MODEL-DRIVEN VARIABILITY MANAGEMENT IN CHOREOGRAPHY SPECIFICATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELMA SÜLOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

SEPTEMBER 2013

Approval of the thesis:

MODEL-DRIVEN VARIABILITY MANAGEMENT IN CHOREOGRAPHY SPECIFICATION

submitted by **SELMA SÜLOĞLU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Ali H. Doğru _____
Supervisor, **Computer Engineering Department, METU**

Assist. Prof. Dr. Bedir Tekinerdoğan _____
Co-supervisor, **Computer Engineering Dept., Bilkent Uni.**

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Department, METU _____

Prof. Dr. Ali H. Doğru
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _____

Assist. Prof. Dr. Selim Temizer
Computer Engineering Department, METU _____

Assist. Prof. Dr. Aykut Erdem
Computer Engineering Department, Hacettepe University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SELMA SÜLOĞLU

Signature :

ABSTRACT

MODEL-DRIVEN VARIABILITY MANAGEMENT IN CHOREOGRAPHY SPECIFICATION

Süloğlu, Selma

Ph.D., Department of Computer Engineering

Supervisor : Prof. Dr. Ali H. Doğru

Co-Supervisor : Assist. Prof. Dr. Bedir Tekinerdoğan

September 2013, 352 pages

In this thesis, model driven variability management in choreography model is introduced, which brings variability management and choreography specification together in one single model. Service Oriented Architecture (SOA) is a means of facilitating inner and inter-organizational computing which reveals a reusable architecture comprising service consumer, producer and broker. To achieve assembling and composition of services, orchestration and choreography concepts are utilized, which are two interrelated views of the system architecture. In the architectural level, orchestration and choreography models are tailored by variability specifications in order to deal with reuse challenge. Several approaches have been introduced to support variability in orchestration and choreography languages. Unfortunately, variability specifications are not explicitly addressed in current choreography languages and are not integrated with variable orchestration specifications. Specification of consistent variability binding and configuration of interacting services accordingly have not been considered in the choreography language level. Moreover, there is a lack of support to reuse existing choreographies. A metamodel and its realization, XChor language is presented and validated with regard to service variability needs and service interactions patterns. XChorS Tool is developed to facilitate pre and post analysis of models, configuration of models regarding variability bindings in a consistent way and transformation of models to existing languages. Verification of XChor models is enabled

and implemented by means of transforming to a model checking system, Featured Transition Systems. Lastly, variability management of assets and artifacts in software product lines with the help of XChor metamodel and language is explained. Case studies are provided for demonstration purposes.

Keywords: Service-Oriented Architecture, Choreography Model, Choreography Language, Variability Management, Model Driven, Software Product Lines

ÖZ

KOREOGRAFİ TANIMINDA MODEL TABANLI DEĞİŞKENLİK YÖNETİMİ

Süloğlu, Selma
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ali H. Doğru
Ortak Tez Yöneticisi : Yrd. Doç. Dr. Bedir Tekinerdoğan

Eylül 2013 , 352 sayfa

Bu tezde, değişkenlik yönetimini ve koreografi belirtimini bir modelde birleştiren, koreografi modelinde model odaklı değişkenlik yönetimi önerilmiştir. Servis Odaklı Mimari (SOM), servis kullanıcılarını, sağlayıcılarını ve arabulucuları içeren ve yeniden kullanılabilir bir mimari temelinde organizasyonların kendi içinde ve organizasyonlar arası birliktelikler gerektiren sistemlerin gerçeklenmesine olanak sağlar. Servisleri biraraya getirmek ve onların tümleştirilmesini sağlamak için SOM'da orkestrasyon ve koreografi kavramları kullanılır. Bu iki kavram birbirleriyle sıkı ilişkiler içerisinde olan sistem mimarisinin birbirleriyle ilişkili bakış açılarıdır. Mimari seviyede, yeniden kullanılabilirliği sağlamada yaşanan zorluklarla baş etmek için orkestrasyon ve koreografi değişkenlik tanımlamalarına göre uyarlanır. Değişkenliği orkestrasyon ve koreografi seviyesinde destekleyen birçok yaklaşım bulunmaktadır. Ancak şu anki koreografi dillerinde değişkenlik tanımlamalarına açık bir şekilde değinilmemiş olup değişken orkestrastyonlarla bütünleştirilmesi yapılmamaktadır. Birbirleriyle ilişkili olan servislerin tutarlı bir şekilde değişkenlik ilişkilendirilmesi ve bu ilişkilendirmeye göre konfigüre edilmesi koreografi dili seviyesinde ele alınmamıştır. Ayrıca, varolan değişken koreografilerin kullanımı konusunda dil seviyesinde destek verilmemektedir. Bir metamodel ve gerçekleştirimi olan XChor dili anlatılmış olup XChor dili servis değişkenlik gereksinimlerine ve servis etkileşim örgülerine göre geçerlenmiştir. XChor modellerini ön ve son analizlerini yapmak, değişkenliği tutarlı bir şekilde ilişikilendirerek modelleri konfigüre etmek ve varolan dillere dönüştürmek için XC-

horS aracı geliştirilmiştir. XChor modellerinin Özellikli Geçiş Sistem modellerine dönüştürülme kuralları tanımlanarak ve gerçeklenerek XChor modellerinin doğrulanması adım adım anlatılmıştır. Son olarak yazılım üretim bantlarında varlık ve yapı birimlerinde değişkenlik yönetiminin XChor metamodeli ve dili ile nasıl yapılacağı gösterilmiştir. Durum senaryoları gösterim amaçlı sunulmuştur.

Anahtar Kelimeler: Servis Odaklı Mimari, Koreografi Modeli, Koreografi Dili, Değişkenlik Yönetemi, Model Tabanlı, Yazılım Üretim Bantları

To my family and the ones who are now reading this page

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Ali H. Doğru and co-supervisor Associate Professor Bedir Tekinerdoğan for their constant support, guidance and friendship. It was a great honor to work with them for the last seven years and our cooperation influenced my academical and world view highly. I also would like to thank Associate Professor Halit Oğuztüzün for his support and guidance. He also motivated and infuenced me highly in scientific context.

And there are a lot of people that were with me in these seven years. They defined me, they made me who I am, they are true owners of this work. It is not possible to write down why each of them is important to me and this work, because it will take more space than the work itself. I am very grateful to all people I know during my research assistantship in METU-CENG, they changed me deeply; my vision towards life, happiness and friendship. I am very luck to have them all. So I'll just give names of some of them; Hande Çelikkanat, Ömer Nebil Yaveroğlu, Burçin Sapaz, Sinan Kalkan, Nilgün Dağ, Utku Erdoğdu, Onur Deniz, Özgür Kaya, Gökdeniz Karadağ, Can Eroğul, Ali Anıl Sınacı, Erdal Sivri, Hilal Kılıç, and Cengiz Toğay. I would also like to thank you Meltem Turhan Yöndem for always listening and leading me from the start of my masters. I am greatly indebted for Utku Erdoğdu's endless time and effort preparing me last versions of latex. This work is also supported by TÜBITAK-BIDEB National Graduate Scholarship Programme for PhD (2211).

