRequest" toolSpecificTag="true">
            <value tag="type">
                <value tag="value">Exponential</value>
            </value>
            <value tag="value">
                <value tag="value">30.0</value>
            </value>
        </local>
        -->
    </toolspecific>
</PNMLextension>
```

Figure 22: A PeML File

The *global* or *local* tag has a data described using the nested *value* tags. The global tag specified with a node type appends a data to the transitions, the places, or the arcs of the

same type while the local tag with a name of a node does it to the particular node. The weaver is designed to process the global tags first. Thus, it is possible that the local tag processing overwrites tag values created by the global tag processing.

5.3.2 Extension Process

The extension process is the weaving process of a set of data enclosed with the toolspecific tag into a Petri net for a service. Although a PeML file can contain a number of toolspecific tags, the weaver can work with one of them. Suppose that the weaver is invoked with the PeML file in Figure 22 and the options, "tool=PIPE" and version="1.5b". Since the processing of global tags precedes that of the local tags first, the weaver puts the set of tags "<rate><value> 2.0 </value></rate>" and "<timed><value>false</value></timed>" under all the transitions of the Petri net *n1* in the input PNML file. The value of the *tag* attribute of the *value* tag is used as a new tag name created during the extension process. If the *toolspecificTag* attribute of the global tag is set to true, the corresponding generated tags are enclosed by the toolspecific tag with attributes for the tool name and version. Then, the weaver searches the "InventoryProcess" tag in the Petri net *n1* and tries to put "<rate><value> 0.5 </value></rate>" under it. Since the weaver program already creates it during the global tag processing, it just changes the content of the <value> tags from 2.0 and false to 0.5 and true respectively. The result of this weaving process transforms the basic Petri net to a stochastic Petri net that can be analyzed in the PIPE tool.

Once the weaver is invoked with the same PeML file and the options, "tool=stpnplay" and version="0.85b", it produces the PNML file for an augmented Petri net with the hypothetical data required to use the StpnPlay tool version 0.85b. Unfortunately, the StpnPlay tool for modeling and simulating stochastic Petri nets saves a Petri net in a simple text file. Actually, there are many Petri net tools that stick to their own file format or its own XML file format to store a Petri net and the design-specific data related to it. To exploit

existing Petri net tools, a PNML file obtained after the extension process needs to be transformed into another format file required by a Petri net tool. For this case, we decide to use a simple conversion program implemented in Java in order to the transformation of a PNML file to a text file for the StpnPlay tool. Another option for this transforming process is to use the XML translation language (XSLT) [50] that facilitates the translations of a PNML file into another format file such as text, HTML and XML including PNML by writing a XSLT document for a Petri net tool. For example, an application of XSLT using the PNML format is shown in [60] to convert a class of Time Petri nets into another class of Petri nets. Note that a similar way could be applicable to translate SvML and AsML files for a service-oriented system into an UML-tool accessible file, and vice versa.

5.3.3 Example

Based on three integrated Petri nets for the order service shown in Figure 19 and a PeML file in Figure 22, we create five PNML files for augmented Petri nets with the design-specific data for the StpnPlay tool: two from OrderServer2, one from OrderServer3 and two from OrderServer4. One of the augmented Petri nets extended from OrderServer2 is obtained by applying only the global tag section in the PeML file. One of the augmented Petri nets extended from OrderServer4 has an extra communication delay for the back order aspect in the commented local tag section in the PeML file. These PNML files are transformed to the StpnPlay text file format by the Java program implemented. As an initial setting, we place 1000 initial tokens in the request place and run simulations to measure the time to complete all 1000 service requests. The more tokens in the request places, the more service requests for the order services.

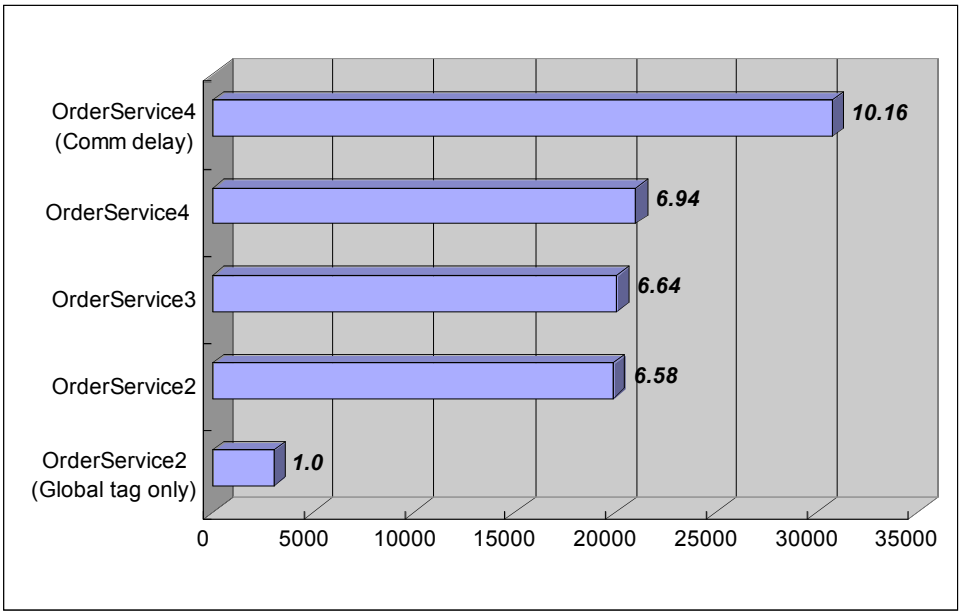


Figure 23: The simulation results of integrated order services extended with the PeML file

The chart in Figure 23 results from the simulation of the five augmented Petri nets on the StpnPlay tool. The X-axis represents a time unit. The service completion time of OrderService2, OrderService3 and OrderService4 are more than six times as long as that of OrderService2 with global tag only. OrderService4 with extra communication delay requires ten times of the service completion time of OrderService2 with the global tag only. This is because we assume that the back order aspect needs a long communication time. If temporal or stochastic data for the operations in the order service are obtained historically or experimentally, and applied in the simulation, we could measure or predict its performance more precisely and more accurately.

The simulation result demonstrates that our design and extension approach can support comparative evaluation of the variations of a service from the perspective of performance. Here, the extension mechanism plays a vital role to make it possible to measure the performance of services, accurately predict the performance change of modified or

updated services, and comparatively evaluate the performance of different versions or variations of a service.

5.4 Extension Mechanism for Resource-related Data

In many cases, the performance analyses of the Petri net for a service need to consider the resource limitations caused by a specific platform. For this purpose, the extension with the resource-related data modifies a basic Petri net in order to reflect the resource interferences. The resource extension markup language (ReML) is designed to describe one or more resource-related data by defining the resources in three types in accordance with its accessibility and availability, and specifying their interactions with a service in terms of weaving rules. The weaving process with a resource-related data generates augmented Petri nets for a service extended to reflect interactions of the defined resources. These Petri nets can be used to determine an optimal resource necessity to complete a service modeled. Similarly, a particular resource-related data can be applied to several Petri net-represented services in order to find a superior one under the circumstance of resource interference cased by the underlying platform. Besides, a resource-related data, especially its resource definitions, can be easily reused to describe another one.

We start with the assumption that any operation in a service cannot demand a resource in the middle of its execution. In other words, all the transitions in the Petri net for a service are considered as atomic operations. Hence, the operation that requires a resource during the execution needs to be decomposed into multiple operations based on its resource requests.

5.4.1 Resource Extension Markup Language (ReML)

In ReML, a resource-related data is enclosed with the *resourcegroup* tag of which *tool* and *version* attributes is used to distinguish it from others. Thus, a ReML file can contain multiple sets of resource-related data. Just like the weaving process with a PeML file, we can

choose a resource-related data in a ReML file for the weaving process by passing the name and version of a Petri net tool to the resource weaver. A resource-related data is divided into two sets: a set of resource definitions and a set of the weaving rules

```
<?xml version="1.0" encoding="UTF-8"?>
<PNMLextension>
   <resourcegroup tool="stpnplay" version="0.85b" netId="n1" toolSpecificTag="true">
      <!-- RESOURCE DEFINITIONS -->
      <resource name="network" accessMode="shared">
         <value tag="type">
            <value tag="value">exponential</value>
         </value>
         <value tag="value">
            <value tag="value">20.0</value>
                  </value>
      </resource>

      <resource name="memory" accessMode="shared" limit="1">
         <value tag="type">
            <value tag="value">exponential</value>
         </value>
         <value tag="value">
            <value tag="value">10.0</value>
         </value>
      </resource>

      <resource name="processor" accessMode="exclusive" limit="1">
         <value tag="type">
            <value tag="value">exponential</value>
         </value>
         <value tag="value">
            <value tag="value">5.0</value>
         </value>
      </resource>

      <!-- WEAVING RULES -->
     <crosscut resourceBinding="network" operation="Operation0" position=before"/>
      <crosscut resourceBinding="memory" operation="Operation0" position="after"/>
     <crosscut resourceBinding="processor" operation="Operation0" position="Operation0"/>
   </resourcegroup>
</PNMLextension>
```

Figure 24: A ReML file

In Figure 24, the resource-related data for StpnPlay tool version 0.85b contains three resource definitions and three weaving rules. Each resource is defined using the resource tag while an interaction of a defined resource is specified using the crosscut tag. Their details are discussed in the following two subsections. Since that the ReML is orthogonal to the PeML, the resource-related data in ReML can be placed under the same *PNMLextension* tag, together with the design-specific data in PeML. As shown in Figure 24, the PNMLextension tag surrounds the resource-related data in ReML.