Lastly, sincerest thanks to each of my family members for supporting and believing in me all the way through my academic life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ASM	Abstract State Machines
BPEL	Business Process Execution Language
BPEL4WS	Web Service Business Process Execution Language
BPML	Business Process Markup Language
BPMN	Business Process Markup Notation
CA	Constraint Automata
CBFM	Cardinality-Based Feature Modeling
CBS	Coordination Behavioral Structure
ConIPF	Configuration in Industrial Product Families
COVAMOF	ConIPF Variability Modelling Framework
CVL	Common Variability Language
CVP	Configuration Variation Point
ebXML	eXtensible Markup Language
ESB	Enterprise Service Bus
featureRSEB	feature Reuse-Driven Software Engineering Business
FODA	Feature-Oriented Domain Analysis
Forfamel	Feature Modeling For Software Product Families
FTS	Featured Transition System
GML	Generalized Markup Language
HTML	Hyper Text Markup Language
IETF	Internet Engineering Task Force
MDE	Model Driven Engineering
OASIS	Organization for the Advancement of Structured Information Standards
OVM	Orthagonal Variability Model
OO	Object Orientated
OMG	Object Management Group(OMG)
RAS	Reusable Asset Specification
RequiLine	A Requirements Engineering Tool for Software Product Lines

SaaS	Software as a Service
SGML	Standard Generalized Markup Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPL	Software Product Lines
XML-RPC	XML-Remote Procedure Call
TVL	Text-based Variability Language
UDDI	Universal Description, Descovery and Integration
UML	Unified Modeling Language
VP	Variation Point
VSL	Variation Specification Language
WIDL	Web Interface Definition Language
WS	Web Service
WSCI	Web Service Choreography Interface
WS-CDL	Web Services Choreography Description Language
WSCG	Web Services Chroegraphy Group
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WSMO	Web Service Modelig Ontolgy
WWW	World Wide Web

CHAPTER 1

INTRODUCTION

Complexity and change are major challenges that software development industry have been facing. Dealing with change is an inevitable and essential concern, inherent in software development nature. During software life cycle, software evolves in response to change. This evolution which comprises alteration in software structure and behavior directly reflects to its design and implementation accordingly. Changes in business needs and user demands cause alteration in problem domain which result from requirement inconsistency. On the other hand, rapid and unpredictable technology evolution, new technologies and new trends, and other external factors require alteration in the solution domain. These can be overcome with iterative or agile development strategies, adaptive design techniques and refactoring. Another significant consideration is dealing with complexity. Software development industry, while being the key driver of modern economy has an unacceptably high level of failures, caused to a large extent by high complexity of software systems. Software development industry has significant problems with managing this complexity, with raising the level of abstractions and narrowing the abstraction gap between problem and solution domains, keeping track of changes, and reusing knowledge from previous projects. The main barrier in overcoming these problems is the lack of widely accepted and easy to apply mechanisms for expressing and reusing coherent solutions to problems formulated as user requirements. Software reuse is the use of existing assets and reapplication of various kinds of knowledge in some form within the software product development process. The reused knowledge includes such things as domain knowledge, technology expertise and development experience. More than just code, assets are products and by-products of the software development life cycle

and include software components, test suites, designs and documentation[115, 70].

Several organizations develop, share and reuse business processes by establishing collaboration with other organizations in order to fulfill different stakeholder needs. Being agile is an important challenge in business process integration context which requires a dynamic environment. In this respect, Service-Oriented Architecture (SOA) is a promising approach to realize such environments by designing and developing distributed systems[60]. SOA aims to facilitate reuse of services and incorporates service consumers and service providers. A service is self-contained, and can be independently deployed in a distributed component. Building enterprise solutions to realize business processes typically requires the composition of multiple existing enterprise services. Composite services can be further recursively composed with other services to derive higher level solutions. Two different types of service compositions are defined.

Service choreography where the interaction protocol between several partner services is defined from a global perspective without a central mechanism.

Service orchestration where the interaction logic is specified from the local point of view of one single participant, called the orchestrator.

Many approaches have been proposed to tackle with complexity and change via variability management mechanisms, middleware and reconfiguration solutions, dynamic adaptations, and rule-based approaches. Within these approaches, using variability management mechanisms in different granularity levels, namely choreography, orchestration and atomic services, enables reuse in services. Assuming that all granularity levels can be treated as services, variability can come from (i) their interfaces (functions and parameters), (ii) connectors (the way they interact) and (iii) composition (the way they are gathered in order to achieve a goal). Interface variability requires a configuration mechanism specifying when and how to change its functions and parameters. Connector variability needs a relation mechanism to indicate when and which connector is used between two services. Composition variability necessitates a tailoring mechanism to define in which order and how services are interacting with each other. Services offer different functionalities regarding their variability

bindings. Therefore, it is the composition's responsibility to provide a consistent variability binding between interacting services. This requires a mechanism to establish variability associations which determines when and how interacting services bind to specific variants. In other words, composition is responsible for handling consistent variability binding of interacting services and providing a configuration infrastructure to reveal seamless integration of services. To cope with such challenges several approaches have been introduced. However, explicit introduction of variability integrated with choreography languages is not addressed. Specification of consistent variability binding and configuration of interacting services are not considered in the choreography language level. Moreover, there is a lack of support to reuse existing choreographies. All in all, reusing existing services and service architectures in an efficient and systematic way is a difficult task.

1.1 Background

Ad-hoc approaches to reuse in software development lead to the process more expensive and more time consuming because of building software from scratch. To realize the benefits of reuse, a mature approach must be adopted called systematic reuse, increasing the returns on investments in production assets, implementation assets through economies of scale and scope.[69] Three basic strategies are pointed out for leveraging systematic reuse: working faster via tools to automate the labor-intensive tasks, working smarter with process improvement, and working less via reuse of software artifacts. An extensive analysis is made on the question that which strategy will produce the highest payoff and concluded that "working less" is more valuable three times than "working smarter" and six times than "working faster"[38]. Therefore, reusability is a key to improving productivity in the software development area.[116, 38, 89, 31]

Service-Oriented Architecture reuses services, service descriptions and architecture, as well as providing flexibility and adaptation with dynamic discovery mechanisms and late binding of services to improve productivity. Services are composed in order to achieve a common goal as a reusable asset. Orchestration, Choreography, Coordination and Assembly are four types of composition models followed by SOA. Within

these, orchestration and choreography are tightly related concepts, as interrelated views of architecture. Orchestration, as a central mechanism, provides coordination between related services; on the other hand choreography is a non-executable specification of global view on interacted services without a central mechanism. Tightly relation causes orchestration and choreography view consistency with each other while constituting service architecture[60, 59, 34]. Several standardizations, approaches, graphical and development tools defined for orchestration, choreography and coordination languages exist. Examples for orchestration are WS-BPEL-Executable Processes (BPEL, BPEL4WS)[102], BPML[58], BPMN[104], for choreography WS-BPEL-Abstract Processes[102], WSCI[124], WS-CDL[137], and BPMN.

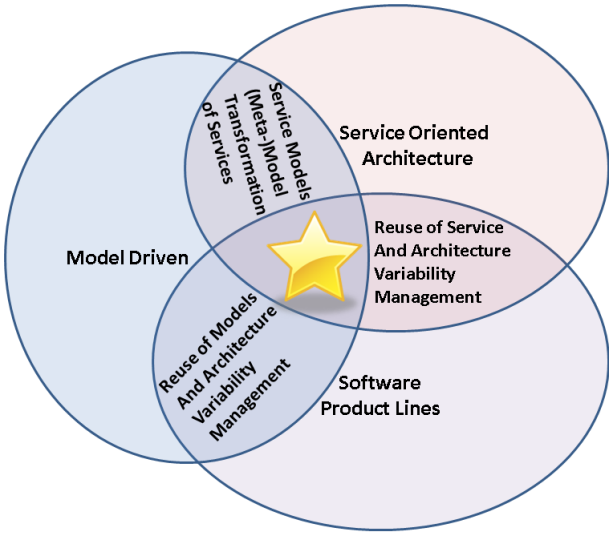

Figure 1.1: Relations Between SOA, Model Driven and SPL Approaches.

As expected, flexibility and reuse are important features of SOA to tackle dynamic business environments and process flows. Dynamicity and change requires management of variability, rather in a systematic way. Software Product Lines (SPL) approach offers variability management environment by sharing a common and managed set of features, reusing a common set of core assets in a prescribed way. Possible variability types in SOA are variation in service function, in required parameters, protocols and composition of services. These types of variability can be handled within specified variability model in architectural levels namely orchestration and choreography. Several variability models are defined in SPL comprising specification and modelling of variation points and types, related variants, constraints on them, rela-

tion with other artifacts such as features, realizations. CVL, VSL, ConIPF, CBFM, COVAMOF, and OVM are some examples of variability models. Not only definition of variability, but also management and its impact on other artifacts (such as requirements, features, and code) should be taken into account. In order to provide variability consistency between orchestration and choreography, Model Driven Engineering (MDE) techniques such as models, meta-modeling and transformation can be utilized. As models play an important role, they are not the whole solution if variability in choreography is applied to SPLs. Scaling up to higher levels of productivity will require the ability to rapidly configure, adapt and assemble independently developed, self-describing, location independent services to produce families of similar but distinct systems. Therefore, a Domain Specific Language and related tools are required.