5.4.2 Resource Definition

In ReML, the *resource* tag is used to define an abstract and platform-independent resource. A resource is defined as one of three types, *exclusive*, *shared* and *limited shared*, in terms of its accessibility and availability. The value of the *accessMode* attribute of the resource tag is used to specify whether a resource is shared or exclusive. A similar classification of abstract resources is found in [69]. Based on resource accessibility, a resource is regarded as exclusive if it can be occupied by one or multiple consecutive operations in a service. Otherwise, a resource is considered as shared and can be accessed before or after an operation is executed.

When a shared resource has unlimited availability, it only introduces an access or contention delay without causing a bottleneck. However, some shared resources can be simultaneously accessed only by a predetermined number of operations. For example, an operation can access a common memory bus when any other operation does not use it. This sort of shared resource needs to be explicitly acquired by an extra operation and released then, which probably causes additional waiting delay and quite often leads a bottleneck. Therefore, we need to distinguish the shared resource with limit availability from the one with unlimited. A limited shared resource is identified when its resource definition with the shared access mode has a *limit* attribute in its resource tag. Accordingly, its availability is restricted within a specified value in the limited attribute. Sometimes, it is difficult to tell whether a resource is shared or limited shared. In this case, the decision on a resource type is left as designer's option. On the other hand, all the exclusive resources are regarded as having limited availability so that the limit attribute in their definition is mandatory.

The use of a resource always introduces an access or contention delay that occurs before it is accessed or locked for a resource. Since a contention delay for a resource can be considered as a design-specific data, it is stipulated using the nested *value* tags under a resource tag, which is the same way to specify a design-specific data in PeML. For instance,

a shared resource *network* in Figure 24 has two child nodes to represent its contention delay of which stochastic type and value are exponential and 20.0 respectively. Accordingly, we can define a resource as follows.

Definition 16: A resource *R = (c, l, d)* where *c* is either "shared" or "exclusive" that denotes an access mode, *l* is a non-negative integer number that represents a maximum number of instantiations of the resource, and *d* is a design-specific data such as a delay.

5.4.3 Resource Interference

The *crosscut* tag in ReML is used to specify how a defined resource interferes with a service. Actually, each crosscut tag contains a weaving rule of a defined resource by means of three attributes: *resourceBinding*, *operation*, and *position*. The resourceBinding attribute is used to indicate a defined resource to be weaved, that is called the *bound resource*. The operation attribute is used to denote an operation name that requires the bound resource. The position attribute is differently interpreted according to the type of a bound resource. When the type of a bounded resource is shared or limited shared, the operation attribute is used to address when the operation specified in the operation attribute accesses the bound resource. Remember that the operation is supposed to be atomic so that it can access the bound resource before or after it is executed. Accordingly, the value of the position attribute for a shared or limited shared resource is only allowed to be either *before* or *after*. For instance, the network and memory resource in Figure 24 would be accessed before the execution of the operation *Operation0*. When the type of a bounded resource is exclusive, the operation attribute is used to designate an operation name that returns the bound resource occupied by the operation specified in the operation attribute. Precisely speaking, an operation specified in the operation attribute locks the bound resource before starting and the one in the position

attribute releases after finishing. For example, the processor resource in Figure 24 would be occupied only during the execution of *Operation0*. In summary, a resource interference can be defined as follows.

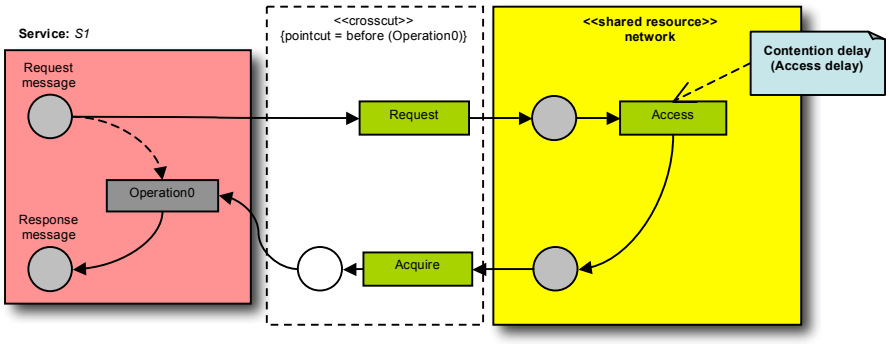
Definition 17: An interference of the resource $R = (c, l, d)$ with an atomic service $S_E = (P_E, T_E, I_E, O_E, M_{0E})$ is specified by either of the following:

- $\theta\ (o)$ if c = "shared", where a crosscutting method θ = "before" or "after", and a crosscut operation $o \in T_E$.
- (o, o') if c = "exclusive", where two crosscut operations $o, o' \in T_E$

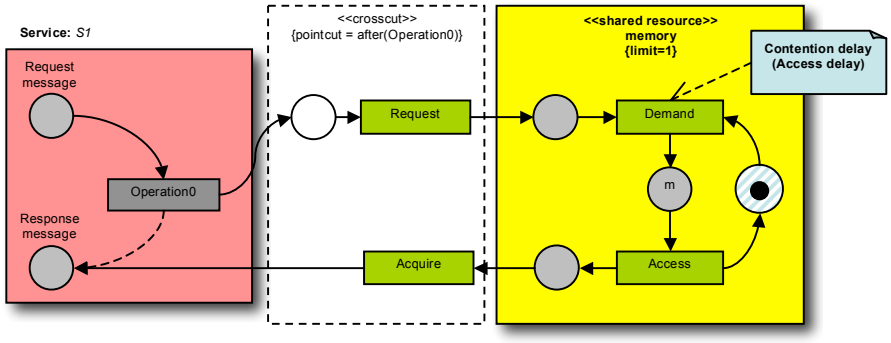
5.4.4 Extension Process

The extension process is the weaving process of adding the resources modeled in the form of Petri net and modifying the Petri net for a service according to the interference specifications. Based on the resource definitions in the resource tag and the specified weaving rules in the crosscut tags, the weaver in the extension mode generates an augmented Petri net by creating aspects for instantiating the resources and attaching them to the Petri net for a service. In Figure 25, three augmented Petri nets are obtained by weaving each of three crosscut tags in the ReML file of Figure 24 into the service *S1* that consists of a single operation and a message pair. When a weaving rule, i.e., a crosscut tag in a ReML file, refers to a defined resource, the weaver program creates its corresponding aspect, called the *resource aspect*. In other words, each crosscut tag is instantiated as a resource aspect with fundamental operations required for manipulating its bound resource. For example, the weaver creates a resource aspect with an access operation and two intermediate messages for the crosscut tag referring to a shared type *network* resource, and weaves it before the operation named *Operation0*. As shown in Figure 25, a resource aspect for a limited shared

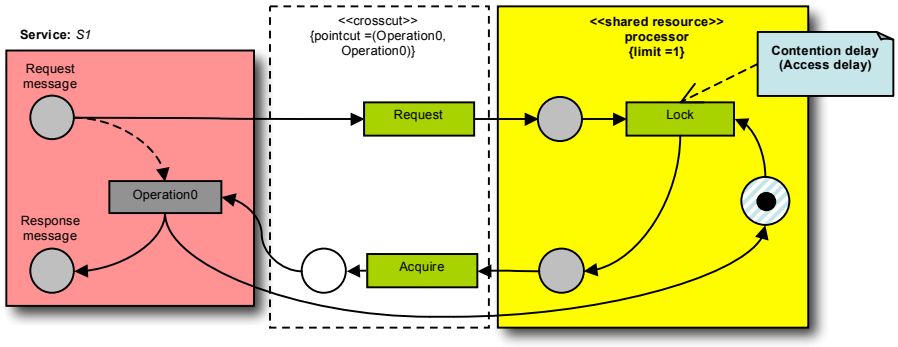
resource requires an additional demand operation before an access operation while a resource aspect for an exclusive resource has a lock operation instead of an access operation.



(a) Shared resource using the before method



(b) Limited Shared resource using the after method



(c) Exclusive shared resource

Figure 25: Weaving instantiated resource aspects

If an explicit behavior is assigned to manipulate a resource and represented using a Petri net, it can be regarded as a common behavior scattered across a module and encapsulated in an aspect. This aspect can be weaved into a Petri net of the module using the Petri net-based weaving rules of the SODA method in the previous chapter. Moreover, we can reuse the Petri net for a common behavior in the aspect when we define another resource.

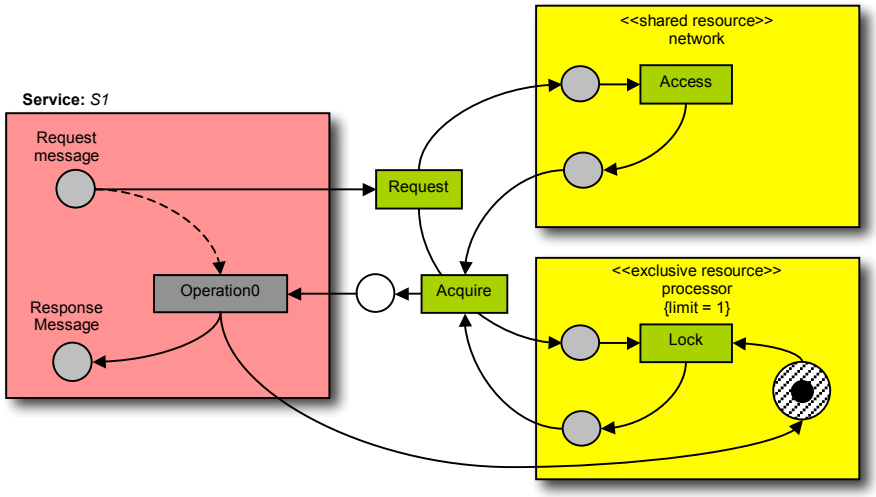


Figure 26: The augmented Petri net with two instantiated resource aspects

The resource aspect is linked with the Petri net for a service via two special operations (transitions), *request* and *acquire*, of which role is almost identical to that of two glue operations, *call* and *return* introduced at the crosscut points during the weaving process of aspects in our service-oriented design approach. A pair of the request/acquire operation could be created before or after an operation only when resource aspects crosscut it. The request operation is used to invoke resource aspects simultaneously while the acquire operation is used to wait for their completion. In addition, the multiple incoming or outgoing arcs of an operation would be merged by the request operation placed before it or forked

from an acquire operation placed after it. For example, Figure 26 shows the weaving result of first two crosscut tags in the ReML file of Figure 24 into the service *S1*. Note that the resources weaved at the same point cannot be invoked in order. This is because we do not assume existence of a resource manager or an operating system that enforces a policy or a rule designated for that purpose.

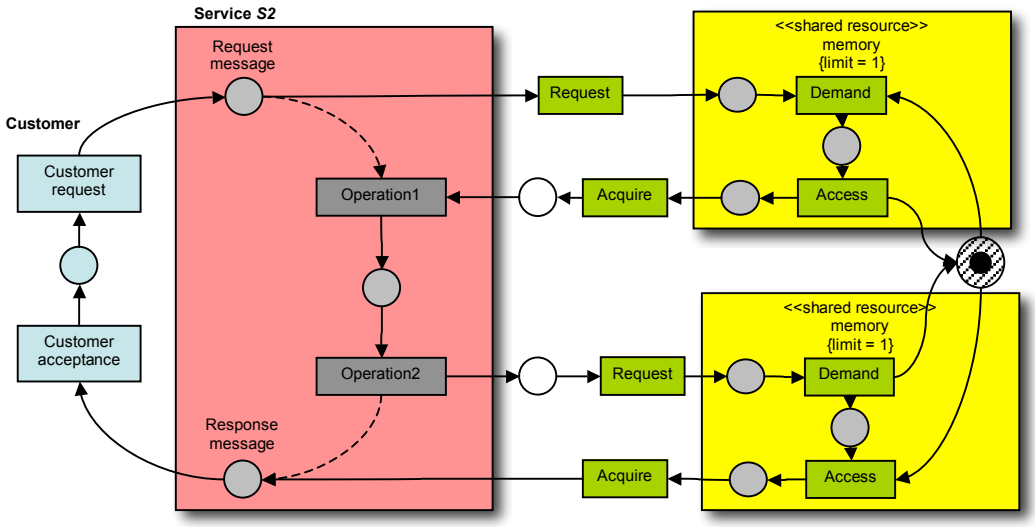


Figure 27: Multiple crosscutting of a service by a resource with the tagged value "limit"

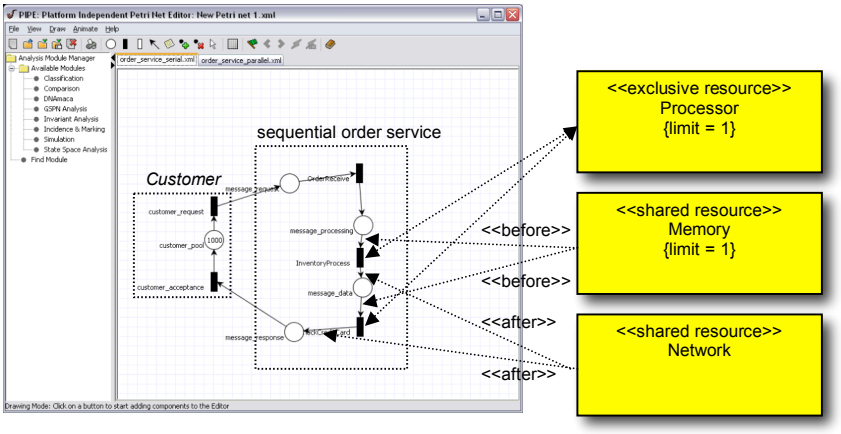
The resource aspects for a limited shared or exclusive resource can only proceed after it obtains one of its available resources. In contrast, the resource aspects for a shared resource are free from this limitation. In terms of the Petri net-based semantics, the resource aspects created from the same limited shared or exclusive resource definition are forced to include a unique shared place with the tokens as many as a number value of its limit attribute. Each token denotes an available resource instance. The shared places for a limited shared and an exclusive resource are denoted by dashed places in Figure 25. Figure 27 clearly exhibits how to represent the case that a limited shared resource is used twice in the module *M2*: before two operations named *Operation1* and after *Operation2* using Petri net.

As mentioned in Section 5.4.2, the access or contention delay of a resource is represented using toolspecific tags created by recursively translating the value tags under its corresponding resource tag. As seen in Figure 25, this delay information is associated with the *access* operation for a shared resource, the *demand* operation for a limited shared resource, or the *lock* operation of an exclusive resource. For instance, the resource weaver would place "<type><value> exponential </value></type>" and "<value><value> 20.0 </value></value>" under the transition tag for the access operation in the resource aspects for a shared resource *network* when processing with the ReML file in Figure 24. Formally, the extension process with the resource-related data can be described as follows.

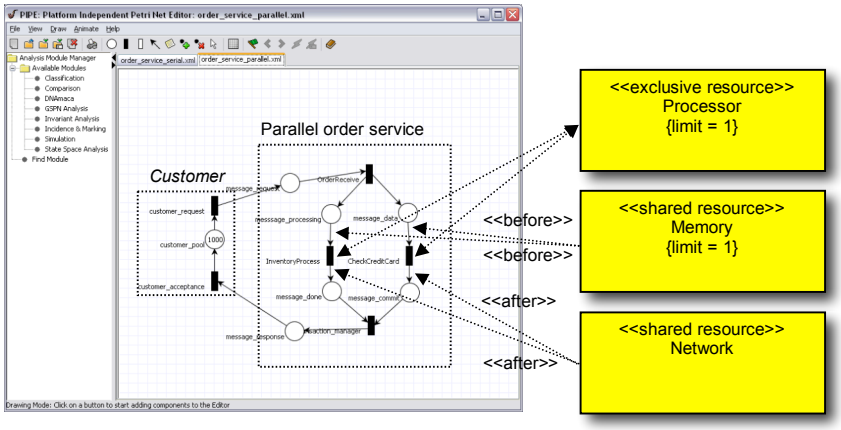
Definition 18: Weaving process with a resource $R = (c, l, d)$ and its interference specification $\theta(o)$ or (o, o') based on Definition 17 to an atomic service $S_E = (P_E, T_E, I_E, O_E, M_{0E})$ generates an augmented Petri net for an atomic service S_W by performing Algorithm 5 (the second weaving algorithm) in Appendix.

5.4.5 Example

This section demonstrates how to apply a resource-related data using a sequential order service and a parallel order service. Each service is represented using a Petri net stored in a PNML file. In Figure 28 we use the PIPE tool [52] to open the PNML files of those two order serves that are a little modified based on the order service shown in Figure 18. Again, any Petri net tool that supports PNML such as Petri net Kernel [53]or Renew [54] can be used to open and examine them. The sequential order service shown in Figure 28-(a) performs two major operations, the *InventoryProcess* and the *CheckCreditCard* operation, in sequence after the *OrderReceive* operation accepts a customer request, whereas the parallel order service in Figure 28-(b) does both of them concurrently.



(a) Sequential order service



(b) Parallel order service

Figure 28: Two order services with the customer modeling

In order to perform simulation during a certain period and eliminate an unnecessary deadlock occurring at the result message place, we need to make the Petri net for the service strongly connected. For this purpose, we add a customer that consists of two transitions connecting with service's request and response places (named the *message_request* and the *message_response* in Figure 28) and a place having initial tokens. This customer also promotes convenience of simulation. For example, we can append a probability distribution

function using a PeML file to the *customer_request* operation of the customer in order to specify an arrival rate of each customer if necessary.

To observe how different resource composition models affect the performance of a server model, we create two ReML files, *Resource_1.reml* and *Resource_2.reml*, based on the ReML file in Figure 24. We use its resource definition without any modification, but replace its weaving rules with the ones shown in Figure 29. Figure 28 delineates how the resource-related data in the ReML file in Resource_2.reml intervenes in two different order services. A rectangle and a dashed arrow denote a resource definition and a crosscut tag, respectively.