Relationship between SOA, MDE and SPL approaches in order to improve productivity while tackling complexity and change is depicted in Figure 1.1. The intersection of the approaches represents the techniques while these are applied together. SOA and SPL approaches enable reuse of services, architectures and managing variability in SOA systems in a systematic way. While using MDE and SPL together, reuse of models and architectures is achieved and variability in models is managed efficiently. Service Models are defined and model and meta-model transformation of services can be applied as model-to-model or model-to-text style while MDE and SOA approaches are used together. Therefore, the intersection of the three brings more benefits ranging from modeling variability and transforming at the meta-model level to managing variability in a systematic way.

1.2 Problem Statement

Service Oriented Architecture (SOA) is a means of facilitating inner and inter-organiza tional computing which is a way of developing distributed and autonomous systems where the components of the system are services. As in all systems, SOA has a structure and a dynamic nature. While structure comprises services, their functional and non-functional properties, interfaces and relation between other services, dynamicity includes behavior of services while achieving a goal together and rules on these in-

teractions. A bidirectional effect between structure and dynamicity represents a tight relationship. Any effect on one of these reflects to other, as stated in Figure 1.2. Therefore, when variability is added to the system in order to tackle complexity and change, it affects system structure and dynamicity and should be managed in both views.

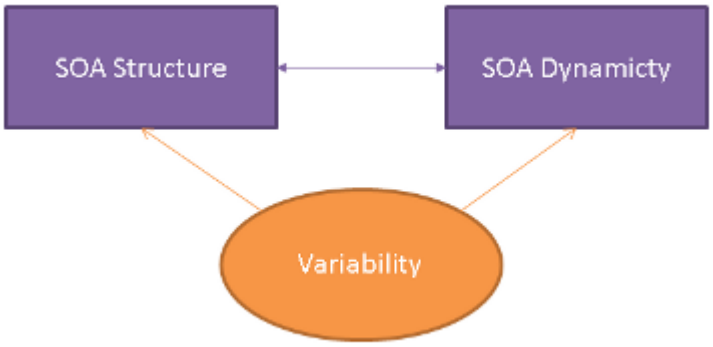

Figure 1.2: Relation with SOA Structure, Dynamicty and Effet of Variability.

Orchestration and choreography are views for service architecture defining a central mechanism and a non-central representation respectively. SOA architectures addressing variability efficiently and in a systematic way should handle variability management in orchestration and choreography in a consistent manner as depicted in Figure 1.3. As each view has different requirements to define and express variability to outer SOA space, the variability model should fulfill the needs of each and relate variability of each other.

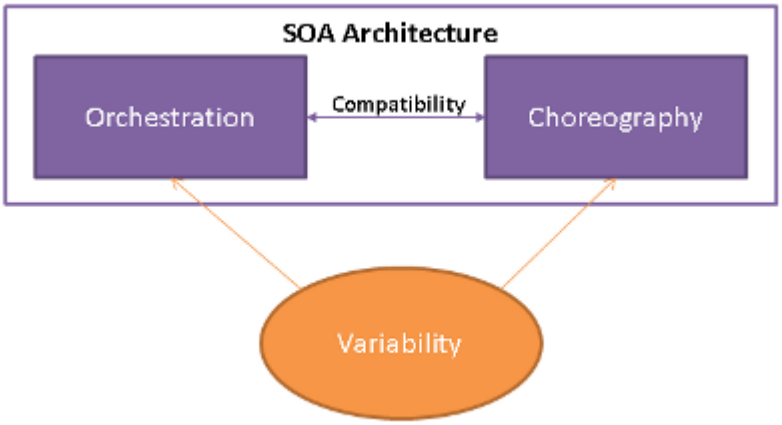

Figure 1.3: Orchestration and Choreography Relation and Effect of Variability.

There are existing approaches proposed to support variability which can be categorized in different groups: Adaptation managers/modules, Enterprise Service Bus (ESB) like middleware and reconfiguration of components, rule-based systems and variable modeling and variability supported languages. In these approaches variability is mapped from problem domain artifacts - features, functional and non-functional requirements, UML diagrams and components- to solution domain artifacts - decision models, business process templates, orchestration and/or choreography languages, components, adaptation rules, plans and so on. In the literature, several approaches and languages are proposed to define and manage variability. However, variability in choreography models compatible with variability in orchestration models is not handled. Therefore, specification of variability in choreography consistent with orchestration cannot be achieved and a global view of variability cannot be gathered which provides a coarse grained variability model. Therefore, a variability model is needed to manage variability in choreography as variability depiction globally and to map the variability to orchestration and services as variability depiction locally.

From SPL perspective, SPL specifies a reference architecture for possible products with variabilities, while choreography represents a product configuration in SOA. Therefore in the context of SOA, a reference architecture should include possible choreographies with variabilities. Besides, the success of a variability supported SOA is largely dependent on effective variability management throughout the development life cycle, in our case architecture. Therefore, there should be a variability model defined for possible choreographies. However, there is no choreography model with variability support which depicts possible products while consistent with interacted orchestrations.

Orchestration and choreography concepts in SOA can be related with artifacts and assets respectively. Seamless collaboration between artifacts to behave according to required product features should be achieved via a systematic variability management in choreography and orchestration in a consistent manner.

1.3 Approach

The motivation is to response to changing requirements and fulfilling diverse range of user demands fast and easily in service-oriented environments. One of the main challenges is to cope with change with little effort in a short time. The main idea is reusing existing service architectures; choreographies and orchestrations via explicit variability definition and management. That is, establishing a configurable service architecture realized through choreography specifications while taking into account variability needs of services is the ultimate target.

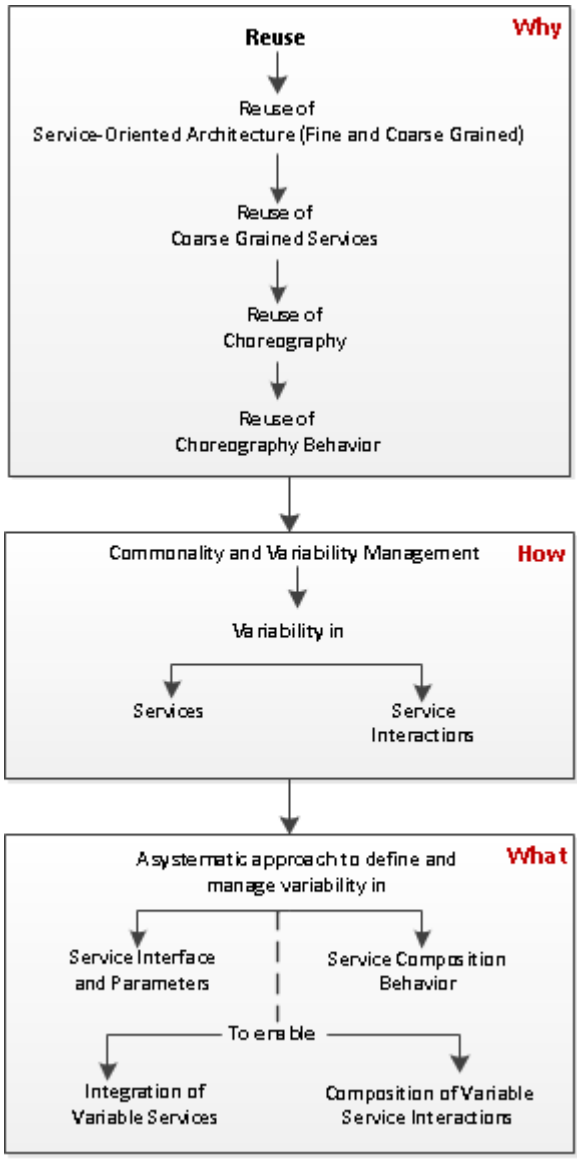

Figure 1.4: The approach answering why, how and what questions.

Our main purpose is the reuse of service-oriented architectures, namely the coarse grained ones which are realized as choreographies. Reuse of choreographies brings about reuse of choreography behavior which describes the way how services are interacting with each other.

In Figure 1.4 *Why* part is dedicated to the purpose of our approach. The way how to reach our goal is by managing commonality and variability of service architectures, defining variability in services and service interactions depicted in *How* part. To put into practice this kind of service systems, a metamodel and its realization XChor is specified to define variability in service interface (in functions and parameters) and service composition behavior. Here service concept comprises choreography, orchestration and atomic services. In our approach service composition is realized by choreography specifications. The systematic approach enables to integrate variable services and composite variable service interactions.

In this thesis, firstly SOA ecosystem constituents, service composition, orchestration and choreography views on composition are investigated. Existing orchestration and choreography languages are examined and compared according to basic functionalities, composition mechanism, tool support, variability support, and explicit variability specification. It is found that variability support of choreography languages and consistency of variability between choreography and related orchestrations are not fully covered. For addressing variability issue, SPL approach which defines a way to effectively manage variability and enable systematic reuse is analyzed and existing variability models affecting architecture are compared. Variability notion in SPL is discussed and required variability types are specified in the context of variability needs of service-oriented systems.

Two example cases describing the necessity of variability in choreography languages is presented with UML sequence diagrams. Requirements for managing variability in all levels, respectively choreography, orchestration and atomic services are revealed. Existing variability models and variability supporting languages are a starting point to define the way to integrate variability model with choreography constructs. Although there exist different orchestration and choreography models, a few supports variability in metamodel and language level. Therefore, required interface types, variation

depiction and the mechanism to configure services and their composition are investigated. In consideration of required variability mechanism and service composition needs, a meta-model and its realization XChor is defined which enable to form a variable SOA environment in a prescribed way. The comparison of existing languages with XChor language is presented and explained. The metamodel and XChor language are validated against service variability types and service interaction patterns.

Transformations from XChor language to existing choreography and orchestration languages are defined with rules. Verification of XChor language is enabled by transforming XChor models to one of the model checking approaches, namely Featured Transition Systems. XChor is applied to SPL domain to manage asset and artifact variability. Lastly, the thesis is finished by discussions, short and long term future works. Xtext specification of XChor Language, case study implementations in XChor and XChorS tool code reference manual are presented as appendices.

1.4 Contribution

As a result of this thesis study, (1) summary and comparison of existing orchestration and choreography languages are revealed, (2) a variability metamodel for choreography and its realization XChor language are defined,(3) XChorS tool facilitating analysis, configuration, verification and transformation to existing languages is implemented, (4) transformation rules are described and metamodel and XChor language is applied to SPL domain. Moreover, in order to manage variability in choreography model, following tools are developed:

- An analysis tool to analyse, sketch, define and modify variability specifications.
- A validation tool which analyses specified variabilities and checks consistency.
- A configuration tool to form ultimate service-oriented application according to user specified bindings.
- A transformation tool to convert XChor models to appropriate existing choreography and orchestration language models.

- A verification tool to transform XChor models to Featured Transition System models.
- A design tool which enables to specify domain choreographies and services with their variability specifications and to derive application choreographies and services regarding user selected features.

User requirement dynamicity reflects to architecture, in our case orchestration and choreography. Realizing reflection from requirements to architecture requires a relation mechanism between them. Requirements are mapped into services and variability points. However, this mapping mechanism is out of scope of this thesis. Main contribution of this thesis stands for defining, modifying, and managing variability scattered over orchestration and choreography, which have already mapped to requirements.

1.5 Outline Of Thesis

Chapter 2 represents existing orchestration and choreography languages, analysis of variability support in SOA domain. Then obstacles are depicted and problems are stated in detail. In Chapter 3 XChor metamodel and language facilitating variability in choreography language is specified and validation of XChor is explained in detail. Chapter 5 introduces transformation from XChor models to existing choreography and orchestration languages, namely BPEL4Chor and VxBPEL. Chapter 4 explains verification of XChor metamodel via transformation from XChor models to Feature Transition System models. In Chapter 6, variability management approach in choreography for SOA domain is applied to SPL domain. Variability management in software product lines with XChor metamodel and language is depicted in detail. Thesis concludes with Chapter 7. Dependency among chapters are depicted in Figure 1.5. The arrows represent information dependency where the source chapter needs specifications done in destination chapter.

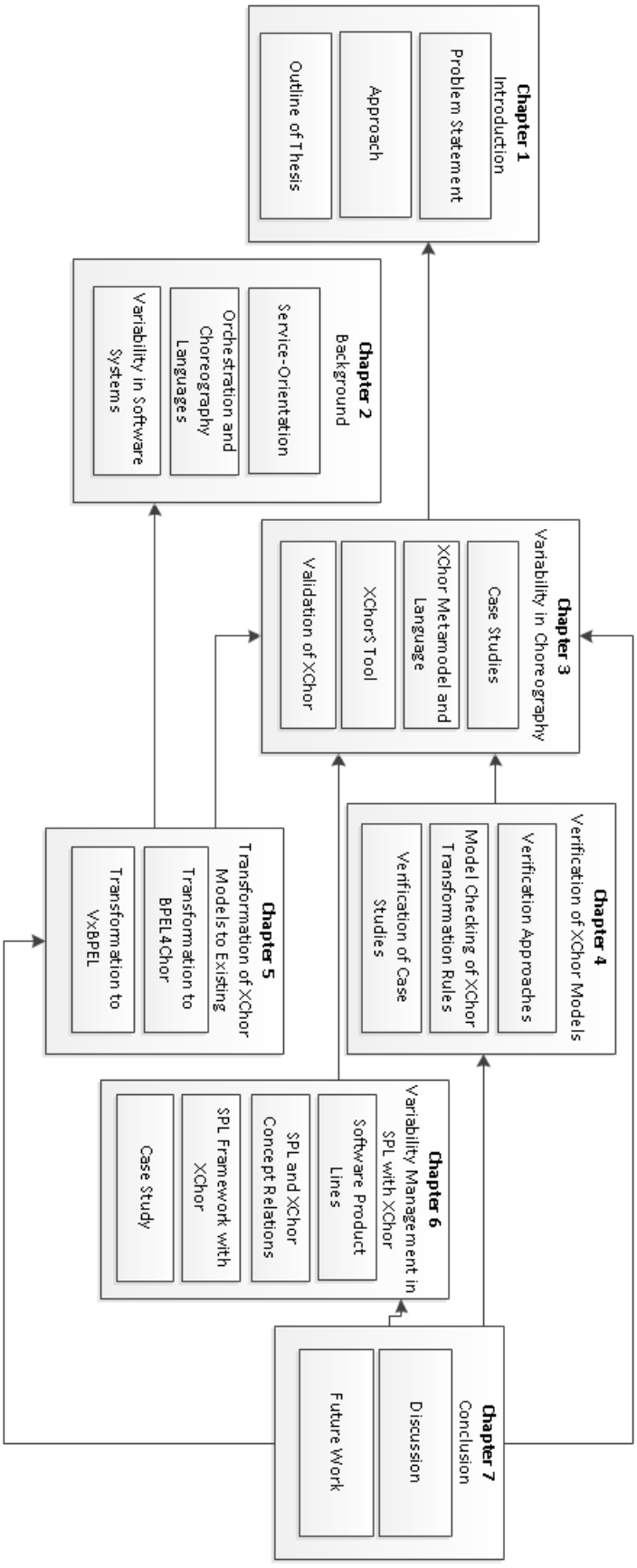

Figure 1.5: Chapter Content Dependency.

CHAPTER 2

BACKGROUND

This chapter elaborates first the history of web standards, organizations and paradigms, along with a brief introduction of Service Oriented Architecture (SOA) and composition in SOA. Then, the details of our systematic literature review is given with respect to variability modeling in orchestration and choreography languages and selected orchestration and choreography languages are explained briefly. After analyzing variability notion in software systems especially in SOA, variation support in existing approaches are compared and existing variability models addressing architecture and product derivation are listed. In order to elaborate capabilities of existing orchestration and choreography languages, a Comparison Framework is introduces with three components in detail. Selected languages and variability models are compared and discussed accordingly. Lastly, existing problems are stated according to the results of comparison framework.

2.1 Chronological History of Web Standards, Organizations and Paradigms

When we look at the development of computer science, we can see that the need to overcome data integration and representation problem always present. The chronological history of introduced web standards, concepts, paradigms and organizations are depicted as a time line in Figure 2.1. The roots of the work stand on the invention and development of Generalized Markup Language (GML) by IBM in 1969. The usage of (GML) in text processing was in later 1973. Then, Standard Generalized Markup Language (SGML) is followed GML, a ISO Standard[2].

Linda[67], a model of coordination and communication among parallel processes operating on an ordered sequence, is implemented as a a coordination language. The language is developed originally for the SBN network computer in 1982 and was used in coordination of processes/tasks later.

In 1989/90 World Wide Web (WWW) was founded by Tim Burners-Lee at CERN integration of PCs, servers, applications over internet[3] In early 90's Object Oriented (OO) approach was appeared even if it was invented in 60's.