```
<crosscut resourceBinding="processor" operation="InventoryProcess" position="CheckCreditCard"/>
<crosscut resourceBinding="memory" operation="InventoryProcess" position="before"/>
<crosscut resourceBinding="network" operation="CheckCreditCard" position="after"/>
```

(a) For Resource_1.reml

```
<crosscut resourceBinding="processor" operation="InventoryProcess" position="InventoryProcess"/>
<crosscut resourceBinding="processor" operation="CheckCreditCard" position="CheckCreditCard"/>
<crosscut resourceBinding="memory" operation="InventoryProcess" position="before"/>
<crosscut resourceBinding="memory" operation="CheckCreditCard" position="CheckCreditCard"/>
<crosscut resourceBinding="network" operation="proxy" position="after"/>
<crosscut resourceBinding="network" operation="InventoryProcess" position="after"/>
<crosscut resourceBinding="network" operation=" CheckCreditCard " position="after"/>
```

(b) For Resource_2.reml

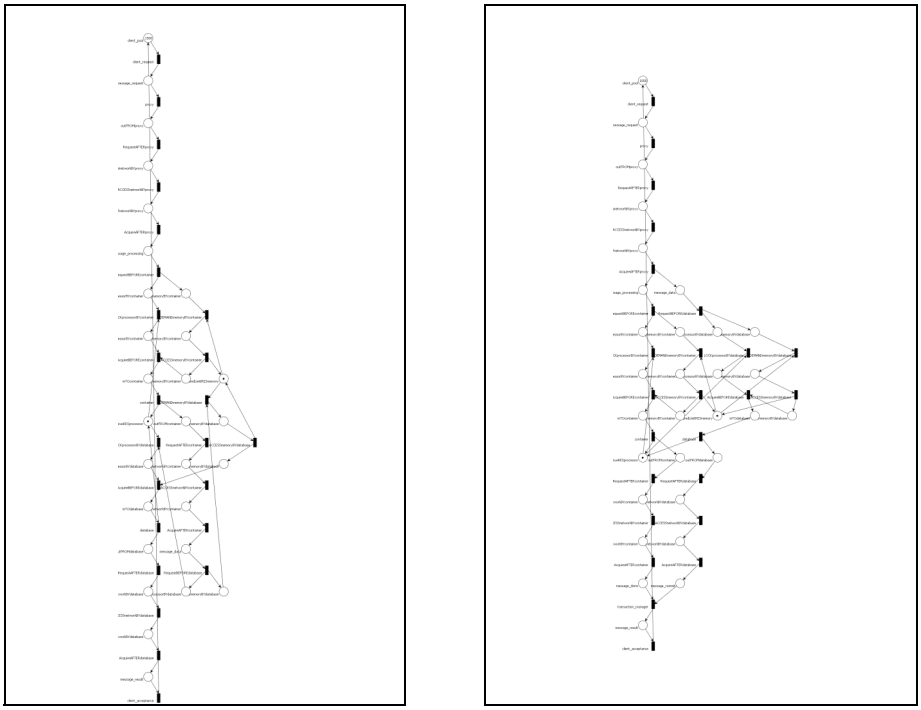
Figure 29: The replaced weaving rules of the ReML file in Figure 24

Table 2: The weaving results of the sequential server model

Weaved ReML file	#of Places	#of Transitions	#of Arcs
None	5	5	10
Resource_1.reml	16	13	32
Resource_2.reml	28	24	60

Table 2 shows the size of the augmented Petri nets obtained through the automatic weaving process with Resource_2.reml and the PNML file for a sequential server model,

which proves that increase of necessary resources and their interactions easily makes the work of manually drawing an augmented Petri net impossible.



(a) Sequential **(b) Parallel**

Figure 30: The augmented Petri nets for two order services after weaving the Resource_2.reml file

Figure 30 represents the augmented Petri nets that correspond to the two order services with the illustrated resource composition model in Figure 28. They are obtained through the automatic weaving process with the PNML file for each server model and Resource_2.reml file via the weaver program implemented in Java. We put 1000 initial tokens in the *customer_pool* place and run simulation on the PIPE tool. The resource limitation caused by the weaved resource-related data has revealed some bottleneck places in the sequential order service so that the tokens initially set in the customer_pool place would

be piled up and left in these places after simulation has done. In other words, we can say that the less interferences a resource-related data causes, the more tokens the customer_pool place has after simulation and the better performance the model extended with it shows.

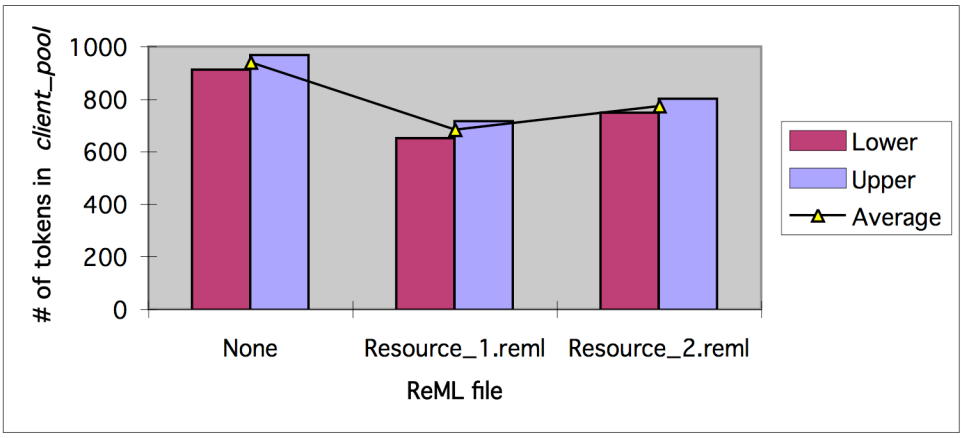


Figure 31: The simulation results of the sequential order service extended with two ReML files

In Figure 31, the simulation results of two augmented Petri nets for the extended sequential servers are shown with the simulation result of the Petri net for the base sequential order service in Figure 28-(a). A lower and an upper number of tokens of the customer_pool place are calculated from three runs of the PIPE simulation that yields the average number of tokens of all the places and its variation for their 95% confidence interval. The sequential order service extended with Resource_1.reml uses the defined resources once over two operation executions after the OrderReceive operation accepts a customer request whereas the one with Resource_2.reml does twice for each of them. Compared with the base sequential order service, the average number of tokens in the customer_pool place of the former decreases by 27.2%, while the one of the latter does by 17.6%. The simulation result shows that the resource-related data in Resource_2.reml deteriorates the performance of the

sequential order service less than the one in Resource_1.reml. This is because the resource-related data specified in Resrouce_2.reml lets those two operations be executed more independently of each other than the one in Resource_1.reml.

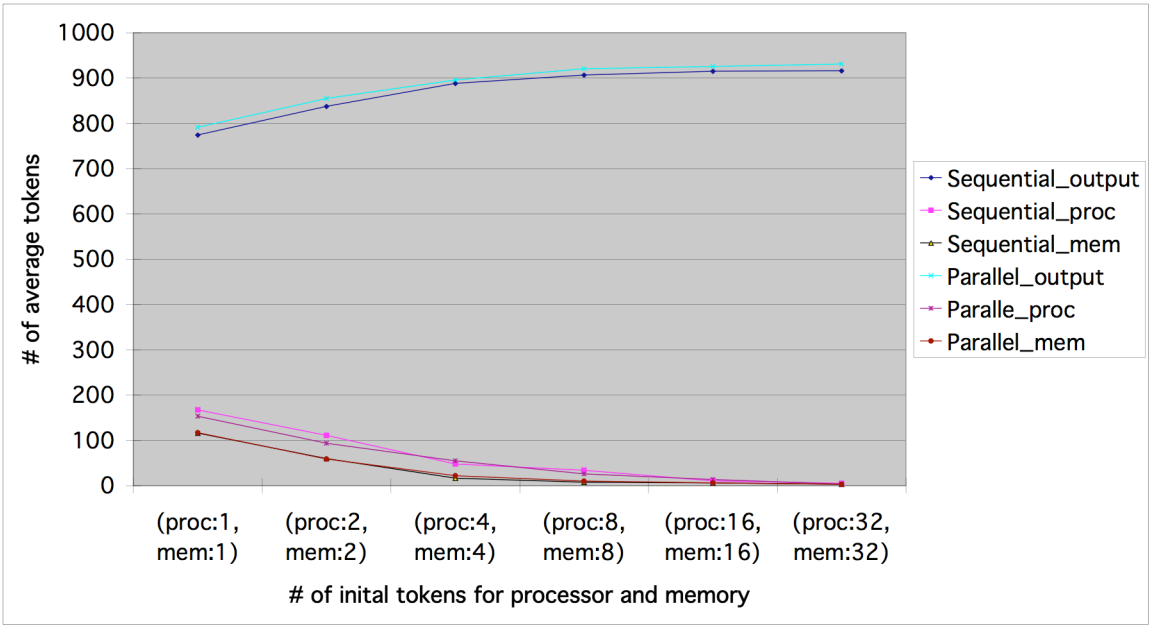


Figure 32: The simulation results of the sequential and the parallel order service with varying resource availability

To find the optimal number of available resources in terms of a resource-related data, we run simulations using the augmented Petri nets for both server models extended with Resource_2.reml in Figure 30. Figure 32 shows the simulation results according to increasing of the numbers of the available limited shared and exclusive resources. An average number of the tokens in the customer_pool place are used again to stand for the performance of two server models. Two bottleneck places caused by the processor and memory are identified and its average numbers of tokens are shown too. In proportion to the increase of the number of available resources, the performances of both order services are improved and their

bottleneck places are mitigated. The performance improvement of both order services happens until the numbers of two available resources become 8. Also, the performance of the parallel order service is always a little better than the one of the sequential order service regardless of the limitation of resource availability.

In this section, we do not show the analysis of the augmented Petri net for those order services extended with design-specific data in a PeML file and the value tags of the resource definitions. Just like the example in the previous section, we can measure or predict its performance more precisely if the temporal or stochastic data for the operations in its base services and the resources in a resource-related data to be weaved could be obtained historically or experimentally.