In 1991, Object Management Group (OMG) introduced Common Object Request Broker Architecture[103]. In the same year, Tim Burners-Lee is developed Hyper Text Markup Language (HTML)[136].

In 1993, freeness of WWW to everyone and formally definition of first draft of HTML (SGML was used to define) by IEFT (Internet Engineering Task Force) affected graphical and textual browser occurrence, such as Viola, Mozaic.

In 1994, WWW was founded at MIT and supported by DARPA & EU Commission and W3C organization was founded for standardization.

HTML 2.0 was released by IEFT, HTML Working Group in 1995 and since 1996 the work done for HTML was conducted by W3C[4] with input from other software vendors.

In 1996 first version of XML, extended from SGML, was developed by W3C[5] and Microsoft introduced COM/ DCOM[92]. Webmethods was submitted Web Interface Definition Language (WIDL) to W3C in 1997. WIDL was affected by ORB mechanism resided in CORBA. Simple Object Access Protocol (SOAP)[135] was first defined in 1998 whose predecessor was XML-RPC (XML-Remote Procedure Call). XML 1.0 W3C Recommendation was published in the same year.

Electronic Business using eXtensible Markup Language (ebXML)[8] was defined by Organization for the Advancement of Structured Information Standards (OASIS)[1] in 1999. In the same year, Microsoft first introduced the concept of Web services[7] and Indigo project was started[6].

HTML became ISO Standard in 2000, Universal Description, Descovery and Inte-

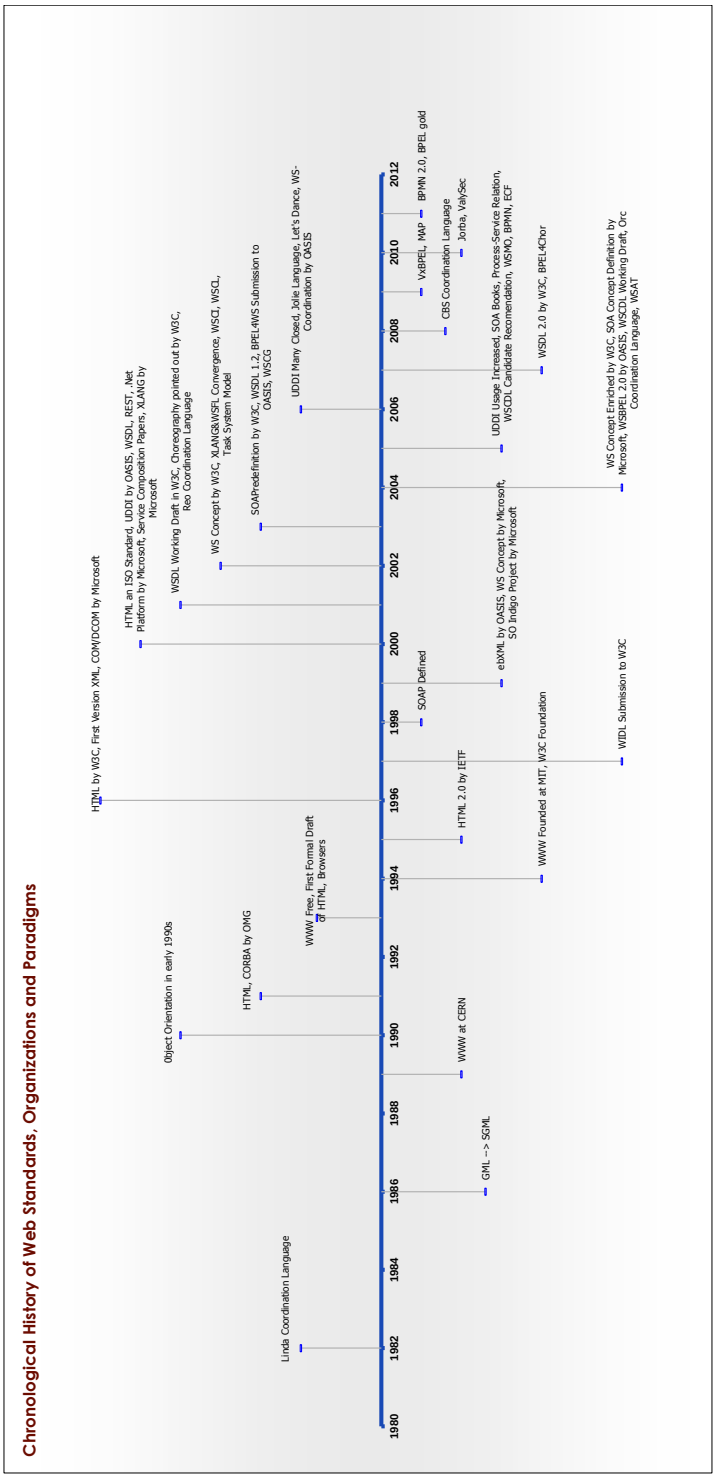

Figure 2.1: Chronological History of Web Standards, Organizations and Paradigms.

gration (UDDI)[9] was introduced by OASIS, Web Service Description Language (WSDL)[10] was specified by IBM, Microsoft and Ariba. Representational State Transfer (REST) was introduced and published in Fielding's dissertation[63]. At the same time Microsoft advertised .NET platform. Papers about service composition was seen in the literature[46, 45, 91]. For service orchestration, Web Services Flow Language (WSFL)[11] was defined by IBM, on the other hand XLANG[13] was introduced by Microsoft used in BizTalk. The two specifications are related with each other.

In 2001, working draft of WSDL at W3C was seen as version 1.1, and W3C pointed need for choreography[12]. A coordination language, Reo was proposed by Farhad Arbab at Centrum Wiskunde & Informatica[14].

In 2002, W3C Web Service Architecture Working Group defined Web Service concept as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL)"[18]. XLANG & WSFL specifications were worked for converge. For choreography consideration, Web Services Choreography Interface (WSCI) was defined by Sun, SAP, BEA, and Intalio. Concurrently Web Service Conversation Language (WSCL)[15] was specified by Hewlett Packard. Task System Model[88] aiming at process decomposition and process relations is proposed.

In 2003, SOAP was redefined and WSDL version 1.2 was published by W3C. BPEL4 WS was submitted to OASIS. Web Services Choreography Group (WSCG) organization founded at W3C.

In 2004, Web Service concept was enriched by W3C. Service Oriented Architecture (SOA) was first defined by Microsoft, Don Box (one contributors of SOAP specification) explaining fundamental tenets of Service Orientation[17]. XLANG & WSFL specifications were worked for converged to BPEL4WS which was submitted to OASIS, commonly known as BPEL. Web Service Choreography Language (WS-CDL) was Working Draft at W3C. Orc coordination language is defined[16].

In 2005, the utilization of UDDI was increased because of being open to use and WS-CDL was Candidate Recommendation of W3C. Published SOA books has been

increased and process service relation was started to point out. Web Service Modeling Ontolgy (WSMO) was proposed [109]. A framework for on-demand choreography deployment named as Executable Choreography Framework (ECF) is specified and an XML based language Executable Choreography Language (ECL) is introduced into which activity diagrams or WS-CDL specifications are translated [53]. Moreover, Choreography Language (CL) was introduced as a simple choreography language which provides a formal framework including declarative and conversational parts of choreography.

In 2006, many UDDIs were closed to use. WS-Coordination[21] was specified by OASIS in order to coordinate web services by registration and activation services. Jolie Language and interpretation engine for orchestration of services was proposed [20]. On the other hand, for choreography Let's Dance view for assembling services and the tool were introduced in [141, 140, 55].

WSDL 2.0 was published by W3C in 2007. Extracting three models for a choreography specification from BPEL orchestrations, BPEL4Chor is proposed[56]. In 2008, Coordination Behavioral Structure (CBS) was proposed to represent topology of interactions in a formal way within coordination approaches [142].

In 2009, Multiagent Protocols (MAP) Web service choreography language [32] is introduced for choreography modeling, verifying and enacting via simple process language along with its open-source framework. Supporting variability for Web services in BPEL, VxBPEL was proposed in 2009 and an analysis tool, ValySec was implemented in 2010 [80, 125]. As a domain specific language, Business Choreography Language (BCL), was proposed which is heavily influenced by UN/CEFACT Modeling Methodology (UMM).

Jorba laying over Jolie orchestration language is proposed to enable dynamicty in orchestrations [85]. Early 2011, BPMN 2.0[104] by OMG was avaliable and almost all BPEL engines supported new specification. $BPEL^{gold}$ was proposed as an extension to BPEL language[81]. Another language, eSML was introduced for modeling e-business collaborations whose formalism is based on Petri-nets. Recently, AB-WSCL, based on actor system theory, was proposed as a web service composition language for choreography purposes.

2.2 Definitions and Main Terminology

2.2.1 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is a means of facilitating inner and inter-organiza tional computing which is a way of developing distributed and autonomous systems where the constituents of the system are services. According to W3C's Web Service Glossary[18] in 2004, a service is "an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent". In this respect, web services technology is a way to realize the concept of services, and to model interrelations and interaction constraints among them. The service forms a contractual agreement between a provider and a consumer through its interface. The service interface reveals its functionalities to the consumers with their signature regardless of internal implementation details. Even services become prominent with their functionalities, non-functional properties are as important as functional ones. The service should also provide required quality of service such as performance, availability and reliability along with its functionalities. Moreover, service middleware concept is introduced in order to overcome service integration problems and quality conflicts. As a middleware, Enterprise Service Bus (ESB) is the facilitator of enabling implementation, deployment and management of SOA-based enterprise solutions, which is constructed to be an open standards based message backbone. The Figure 2.2 depicts service utilization where *service provider* publishes implemented service interfaces to service broker, *service consumer* searches and finds related web services from service broker, *service consumer* binds found services via agreed messaging protocols and the interaction starts.

Service consumers can be end users and other services. In the context of web services technology; UDDI is the service broker, WSDL is the provided services interface language and SOAP is the messaging protocol between consumer and provider.

SOA reuses services, service descriptions and relying architecture which represents static and dynamic behavior of the system through service interaction modeling. Moreover, it provides flexibility and adaptation with dynamic discovery mechanisms

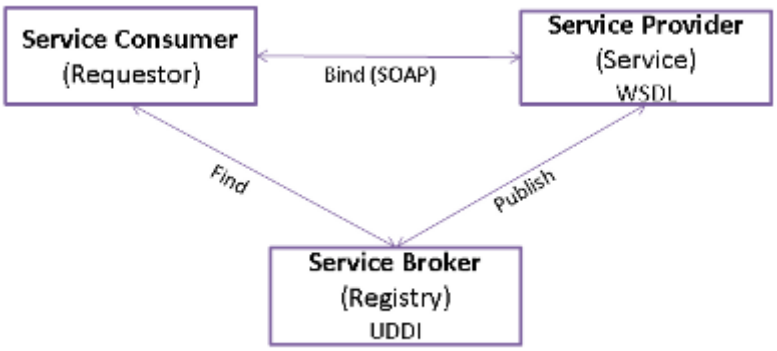

Figure 2.2: Service Oriented Architecture.

by means of published interfaces and enables late binding of services to improve productivity. Services can be categorized according to their granularity; atomic service, orchestration and choreography. An atomic service serves basic functionalities as an autonomous entity, whereas an orchestration is a service composition providing high level and complex functionalities. As a coarse-grained structure, a choreography includes atomic services and orchestrations to model globally defined interactions. As every system has a different service architecture specified by orchestration and/or choreography, service composition plays an important role to achieve flexibility to respond rapid changes. [105, 119, 120]

2.2.2 Composition in SOA

Concept development sequence in service world starts with single services and continues towards higher level concepts with to composite services, orchestration and choreography. After the need for choreography was pointed out in 2001, WSCG organization was founded by W3C in 2003 and launched WS Choreography model and WS-CDL in 2004. In the meantime, although the idea has origins in 60's, Open Innovation concept appeared in 2003. "Open innovation is a paradigm that assumes that firms can and should use external ideas and internal ideas, and internal and external paths to market, as the firms look to advance their technology. The boundaries between a firm and its environment have become more permeable; innovations can easily transfer inward and outward."[49] Therefore, by definition, choreography concept can be influenced by open innovation paradigm.

In service-orientation orchestration and choreography are two tightly interrelated concepts in service composition and their specifications should be consistent with each other. Orchestration of web services defines the ordering of service invocations and conditions. An orchestrator, as a central mechanism, is responsible for coordinating services which are interacting with each other at the message level. These interactions can be long lived, transactional and multi step. However, service choreography defines inter and intra collaboration including temporal and logical dependencies for each service without a central mechanism. Choreography tracks the sequence of messages between different parties. In Table 2.1, similarities and differences properties of orchestration and choreography are depicted.

Table 2.1: Similarities and differences of orchestration and choreography

	Orchestration	**Choreography**
Content	Depicts detailed information about composition such as connections and sequences	Depicts a restricted set of interface definitions with rules and constraints
View	Description of interactions as local view	Description of interactions as global view
Relation	Intra organizational	Inter and intra organizational
Depiction	a descriptive and formal format	a descriptive and formal format
Execution	Executable	Non-executable
Design Level	Low level	High level
Serve for	Composition	Composition

Although the role and importance of the architecture is mentioned in SOA, neither a defined architecture structure nor a road map to form a service architecture is introduced. In fact, orchestration and choreography constitute the architecture of the system. Therefore, choreography can be thought as a view of service architecture defining (i) components as participants, (ii) their properties as their interface definitions and (iii) relations between them, constraints on them, and control flow. Choreography and orchestration can also be taken as composition viewpoints. Choreography is seemed as a global view, a behavioral interface as a view of choreography from one participant, whereas orchestration is handled as an internal view of the participant. The relations between them are that choreography can be used as generating

orchestration skeleton and analyzing contradictions between choreography and orchestration, making out behavioral interfaces.[26] There have been several languages and graphical notations to model service orchestrations and choreographies.

2.3 Systematic Literature Review

To investigate the way how orchestration and choreography languages address variability, we have conducted an extensive search of papers in the literature. The guidelines described by Kitchenham [79] is used to develop the systematic literature review regarding variability modeling in orchestration and choreography languages. Research questions and the strategy in order to select related studies are given as follows.

Research Questions are derived an listed based on our research objectives:

RQ1 What are the current languages for modeling orchestration and choreography?

RQ2 What are the main characteristics, common and variant features among different orchestration and choreography languages?

RQ3 What are the research challenges and needs in the context of variability modeling of services orchestrations and choreography in language level?

RQ4 Which types of variability are supported by current orchestration and choreography languages?

Our search scope included the papers that were published over the period between 2000 and 2013, as service composition concept and related studies appeared after 2000. We searched for full papers in selected venues that publish high quality papers. We used the following search databases: IEEE Xplore, ACM Digital Library, Science Direct, Springer Link and ISI Web of Knowledge. Table I represents the numbers of studuies searched for each source. Our targeted search items were journal papers, conference papers, workshop papers.

To search the selected databases we used both manual and automatic search. Automatic search is realized through entering search strings on the search engines of the

electronic data source. Manual search is realized through manually browsing industrial standards introduced by standardization bodies, World Wide Web-W3C, OASIS, and OMG. The manual searches appeared to be quite useful since such industrial wide used languages are important for practitioners.

Table 2.2: Publication Sources Searched

Source	**Number of Included Studies After Applying Search Query**	**Number of Included Studies After Exclusion Criterion**
IEEE Xplore	132	9
ACM Digital Library	31	3
Springer Link	94	4
Science Direct	142	3
ISI Web of Knowledge	98	0
Total	497	19

A set of key terms are used to reveal related studies which are "orchestration language", "choreography language", "variable service composition", "business process language". These terms are combined with an OR operator with "service behavior modeling", "variable", and "variability" and with a NOT operator with "analysis", "verification", "test" and "framework" in order to scope the search relevant.

The adopted search string was as follows: ("orchestration language" OR "choreography language" OR "variable service composition" OR "business process language") AND ("web service" OR "service behavior modeling") NOT ("analysis" AND "verification" AND "test" AND "framework")

Table 2.3: Languages Introduced by Standard Bodies

Standard Organizations	**Number of Languages**	**Languages**
W3C	WS-CDL, WSMO	2
OASIS	ebXML, WSCI, BPEL, WS-Coordination	4
OMG	BPMN	1
Total		7

The result of the overall search process after applying the search queries and the manual search is shown in the second column of Table 2.2. As it can be seen from the table we could identify 493 papers at this stage of the search process.