5.5 Related Work

The Petri net representation of a multiprocessor system in [65] provides a good example of how to abstract major resources such as memory in term of Petri net semantics and supports the claim our simplified resources representation is reasonable from the development perspective. In [66], a resource manager for workflow management systems (wfms) is proposed and modeled using high-level Petri nets. It assigns a free resource in the response to a request of a task connected to it. They assume that a supervisor system takes the responsibility of control workflow by managing jobs and resources, but they omit how to describe it using Petri net. Also, the employment of high-level Petri nets tends to make the development of a model without a concrete algorithm or procedure more complex. In [70], we can find an example of modularizing and implementing a read-write lock pattern for an account class that models accounts in a banking system in AspectJ, which implies that the resource aspects generated by the resource weaver could be treated as a reusable pattern and implemented separately from base modules. In [69], a resource modeling framework is proposed to support changing of the engineering model without changing of the logical

model. The realization mapping using UML materializes a logical model of an application into an engineering model with a set of engineering elements such as processors. Conceptually, it is very similar to our weaving process. However, they leave its semantics and validity as modeler's choice. In [71], software and resource in UML state diagrams are encapsulated using a capsule stereotype and combined via special dispatch modules for each one. Compared with UML-based approaches, our Petri based-approach seems to be more attractive owing to its strong formality and superior performance analysis support via various Petri net tools. Again, PNML [30], an XML-based standard interchange format for Petri nets, would help to encourage Petri nets to be used with other development tools or across different domains like UML.

5.6 Summary and Discussion

In this chapter, an aspect-oriented technique for appending the behavior-specific information is proposed to extend basic Petri nets in PNML to another class of Petri nets, such as timed Petri nets, or model a resource-related data in order to consider resource interferences at the design level. This aspect-oriented extension mechanism is devised to support the analysis of a service design result in Petri net and the comparative evaluation of it against other design results or its variations. In terms of separation of concerns, such behavior-specific information is a particular concern only required and used during a design analysis phases. Therefore, its separation and modularization are possible and necessary for reducing the design process complexity. Moreover, those separated information can be reused. For example, a resource-related data, particularly its resource definitions, can be applied to the variations of a service design result as well as several different design results. Those two data are represented in the XML format: the PNML extension markup language (PeML) and the resource extension markup language (ReML) that can be regarded as part of the SODML in Section 4.4.

A design-specific data in PeML contains a description related to the behavioral characteristics of a service, such as temporal values, that can be applied to its transitions, places, or arcs of the Petri net for that service. The automatic weaving process knits the design-specific data into a basic Petri net. Usually, this design-specific data is used to transform a basic Petri net obtained via the design process to an extended Petri net like a stochastic Petri net. The obtained Petri net can be applied to examine the service designed from the behavioral perspective. Besides, the design-specific data separated in a PeML file increases the interchangeability and interoperability of the PNML file for service design results.

A resource-related data in ReML contains the resource definitions that stand for abstract resources and the weaving rules that explain how to compose the defined resources into a Petri net for a service. A Petri net-represented service with resource interferences can be efficiently generated through weaving a resource-related data and immediately used by various Petri net tools to analyze its behavioral characteristics like performance. During the weaving process, a defined resource is instantiated as a corresponding resource aspect whenever its associated weaving rules specify a crosscut of a Petri net-represented service. This aspect-oriented mechanism makes it possible to develop and reuse a resource-related data regardless of the design and development process of Petri net-represented services. We demonstrate our approach using two variations of the order service and present the simulation results to select a proper resource-related data for a service or identify a better service against a resource-related data. Also, we show the simulation result to identify optimal numbers of available resources in a resource-related data.

Although our current aspect-oriented extension approach does not cover all the factors considered in the design analysis stage, it can be seen as a milestone in the automatic extension of a Petri net-represented service with a set of behavior-specific information – the design-specific data in PeML and the resource-related data in ReML. The augmented Petri

net obtained via our extension method facilitates evaluating a service designed or selecting an optimal or superior one among several service design candidates. Most of all, the separated behavior-specific information and the automatic weaving process are expected to reduce complexity of the development of a service by improving extensibility and maintainability of design results during the development process.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

In terms of service-oriented system design, we propose four objectives - the support of a systematic decomposition process, the standardized representation method of a service-oriented system, the formal and automatic composition of a complex service from the design elements, and the extension mechanism of a service with the data facilitating service analysis. In this chapter, we conclude this dissertation with a summary explaining how our approach supports the objectives proposed and contributes to the state of the art of software system development, in particular, the design process of service-oriented systems. In the end, we discuss some directions of future work.

6.1 Conclusions

Our service-oriented design approach presented in this dissertation fully exploits the fundamental concepts of aspect-oriented programming. Due to fact that the concept of aspect is consistently applied and tightly integrated into the design phases, our approach can enhance separation of concerns, which consequently helps to decrease the complexity of the entire design process. In particular, the Petri net-based formalism supports the weaving process that constructs services from design elements separated during the decomposition process.

The FCD-A method, extended from its origin of the Function-Class Decomposition (FCD) method as a generic decomposition method integrated with the concept of aspect, is presented to decompose any software systems including service-oriented systems. It organizes the functional elements in the function-class view and categorizes other supplementary functional/non-functional elements in the aspect view. FCD-A also provides a hierarchical view of a software system and a classified view of the important concerns. In the case of the decomposition of service-oriented systems, FCD-A can be performed to assign a primitive service, that contains a basic process to perform a specific task, to each functional

module or class, whereas the simultaneously identified aspects can capture service-specific or domain-specific concerns required for delivering high-quality and user-friendly services. At the initial design stage, FCD-A supports a wide range of concerns with improved separation of concerns during the decomposition process.

Using primitive services and aspects identified via FCD-A, the Service-Oriented Design with Aspects (SODA) method delineates a service-oriented system in two views, the structural view and behavioral view, based on an extended UML2 and Petri net representations, respectively. In particular, SODA supports an automatic weaving process based on the Petri net-based semantics and a standardized XML-based representation. It can generate the integrated Petri net for a complex service composed of a primitive service and a set of aspects based on the relationships between them. With the integrated Petri net obtained through the weaving process, SODA facilitates the verification and evaluation of service design results. Moreover, our aspect-oriented design approach makes it possible to evolve service designs in existence or construct various versions of a service design with a reduced development effort by replacing or reusing existing design elements, especially aspects.

A precise assessment or comparative evaluation of the integrated Petri net of a service requires some behavior-related information tailored to the design analysis process. An aspect-oriented extension mechanism provides two different extension processes of an integrated Petri net according to whether those behavior-related information is the design-specific data or resource-related data. An automatic weaving process is also supported to append the design-specific data to an integrated Petri net of a service or reflect resource-related data within it. As a result, the integrated Petri net of a service can become another class of a Petri net or an augmented Petri net that is more useful and widely applicable to identify the behavioral characteristics of the corresponding service including performance evaluation.

In summary, our service-oriented design approach to decompose, represent, construct, and analyze a service-oriented system with highly sensitive quality attributes as well as diverse functionalities utilizes the concept of aspect in a comprehensive and consistent manner. As a result, our approach is expected to achieve improvement in reusability, customizability, manageability maintainability, traceability and interchangeability of the design results. Most of all, a formal and automatic weaving process for building and extending a service in the design stage constitutes the step stone towards a completely automated design process. Our aspect-oriented and Petri net-based approach contributes to the field of service-oriented computing as well as software engineering by providing a systematic, comprehensive, yet generic and formal service-oriented design method that makes it possible to develop flexible and quality-guaranteed service-oriented systems in an efficient and effective fashion.

6.2 Future Work

Since our service-oriented design approach is generic, it is insufficient to tackle the detailed or domain-specific factors such as a dynamic adaptation feature of some service-oriented systems. Hence, one area of future work involves further refinement or enrichment of our approach so as to include detailed or domain-specific factors in our generic design approach. This includes the development of a new syntax and semantics for the weaving process according to the domain-specific system to be developed. For example, a dynamic service-oriented system needs to change the associated aspects including resources according to the flow of time, which requires to devise a new syntax for addressing a time frame in which a set of relationships with the specific factors, such as the population of customers, are specified, or to introduce crosscutting methods specific to that sort of dynamic service-oriented systems.

A significant concern is that our service-oriented design approach clings to the underlying concepts and released specifications currently in use for service-oriented computing. For instance, the Petri net-based behavioral representation of services of SODA hinders modeling of service processes with multiple or nested alternative flows, which requires our approach to support an extension method to convert a basic Petri net to a high-level Petri net like CPN. This kind of extension would be studied and taken into account. Another example is that the representation part of our SODA approach is based on the specific versions of UML2 and PNML. Therefore, our approach is expected to be updated including the update of the weaver program, at the pace of changes of these specifications.

In terms of interchangeability or interoperability, our design approach needs to be supported by an adequate tool set. To support our design approach with a cooperative and integrated work, we need to provide a plug-in or a conversion program that renders our service-oriented design results in our XML-based representations visual using a graphical design tool. To promote the convenience of the weaving process, our weaver program could be combined into an integrated development tool (IDE) or a Petri net tool, along with a way of specifying the crosscut relationships or the behavior-specific data. A management tool would be considered to provide a tool-supported traceability or maintainability of our service-oriented design elements being developed.

Based on the Petri net formalism and the simulation results that provide convincing data as a proof, our approach is in theory expected to satisfactorily address many software design issues such as reusability, customizability, manageability, traceability and interchangeability. However, further experiments are necessary to investigate how much our design approach is useful in the real-world environment or better than other design approaches. Accordingly, our future work includes a wide range of case studies to demonstrate the suitability of our service-oriented design approach, especially, by applying our method to the design of real-life or large-scale service-oriented systems. We anticipate

that the outcome out of such demonstration projects would result in identifying some critical criteria or metrics to be used for systematic comparisons between our presented design approach and others, as well as the measurement or evaluation of agility and flexibility of our design approach. One specific target demonstration project is an on-going work to develop frameworks that support the development of a smart home prototype in the Software Engineering Laboratory at Iowa State University.

APPENDIX. ALGORITHMS

Algorithm 1: The decomposition algorithm of FCD-A

The detailed steps of the FCD-A method are described as follows.

1) Initially, the whole system consists of a single functional module and key aspects to be considered. Place the key aspects as child aspects of the virtual root, “Aspect”. Attach the top-level requirements and the scenarios to the initial module.
2) Allocate the requirements including non-functional ones and their related scenarios to the functional module at the current level and, using them, identify the necessary classes that satisfy the functionality of the current level.
3) For identified classes within each functional module, rearrange or reallocate the requirements and, using the requirements, identify aspects within the class.
4) Create aspect links from classes or functional modules to corresponding aspects. There are two types of the aspect link. For the implementable feature, use an *operational link*. For a method or a policy, apply a *strategic link*. Assign a feature-relevant name and attach the requirements to the aspect link.
5) Examine the roles and interactions of each class in each functional module and put the classes under the subgroups that can minimize external coupling and maximize internal cohesion.
6) Using the use case maps for each functional module, assess coupling and cohesion of subgroups by selecting core scenarios and analyzing their paths. If needed, regroup classes to meet the coupling and cohesion criteria in step 5.
7) If an aspect link starting from functional modules does not affect all of its lower-level functional modules or its classes move it down to the appropriate functional modules or classes.
8) Find the commonality in the aspect links. From the aspect view, if several aspect links have the common feature or the requirements, make a new child aspect and put it under the appropriate high-level aspect. A new child aspect for a single link is allowed. From the function-class view, if a common strategic link occurs in all the classes under a subgroup, it can be move up to the subgroup.
9) Search the conflicts among the names of the aspect links for each aspect. If these conflicts cannot be resolved by moving the aspect links to another aspect, make child aspects under the aspect and adjust the aspect links.

10) Create and name the next-level functional module.
11) Allocate the requirements and scenarios to the each new functional module. The requirements and scenarios across functional modules should not be moved to the child functional modules.
12) Repeat from step 2 to 11 until the decomposition of the system identifies all the primary functionality and aspects. If FCD-A is used for work allocation in distributed development environment, we can stop decomposing the system earlier.
13) Using all the primary functionality and aspects, draw UML class diagram for each leaf node and an appropriate UML diagram for each aspect. Each aspect can generate a stereotyped package or a collaboration diagram by grouping the related classes. A new extended UML diagram can be used if available.
14) From the lowest level in the function-class view, define interfaces between functional modules at the intermediate-levels using the allocated scenarios and the related aspect links, which integrate all the functional modules and weave the aspects into the entire system. This integration step is considerably affected by the development environment.

Algorithm 2: The binding algorithm to construct an atomic service

Input: An atomic action $A = (P, T, I, O, M_0)$, a message pair $mp = (p_i, p_o)$, and two sets of the transitions $T_I = \{t_1, \ldots, t_n\} \subset T$ and $T_O = \{t_1, \ldots, t_m\} \subset T$.

Output: An atomic service S

Steps: $P \leftarrow P \cup \{p_i, p_o\}$

IF $| T_I | = 1$ **THEN**
$I \leftarrow I \cup (p_i, T_I)$
ELSE
$P' = \{p_1, \ldots p_n\}$ be a set of cloned messages of p_i , where $n = | T_I |$
$P \leftarrow P \cup P'$
$T \leftarrow T \cup \{t_f\}$ where t_f is a fork operation
$I \leftarrow I \cup \{(p_i, t_f)\} \cup \{(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)\}$
$O \leftarrow O \cup (\{t_f\} \times P')$
ENDIF
IF $| T_O | = 1$ **THEN**
$O \leftarrow O \cup (T_O, p_o)$
ELSE
Let $P'' = \{p_1, \ldots p_m\}$ be a set of cloned messages of p_o , where $m = | T_O |$
$P \leftarrow P \cup P''$
$T \leftarrow T \cup \{t_j\}$ where t_j is a join operation

$I \leftarrow I \cup (P'', \{t_j\})$
$O \leftarrow O \cup \{(t_j, p_o)\} \cup \{(t_1, p_1), (t_2, p_2), \ldots, (t_m, p_m)\}$
ENDIF

Algorithm 3: The weaving algorithm to generate an integrated Petri net

Input: An atomic service $S_E = (P_E, T_E, I_E, O_E, M_{0E})$ in a service entity, an advice $S_C = (P_C, T_C, I_C, O_C, M_{0C})$ in an aspect, a crosscutting method θ, and its corresponding messages m_{pre}, m_{post} and operations o, o' for crosscut points

Output: An atomic service $S_W = (P_W, T_W, I_W, O_W, M_{0W})$

Steps: $P_W \leftarrow P_E \cup P_C$
$T_W \leftarrow T_E \cup T_C$
$I_W \leftarrow I_E \cup I_C$
$O_W \leftarrow O_E \cup O_C$
$M_{0W} \leftarrow M_{0E} \cup M_{0C}$
$mp_C = \{p_i, p_o\} \in P_C$, where p_i, p_o are the input/output place for the interface of advice S_C
IF θ = "before" **THEN**
$cp \leftarrow (m_{pre}, o)$
$T_W \leftarrow T_W \cup \{t_{call(cp)}, t_{return(cp)}\}$
$P_W \leftarrow P_W \cup \{p_{input(cp)}\}$
$I_W \leftarrow I_W \cup \{(m_{pre}, t_{call(cp)}), (p_o, t_{return(cp)}), (p_{input(cp)}, o)\} - \{cp\}$
$O_W \leftarrow O_W \cup \{(t_{call(cp)}, p_i), (t_{return(cp)}, p_{input(cp)})\} - \{(t_{call(cp)}, p_{input(cp)})\}$
ELSEIF θ = "after" **THEN**
$cp \leftarrow (o, m_{post})$
$T_W \leftarrow T_W \cup \{t_{call(cp)}, t_{return(cp)}\}$
$P_W \leftarrow P_W \cup \{p_{output(cp)}\}$
$I_W \leftarrow I_W \cup \{(p_{output(cp)}, t_{call(cp)}), (p_o, t_{return(cp)})\} - \{(p_{output(cp)}, t_{return(cp)})\}$
$O_W \leftarrow O_W \cup \{(o, p_{output(cp)}), (t_{call(cp)}, p_i), (t_{return(cp)}, m_{post})\} - \{cp\}$
ELSE
$cp1 \leftarrow (m_{pre}, o)$
IF θ = "flow" **THEN**
$cp2 \leftarrow (o', m_{post})$
ELSE
$cp2 \leftarrow (o, m_{post})$
ENDIF
$T_W \leftarrow T_W \cup \{t_{call(cp1)}, t_{return(cp2)}\}$
$I_W \leftarrow I_W \cup \{(m_{pre}, t_{call(cp1)}), (p_o, t_{return(cp2)})\} - \{cp1\}$
$O_W \leftarrow O_W \cup \{(t_{call(cp1)}, p_i), (t_{return(cp2)}, m_{post})\} - \{cp2\}$
IF θ = "proceed" or θ = "flow" **THEN**
IF $\exists\ p_{input(cp1)}$ = FALSE **THEN**
$P_W \leftarrow P_W \cup \{p_{input(cp1)}\}$

$I_W \leftarrow I_W \cup \{(p_{input\ (cp1)}, o)\}$
$O_W \leftarrow O_W \cup \{(t_{call(cp1)}, p_{input(cp1)})\}$
ENDIF
IF $\exists\ p_{output(cp2)}$ = FALSE **THEN**
$P_W \leftarrow P_W \cup \{p_{output(cp2)}\}$
$I_W \leftarrow I_W \cup \{(p_{output(cp2)}, t_{return(cp2)})\}$
$O_W \leftarrow O_W \cup \{(o, p_{output(cp2)})\}$
ENDIF
ENDIF
ENDIF

Algorithm 4: The additional steps for Algorithm 3 to process an advice in the aspect with a "sync" value

Steps: $P_W \leftarrow P_W \cup \{p_{sync\_Sc\ (Sc)}\}$
$M_{0W}\ (p_{sync\_Sc\ (Sc)}) \leftarrow k$
FOREACH S_C is copied
Let $t_{enter} \in T_C$ be the transition of which input arc is from p_i such that $(p_i, t_{enter}) \in I_C$
Let $t_{commit} \in T_C$ be the transition of which output arc is to p_o such that $(t_{commit}, p_o) \in I_C$
$I_W \leftarrow I_W \cup \{(p_{sync\_Sc\ (Sc)}, t_{enter}) \}$
$O_W \leftarrow O_W \cup \{(t_{commit}, p_{sync\_Sc\ (Sc)})\}$
ENDFOREACH

Algorithm 5: The weaving algorithm to generate an augmented Petri net with resource interference

Input: An atomic service $S_E = (P_E, T_E, I_E, O_E, M_{0E})$, an resource $R = (c, I, d)$, a crosscut operation o and crosscutting method θ = "before" or "after" if c = "shared" or crosscut operations o, o' if c = "exclusive"
Output: An atomic service $S_W = (P_W, T_W, I_W, O_W, M_{0W})$
Steps: **IF** c ="shared" **THEN**
IF θ ="before" **THEN**