Selection of relevant studies is determined regarding to the title of the paper, its abstract, its introduction and conclusion sections, and the whole body. Relevant papers are selected among

- peer-reviewed papers from conferences and journals
- papers written in English, with full text available
- papers that propose new orchestration and/or choreography language rather than presenting case studies, reviews and the application of existing techniques

Purely theoretical papers or the ones having empirical validation are excluded. According to our best knowledge, there was no secondary study related to variability modeling in orchestration and choreography languages. After applying the selection criteria 19 papers of the 497 papers remained. We have finally selected 11 orchestration languages and 17 choreography languages with the ones which standard organizations are introduced.

2.3.0.1 Orchestration Languages

There have been orchestration languages and notations introduced since service composition concept took in place in service-orientation. Although they target the same goal of gathering interacting services to seamlessly work in an order, they have different characteristics.

BPEL is a de-facto standard which specifies executable language syntax and operational semantics for implementing business processes through web services. BPEL comes with a wide range of constructs covering from basic send and receive actions to structural ones; conditional behaviors, sequences, repetitive actions, and selective actions. Defining variables, assigning data and specifying expressions through data handling, and interacting service definitions through

partner link definitions are specified. BPEL has two interrelated sides; executable and abstract. An executable BPEL process expresses all detailed information about business process to be executed directly. On the other hand, an abstract BPEL process definition indicates observable behavior from the point of other orchestration and services. SOAP is the underlying technology for messaging and WSDL is used for service interface specification. BPEL enables to define participants, roles, partner links, variables and flow and ordering constraints. An orchestration gathers atomic services and other orchestrations where other orchestrations are exposed as services with their service interfaces. BPEL focuses specifying processes from a single organization point of view, that is BPEL processes have a local view, it only knows the interacting services regardless of system's global behavior.

BPMN is a set of graphical notations in order to model intra and inter organizational interactions by defining control flow constructs; sequence and message flows. Therefore, both orchestration and choreography of services can be expressed. BPMN 2.0 was released in early 2011 by OMG and almost all modeling tools supports this notation. Five basic categories of elements are flow objects, data, connecting objects, swimlanes and artifacts. As flow elements, events, activities and gateways are graphical elements so as to specify business process behavior which are connected by sequence flows, message flows, associations and data associations. Processed data is represented by data objects, data inputs, data outputs and data stored. Grouping these elements can be achievable by pools and lanes. Artifacts are additional information sources for business processes. Although BPMN covers composition specifications, it lacks formal semantics which leaves formal verification of the process undone. In order to overcome this problem, there are approaches mapping BPMN to other languages which holds formal semantics such as BPEL, State Machines, and Petri Nets.

VxBPEL, as an extension to BPEL, provides a variable orchestration specification based on the COVAMOF model. It enables variability definition in service, service parameters, and service composition by using specific variation related constructs. Variation points and related variants are scattered over variable or-

chestrations encapsulating related BPEL specifications. Variability model comprising a subset of COVAMOF model, includes not only definition of variation point (VP) and, related variants, but also realization relations and mapping of variation to service orchestration. Realization relations provide a high level understanding for configuration purposes while hiding details of how low level bindings are done. The approach, in order to use existing BPEL engine, adapts ActiveBPEL and specifies a configuration file in which variation point determinations are specified. VxBPEL Constructs in XML notation in Table 2.4.

Jolie introduced in [20, 94] is an orchestration language providing a C like syntax having an available interpreter and an execution engine, based on the formal orchestration process calculus. It more provides an easy to use environment for programmers through its C/Java like structure instead of XML documents. For instance, except expression and condition syntaxes are similar to those in C. The language covers specifying shared memory for a location, an operation, a variable or a link through identifiers, a predefined program structure by the Jolie grammar and messaging through socket-based communications. The model defines two types of operations; input and output. Input operations specifies the points other orchestrators can access and output operations are used to depict invoking other orchestration's input operations. Input operations support two types of interaction; one-way and request-response. Program control flow, operation, synchronizing, console input/output statements are introduced in detail.

Jorba is a rule-based approach for dynamic adaptation, implemented on top of the Jolie language. Separation between the application and the adaptation specification constitutes the core of the approach. Adaptation hooks are defined to indicate information on structure and behavior of application parts. Jorba defines adaptation interfaces specifying function replacements whenever a change in service interface and parameter is required. Adapter manager checks environment condition changes and user needs. Then it apply a set of adaptation rules if required and reconfigures the application by means of adaptation hooks. The prototype just considers "on activity enter" as an approach for checking rule applicability, and sequential order of rules. Different moments are identified

Table 2.4: VxBPEL Language Constructs

Definition	VxBPEL Specification

Variation Point	`<vxbpel:VariationPoint name = "name">`
	`...`
	`</vxbpel:VariationPoint>`

Variant List	`<vxbpel:Variants>`
	`...a list of variants...`
	`</vxbpel:Variants>`

Variant	`<vxbpel:Variant name = "name">`
	`...`
	`</vxbpel:Variant>`

Inline BPEL Code	`<vxbpel:VPBpelCode>`
	`...BPEL Code...`
	`</vxbpel:VPBpelCode>`

Realization Relation	`<ConfigurableVariationPoints>`
	`<ConfigurableVariationPoint>`
	`<Name> </Name>`
	`<Rationale> </Rationale>`
	`<Variants>`
	`<Variant>`
	`<RequiredConfiguration>`
	`<VPChoices>`
	`<VPChoice>`
	`</VPChoice>`
	`</VPChoices>`
	`</RequiredConfiguration>`
	`<Variant>`
	`<Variants>`
	`</ConfigurableVariationPoint>`
	`</ConfigurableVariationPoints>`

when application rules can be checked. Moreover, rules can be applied in different orders, such as following some priority, can be applied in sequence. The possibility that an already checked rule may become applicable because of such a change is not considered.

WSMO provides the conceptual underpinning and a formal language for semantically describing all relevant aspects of web services. The aim is providing automation of discovering, combining and invoking services over the network via four basic elements, namely ontologies, web services, goals and mediators. Ontologies, central enabling technology for semantic web, enables data modeling which includes resource descriptors and interchanged data. Web services defines computational entities comprising capabilities, interfaces and internal working of the service. When capabilities describes functionality, one or more interfaces are used to define orchestration and choreography. The orchestration depicts the coordination of web services to achieve its capability. Goals representing user desires, model the user view in the web service usage process. WSMO comprises mediators to resolve incompatibilities on data, protocol and process level.[62]

BPML, Business Process Modeling Language, provides modeling of business processes in which four basic entities are addressed, namely process, activities, data and control. BPML utilizes XSD to represent data shared among business infrastructures. Value based, state Based, time based and cycle based control flows can be specified inside and among business processes. Activities can be nested as in sequence or in parallel and can be repeated with use of foreachs and while structures. Exception handling mechanism, coordinated and extended transactions are supported by BPML.[130]

PML, is a high level and simple process modeling language which facilitates to express models at abstract and concrete specification levels. The language follows simplicity, flexibility, expressiveness and enactability goals. PML facilitates to model activities as actions, to define pre and post conditions of actions, to specify control flow with the help of sequence, iteration, selection and branch. For user-defined specifications, PML employs a language construct "qualifier" to represent characteristics of a resource.[29]

RBXPDL, Role-Based XML Process Description Language, provides a way to model business processes taking role concept as first class entity. Roles has granularity specifying the responsible participants with their own authority. Roles process information, meaning that completing an activity with a state. Modeling a system starts with goal decomposing by breaking final goal into several units. Then roles and role interactions are defined respectively.[86]

EPML, Executable Process Modeling Language, is a graphical flow language provided along with its enactment engine. It has graphic notation, formal semantics, expressive power and composability characterictics. EPML facilitates to model flow of executions and interactions among activities through a directed graph where nodes are activities or processors. Activities are computational elements whereas processors handles coordination. The language is given with its operational semantics based on a transition system.[110]

2.3.0.2 Choreography Languages

There have been choreography languages and notations introduced after the need of a global view for service-oriented systems. Although they aim at gathering interacting services to seamlessly work in an order, they have different characteristics.

ebXML BPSS was defined by Organization for the Advancement of Structured Information Standards (OASIS)[1] in 1999 to address standardization of exchanged documents among partners and specify business transactions during collaboration of business-to-business commerce, Electronic Business using eXtensible Markup Language (ebXML)[8]. Shared business documents, partner descriptions and roles, business transactions and collaborations are specified in eXtensible Markup Language (XML). One of the components of ebXML, Business Process Specification Schema (BPSS) facilitates to define collaboration of one way or two way business transactions along with handling roles of interacting partners. The language supports binary collaborations only, that is collaborations between two partners can only be described [**?**].