cp $\leftarrow (m_{pre}, o)$, where m_{pre} is any input place of the transition named o
$P_W \leftarrow P_E \cup \{p_{ready(cp)}, p_{input(R, cp)}, p_{output(R, cp)}\}$
$T_W \leftarrow T_E \cup \{t_{request(cp)}, t_{acquire(cp)}, t_{access(R, cp)}\}$
$I_W \leftarrow I_E \cup \{(m_{pre}, t_{request(cp)}), (p_{output(R, cp)}, t_{acquire(cp)}), (p_{ready(cp)}, o)\}$ - {cp}
$O_W \leftarrow O_E \cup \{(t_{request(cp)}, p_{input(R, cp)}), (t_{access(R, cp)},, p_{output(R, cp)}), (t_{acqure(cp)}, p_{ready(cp)})\}$
$M_{0W} \leftarrow M_{0E}$

ELSEIF θ = "after" **THEN**

cp ← (o, m_{post}), where m_{post} is any output place of the transition named o

$P_W \leftarrow P_E \cup \{p_{wait(cp)}, p_{input(R, cp)}, p_{output(R, cp)}\}$

$T_W \leftarrow T_E \cup \{t_{request(cp)}, t_{acquire(cp)}, t_{access(R, cp)}\}$

$I_W \leftarrow I_E \cup \{(p_{wait(cp)}, t_{request(cp)}), (p_{output(R, cp)}, t_{acquire(cp)}), (p_{ready(cp)}, m_{post})\}$

$O_W \leftarrow O_E \cup \{(o, p_{wait(cp)}), (t_{request(cp)}, p_{input(R, cp)}), (t_{access(R, cp)}, p_{output(R, cp)}), (t_{acqure(cp)}, p_{ready(cp)},)\} - \{cp\}$

$M_{0W} \leftarrow M_{0E}$

ENDIF

IF I = 0 **THEN**

$I_W \leftarrow I_W \cup \{(p_{input(R, cp)}, t_{access(R, cp)})\}$

ELSE

$T_W \leftarrow T_W \cup \{t_{demand(R, cp)}\}$

$P_W \leftarrow P_W \cup \{p_{transient(R, cp)}, p_{(R)}\}$

$I_W \leftarrow I_W \cup \{(p_{input(R, cp)}, t_{demand(R, cp)}), (p_{transient(R, cp)}, t_{access(R, cp)}), (p_{(R)}, t_{demand(R, cp)})\}$

$O_W \leftarrow O_W \cup \{(t_{demand(R, cp)}, p_{transient(R, cp)}), (t_{access(R, cp)}, p_{(R)})\}$

$M_{0W}(p_{(R)}) \leftarrow I$

ENDIF

ENDIF

IF c = "exclusive" **THEN**

cp ← (m_{pre}, o), where m_{pre} is any input place of the transition named o

$P_W \leftarrow P_E \cup \{p_{ready(cp)}, p_{input(R, cp)}, p_{output(R, cp)}, p_{(R)}\}$

$T_W \leftarrow T_E \cup \{t_{request(cp)}, t_{acquire(cp)}, t_{lock(R, cp)}\}$

$I_W \leftarrow I_E \cup \{(m_{pre}, t_{request(cp)}), (p_{input(R, cp)}, t_{lock(R, cp)}), (p_{output(R, cp)}, t_{acquire(cp)}), (p_{ready(cp)}, o), (p_{(R)}, t_{lock(R, cp)})\} - \{cp\}$

$O_W \leftarrow O_E \cup \{(t_{request(cp)}, p_{input(R, cp)}), (t_{lock(R, cp)}, p_{output(R, cp)}), (t_{acqure(cp)}, p_{ready(cp)}), (o, p_{(R)})\}$

$M_{0W} \leftarrow M_{0E}$

$M_{0W}(p_{(R)}) \leftarrow I$

ENDIF

// The design specific data *d* of the resource R is associated with

// either $t_{access(R, cp)}$ or $t_{lock(R, cp)}$ if necessary

BIBLIOGRAPHY

[1] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communication of ACM*, vol. 15, no. 12, 1053-1058, December 1972.

[2] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr., "N Degree of Separation: Multi-Dimensional Seperation of Concerns", *Proceedings of the 21st Int'l Conference on Software Engineering (ICSE)*, pp.107-119, May 1999.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," *Proceedings of the European Conference on Object-Oriented Programming (ECCOP'97)*, LNCS 1241, Springer-Verlag, pp. 222-242, 1997.

[4] C. K. Chang, J. Cleland-Huang, S. Hua, and A. Kuntzmann-Combelles, "Function-Class Decomposition: A Hybrid Software Engineering Method," *IEEE Computer*, pp. 87-93, December 2001.

[5] W. Harrison and H. Ossher, "Subject-Oriented Programming: A Critique of Pure Objects," *Proceedings of the 8th Annual Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'93)*, pp. 411-428, 1993.

[6] S. Clarke and R. J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects," *Proceedings of the 23rd Int'l Conference on Software Engineering (ICSE)*, pp.5-14, May 2001.

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pp. 327-353, 2001.

[8] C. A. Constantinides and T. H. Elrad, "On the Requirements for Concurrent Software Architectures to Support Advanced Seperation of Concerns," *OOPSLA 2000 Workshops on Advanced Separation of Concerns in Object-Oriented Systems*, 2000.

[9] D. Wang, F. B. Bastani, and I.-L. Yen, "Automated Aspect-Oriented Decomposition of Process-Control Systems for Ultra-High Dependability Assurance**,"** *IEEE Transactions on Software Engineering*, vol. 31, no. 9, pp. 713- 732, September 2005.

[10] E. Baniassad, P. C. Clements, J. Araújo, A. Moreira, A. Rashid, and B. Tekinerdoğan, "Discovering Early Aspects," *IEEE Software*, pp. 61-70, January/February 2006.

[11] E. Baniassad and S. Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design," *Proceedings of the 26th Int'l Conference on Software Engineering (ICSE'04)*, pp. 158-167, 2004.

[12] B. Tekinerdogan, "ASAAM: Aspectual Software Architecture Analysis Method," *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pp. 5-14, 2004.

[13] A. Moreira, A. Rashid, and J. Araújo, "Multi-Dimensional Separation of Concerns in Requirements Engineering," *Proceedings of the 13th IEEE Int'l Conference on Requirements Engineering*, pp. 285-296, 2005.

[14] C. A. Constantinides, A. Bader, T.H. Elrad, M. E. Fayad, and P. Netinant, "Designing an Aspect-Oriented Framework in an Object-Oriented Environment," *ACM Computing Surveys*, vol. 32, no.1, pp.41-53, March 2000.

[15] G. M. C. de Sousa, I. G. L. da Silva, J. B.de Castro, "Adapting the NFR Framework to Aspect-Oriented Requirements Engineering," *Proceeding of XVII Brazilian Symposium on Software Engineering*, pp.83-98, 2003.

[16] J. Grundy, "Aspect-Oriented Requirements Engineering for Component-based Software Systems," *Proceedings of IEEE Int'l Symposium on Requirements Engineering*, pp. 84-91, 1999.

[17] N. Noda and T. Kishi, "On Aspect-Oriented Design: An Approach to Designing Quality Attributes," *Proceedings of 6th Asia Pacific Software Engineering Conferences (APSEC'99)*, pp. 230-237, 1999.

[18] A. Rashid, P. Sawyes, A. Moreira, and J. Araújo, "Early Aspects: a Model for Aspect-Oriented Requirements Engineering," *Proceddings of IEEE Joint Int'l Conferences on Requirements Engineering*, pp. 199-202, 2002.

[19] A. Rashid, A. Moreira, and J. Araújo, "Modularisation and Composition of Aspectual Requirements," *Proceedings of the 2nd Int'l Conference on Aspect-Oriented Software Development (AOSD'03),* pp. 11-20, 2003.

[20] S. M. Sutton Jr. and I. Rouvellou, "Modeling of Software Concerns in Cosmos," *Proceedings of the 1st Int'l Conference on Aspect-Oriented Software Development (AOSD'02),* pp. 127-133, 2002.

[21] M. Katara and S. Katz, "Architectural Views of Aspects," *Proceedings of the 2nd Int'l Conference on Aspect-Oriented Software Development (AOSD'03),* pp. 1-10, 2003.

[22] B. Nuseibeh, "Weaving Together Requirements and Architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115-119, March 2001.

[23] E. A. Kendall, "Role Modeling for Agent System Analysis, Design, and Implementation," *IEEE Concurrency*, pp. 34-41, April-June 2000.

[24] International Center for Software Engineering (ICSE), "M-Net: Meeting Net Online Demo," 2002, Available: http://icse.cs.iastate.edu.

[25] C. Zhang and H.-A. Jacobsen, "Quantifying Aspects in Middleware Platforms," *Int'l Conference of Aspect Oriented Software and Development (AOSD)*, pp. 130-139, 2003.

[26] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," *Proceedings of the 4th Int'l Conference on Web Information Systems Engineering (WISE'03)*, pp. 3-12, December 2003.

[27] R. Perrey and M. Lycett, "Service-Oriented Architecture," *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, pp. 116-119, January 2003.

[28] Object Management Group (OMG), *Unified Modeling Language (UML) 2.0 Specifications*, August 2005, Available: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.

[29] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, April 1989.

[30] J. Billington, S. Christensen, K. Hee, E. Kindler, O. Kummer, L. Petricci, R. Post, C, Stehno, and M. Weber, "The Petri Net Markup Language: Concepts, Technology, and Tools," March 2003, Available: http://www.informatik.hu-berlin.de/top/pnml/about.html.

[31] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4-7, 2001.

[32] E. Gamma, R. Helm, R. Johanson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, ISBN**:** 0201633612, 1995.

[33] H. Kreger, "Web Service Conceptual Architecture (WSCA 1.0)," May 2001, Available: http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[34] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *Business Process Execution Language for Web Services (BPEL4WS) Specification Version 1.1*, May 2003, Available: http://www.ibm.com/developerworks/library/ws-bpel/.

[35] B. Meyer, "Applying "Design by Contract"," *IEEE Computer*, vol. 25, no. 10, pp. 40-51, October 1992.

[36] J. Baik, N. Eickelmann, and C. Abts, "Empirical Software Simulation for COTS Glue Code Development and Integration," *Proceedings of the 25th Annual Int'l Computer Software and Application Conference (COMPSAC'01)*, pp. 297-302, October 2001.

[37] L. Baresi, R. Heckel, S.Thöne, and D. Varró, "Modeling and Validation of Service-Oriented Architectures: Application vs. Style," *Proceedings of the 9th European*

Software Engineering Conference held jointly with 11th ACM SIGSOFT Int'l Symposium on the Foundations of Software Engineering (ESEC/FSE'03), pp. 68-77, September 2003.

[38] M. Deubler, J. Grünbauer, G. Popp, G. Wimmel, and C. Salzmann, "Toward a Model-Based and Incremental Development Process for Service-Based Systems," *Proceedings of IASTED Int'l Conference on Software Engineering (SE 2004)*, pp. 183-188, February 2004.

[39] I. H. Krüger and R.Mathew, "Systematic Development and Exploration of Service-Oriented Software Architectures," *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WISCA'04)*, pp. 177-187, June 2004.

[40] V. Tosic, K. Patel, and B. Pagurek, "WSOL – Web Service Offering Language," *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web (WES 2002)*, LNCS 2512, pp. 57-67, 2002.

[41] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan, *Adaptive and Dynamic Service composition in eFlow*, Technical Report, HPL-2000-39, HP Labs, March 2000.

[42] B. Benatallah, Q. Z. Sheng, and M. Dumas, "The Self-Serv Environment for Web Services Composition," *IEEE Internet Computing*, vol. 7, no. 1, pp. 40-48, January/February 2003.

[43] L. Zeng, B. Benatallah, and M. Dumas, "Quality Driven Web Services Composition," *Proceedings of the 12th Int'l Conference on World Wide Web (WWW 2003)*, pp. 411-421, May 2003.

[44] J. Grundy, T. Panas, S. Singh, and H. Stöckle, "An Approach to Developing Web Service with Aspect-oriented Component Engineering," *Proceedings of the 2nd Nordic Conference on Web Services*, 2003, Available: http://www.cs.auckland.ac.nz/~john-g/papers/ncws2003.pdf.

[45] A. Charfi and M. Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," *Proceedings of European Conference on Web Services (ECOWS 2004)*, pp. 168-182, September 2004.

[46] D. Suvée, W. Vanderperren, and V. Jonckers, "JAsCo: an Aspect-Oriented approach for Component Based Software Development," *Proceedings of the 2nd Int'l Conference on Aspect-Oriented Software Development (AOSD 2003)*, pp. 21-29, March 2003.

[47] B. Verheecke, M. A. Cibrán, and V. Jonckers, "AOP for Dynamic Configuration and Management of Web Services," *Proceedings of the Int'l Conference on Web Services – Europe (ICWS-Europe'03)*, LNCS 2853, pp. 137-151, September 2003.

[48] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, OMG Press, ISBN: 0471319201, 2003.

[49] K. Schmidt and C. Stahl, "A Petri Net Semantics for BPEL4WS – Validation and application," *Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN'04)*, pp. 1-6, September 2004.

[50] World Wide Web Consortium (W3C) Recommendation, *XML Transformation (XSLT) Version 2.0*, January 2007, Available: http://www.w3.org/TR/xslt20/.

[51] Sun Microsystems, *Java 2 Platform Standard Edition Development Kit (JDK) 5.0*, September 2004, Available: http://java.sun.com/.

[52] Imperial College DoC MSc Group and MSc Individual Project, *Platform Independent Petri Net Editor (PIPE)*, March 2007, Available: http://pipe2.sourceforge.net/.

[53] Petri Net Kernel Team (Humboldt-University Berlin, Germany), *Petri Net Kernel (PNK)*, June 2002, Available: http://www.informatik.hu-berlin.de/top/pnk/.

[54] O. Kummer, F. Wienberg, and M. Duvigneau, *Renew*, May 2006, Available: http://www.renew.de/.

[55] J. Čapek, *StpnPlay: A Stochastic Petri Net Modeling and Simulation Tool*, February 2002, Available: http://dce.felk.cvut.cz/capekj/StpnPlay/.

[56] World Wide Web Consortium (W3C) Recommendation, *XQuery 1.0: An XML Query Language*, January 2007, Available: http://www.w3.org/TR/xquery/.

[57] Petri Net Markup Language (PNML) homepage, January 2006, Available: http://www.informatik.hu-berlin.de/top/pnml/.

[58] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services*, November 2004, Available: http://www.daml.org/services/owl-s/1.1/overview/.

[59] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, and M. Rowley , "BPELJ: BEPL for Java", March 2004, Available: http://ftpna2.bea.com/pub/downloads/ws-bpelj.pdf.

[60] C. Stehno, "Interchangeable High-Level Time Petri Nets," *Workshop on the Petri Net Markup Language 2005 (PNML 05) - Towards an ISO/IEC Standard Transfer Syntax for Petri Nets*, Helsinki University of Technology, Finland, May 2005.

[61] R. Eshuis and R. Wieringa, "Tool support for Verifying UML Activity Diagrams," *IEEE Transactions on Software Engineering*, vol. 30, no. 7, pp. 437- 447, July 2004.

[62] H. Störrle, "Semantics of Control-Flow in UML2.0 Activities," *Proceeding of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing (VLHCC'04)*, pp. 235-242, September 2004.

[63] H. Yu, D. Liu, X. He, L. Yang, and S. Gao, "Secure Software Architectures Design by Aspect Orientation," *Proceedings of the 10th Int'l Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pp. 47-55, June 2005.

[64] D. Xu and K. Nygard, "Threat-driven modeling and verification of secure software using aspect-oriented Petri nets**,"** *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 265- 278, April 2006.

[65] M. A. Marsan, G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transaction on Computer Systems*, vol. 2, no. 2, pp. 93-122, 1984.

[66] W.M.P. van del Aalst, K.M. van Hee, and C.J. Houben, "Modelling and Analysing Workflow Using a Petri-net Based Approach," *Proceedings of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pp. 31-50, 1994.

[67] C. K. Chang and S. Kim, "I3: A Petri-net Based Specification Method for Architectural Components," *Proceedings of the 23rd Annual Int'l Computer Software and Application Conference (COMPSAC'02)*, pp. 396-402, 1999.

[68] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol 1, Second Edition, ISBN**:** 3540609431, 1997.

[69] B. Selic, "A Generic Framework for Modeling Resources with UML," *IEEE Computer*, pp. 64-69, vol. 33, no. 6, June 2000.

[70] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming, Manning Publications*, ISBN: 1930110936, 2003.

[71] H.H. Ammar, V. Cortellessa, A. Ibrahim, "Modeling resources in a UML-based simulative environment," *Proceeding of ACS/IEEE International Conference on Computer Systems and Applications*, pp. 405-410, June 2001.

[72] C. K. Chang and T.-H. Kim, "Distributed Systems Design Using Function-Class Decomposition with Aspects," *Proceedings of the 10th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, pp. 148-153, May 2004.

[73] T.-H. Kim and C. K. Chang, "Service-Oriented Design with Aspects (SODA)," *Proceedings of the 2005 Int'l Conference on Services Computing (SCC 2005)*, vol 1, pp. 319-322, 2005.

[74] L. Hillah, F. Kordon, L. Petrucci, and N. Treves, "Building an API for ISO/IEC 15909, based on model engineering techniques," *Workshop on the Petri Net Markup Language 2005 (PNML 05) - Towards an ISO/IEC Standard Transfer Syntax for Petri Nets*, Helsinki University of Technology, Finland, May 26, 2005.

[75] J. L. Peterson, "Petri Nets," *ACM Computing Surveys*, vol. 9, no. 3, 1977.

[76] R. Valette, "Analysis of Petri Nets by Stepwise Refinement," *Journal of Computer and System Sciences*, vol. 18, pp. 35-46, February 1979.

[77] World Wide Web Consortium (W3C) Recommendation, *Simple Object Access Protocol (SOAP) version 1.2*, June 2003, Available: http://www.w3.org/TR/soap/.

[78] OASIS Standards, *Universal Description, Discovery and Integration (UDDI)*, February 2005, Available: http://www.uddi.org/.

[79] World Wide Web Consortium (W3C) Note, *Web Service Description Language (WSDL) 1.1*, March 2001, Available: http://www.w3.org/TR/wsdl.

[80] World Wide Web Consortium (W3C) Member Submission, *Web Services Policy 1.2 - Framework (WS-Policy)*, April 2006, Available: http://www.w3.org/Submission/WS-Policy/.

[81] JDOM Project, *JDOM*, October 2006, Available: http://www.jdom.org/.

[82] S. Jones, "Toward an Acceptable Definition of Services," *IEEE Software*, pp. 87-93, May/June 2005.

[83] O. Zimmermann, P. Krogdahl, and C. Gee, "Elements of Service-Oriented Analysis and Design," June 2004, Available: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/.