BPMN is the graphical modeling notation of developing business processes to real-

ize service composition; namely orchestration and choreography. BPMN has choreography constructs to define a global overview, however the definition has no enforcement on web services. Choreographies are outside or within the pools (participants) where sequence of interactions are specified via messages, involving more than one participant. Message flows connect process elements of different participants. In order to specify interactions between participants (i) conditional sequence and default sequence flows, (ii) exclusive, event-based , inclusive, parallel and complex gateways, and (iii) start, intermediate and end events can be defined. Due to non-central structure of choreography, a central source for interacting data is not maintained. Sub-choreography concept is introduced for reusability purposes which is a compound activity which is defined as a flow of other activities.

BPEL expresses choreography as abstract BPEL process definitions which indicates observable behavior from the point of other orchestration and services. Abstract BPEL processes can be formed by using executable ones and vice versa, and there exists one abstract process for each of the executable processes. Although choreography specification of each orchestration resides in their interfaces, BPEL lacks expressing a global view of participant interaction. Abstract process specifications can be mapped to UML2.0 sequence diagrams.[139, 35]

BPEL4Chor is an extension to the BPEL language in order to depict choreography related constructs which can be mapped to BPEL. BPEL4Chor consists of three basic components, namely participant behavior descriptions, participant topology, and groundings. It uses BPEL abstract process definitions so as to construct participant behavior descriptions. Participant topology is the global view of interacting services and their message interactions. Groundings as the technical view of choreography, specifies data and port types for message interactions. BPEL4Chor makes use of participant topology in order to gather interacting participant behaviors. The language is notable for introducing a choreography model integrated with BPEL orchestrations.

WS-CDL is a W3C standard aiming to specify web service choreography which represents a descriptive model including basic concepts and important constructs. Reuse in WS-CDL can be achieved through hierarchic models; namely abstract,

portable and concrete. In order to define interactions, participants, roles and relationships between two participants are specified. For information definition and declaration, types, variables and tokens are used. Variables contain information about objects in the choreography such as the messages exchanged or the state of the roles involved. Interactions involving exchanges of information between two roles have two types; one way and request-response interaction. Activities are the lowest level components of the *choreography* which do the actual work. *Control structures* combine these *activities* with other *control structures* in a nested way to specify the sequence and flow of the exchange of information within the *choreography*. However at the highest level, the *choreographies* consist of *work units*, which contains a single *activity* that is performed whenever an optional enabling condition, a *guard*, is true. The model separates process view and information view. Choreographies can be defined locally (within a defined choreography scope) or globally (sub-choreographies can be included by <perform> tag). Sub-choreographies can be defined and used by the tag <perform> for reusability.

pi4soa, described in [43], is a way to leverage WS-CDL and Pi Calculus to build more robust SOA where some of the concepts are not directly related with WS-CDL. It includes (i) type definitions (participant, role, behavior, relationship,channel, information,tokens and locators), (ii) activities (assign, choice, conditional, finalize, interaction noaction, paralel, perform, sequence, silent action, when, while), (iii) expressions, and (iv) structure (choreography, variables, exception handling). The choreography structure gathers a set of interactions which can also be reused by other choreographies. State information and channel instances are represented by variables. pi4soa supports also an exception handling mechanism which enables choreography to be terminated in predefined occurrences.

Let's Dance is a choreography modeling and representation pointing locally enforceability problem. The language aims to be abstract (conceptual), indicating formal and executable semantics, comprehensible for different stakeholders, expressive and suitable. Without technical specific details the language provides a conceptual level choreography specification. Coming with a behavioral view

of choreography, interactions between more than one service can be expressed with constraints by means of message exchanges. Precedes, inhibits, and weak precedes relationships between interactions enable to define different service behaviors. Not focusing on supporting the implementation phase, Let's Dance is not based on imperative programming constructs (variable assignment, if-then-else and switch statements, sequence, and while loops).

MAP, Multi agent protocols web service choreography language is used for choreography modeling, verifying and enacting via a simple process language. MAP is directly executable at runtime without pre-configuration in design time with multiparty support. It provides choreography interfaces along with service WSDL interfaces so as to describe complex collaborations. Having a formal ground, it provides a transformation to PROMELA verification language in order to check MAP models prior to enactment. Role concept is strongly related with each peer which decides the parts of flow the peer should follow. Roles facilitates to realize multi-cast communications via sending messages to all peers of a specific role.

WSCI, as a descriptive model introduced by OASIS, target the same goal as WS-CDL, defining choreography. Although WS-CDL and WSCI languages have different constructs, WSCI has additional features such as eventhandler and faulthandler. It is an XML-based interface description language indicating the flow of exchanged messages between web services. The observable behavior of service is expressed via temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. For interface specification, WSCI works with WSDL. There exist some work conducted on mapping petri nets with WSCI specification for verification purposes.

WSMO, like in WS-BPEL, has a similar approach for choreography specification with semantic additions, through ontology. WSMO behaves as a choreography model in a communication perspective; choreography decomposes a capability in terms of interaction with the web service. The approach uses abstract state machines (ASMs) in order to execute state transitions. It provides the conceptual underpinning and a formal language for semantically describing all

relevant aspects of web services.

Coordination Languages; Linda, Reo, Orc and CBS based on their own formal models offer different solutions to composition which can be applicable to orchestration and choreography. Orc can be used for formal orchestration modeling which deals well with asynchronous structures and failures. On the other hand, Reo is suitable both for orchestration and choreography and mainly synchronous structures. Reo can be used for transaction and compensation handling in service composition. Among these, Reo is of importance expressing choreography models which use Constraint Automata (CA) for formal foundation. For QoS extension, Stochastic Reo is proposed with verification model (Vereofy) and CTMC for transformation.

WS-Coordination is introduced by OASIS so as to be an extensible framework for coordinating activities (in this context web services as computation units) using a coordinator and a set of coordination protocols. WS-Coordination covers long running business transactions and atomic transactions. It can be used in conjunction with other specifications and application specific protocols to accommodate a wide variety of protocols related to the operation of distributed Web services. The model has three parts: (1) *Protocols* comprise coordination type-specific coordination protocols, (2) *Activation Service* enables creation of coordination context, and (3) *Registration Service* enables registration for coordination protocols. *Coordination Service* consists of *Activation and Registration Service*. Applications may not use the same coordination, instead can bound to different coordinators. WS-Addressing can be used for endpoint interpretation. For security issues, WS-Security, WS-Trust, WS-Policy, WS-SecureConversation specifications can be utilized.

CL, Choreography Language, is introduced as a simple choreography language with formal semantics. It provides a formal framework including declarative and conversational parts of choreography. Variables and roles are specified for the declarative part. A process algebraic approach is followed for specification of the conversational part which includes role interactions. The interactions are specified by one-way and request-response basic operations combined with parallel, choice and sequence operations.[44]

ScriptOrc providing a modular specification of choreography with the help of scripts by abstracting conversations among agents. Orchestrations are integrated with scripts including roles, data parameters and executable abstractions. Attributes of scripts are delayed and immediate initiation, delayed and immediate termination. Syntax and operational semantics of ScriptOrc is given formally. Orc orchestration language is used for orchestration specification which are integrated thorough ScriptOrc scripts with timeouts.[37]

BCL, Business Choreography Language, is a domain specific language which is heavily influenced by UN/CEFACT Modeling Methodology (UMM). UMM is an approach for modeling B2B global choreographies built upon UML as a profile. BCL is a means of defining a reduced set of UMM elements which gathers business collaborations and business transactions in a single diagram. The language also provides a graphical visualization for modeling choreographies.[95]

AB-WSCL, based on actor system theory, is a web service composition language given with its syntax. The aim lies in capturing the relationship between web service orchestrations and choreographies. The system is modeled with respect to actors which comprises a set of states, control thread and a set of local computations as a functional unit. The messaging among actors are asynchronous. The types of actors are activity actor, web service, web service orchestration and web service choreography which have different characteristics.[138]

eSML, eSourcing Markup Language, is a choreography language for modeling e-business collaborations whose formalism is based on Petri-nets. The language is an adoption of ECML (Electronic Contracting Markup Language) which deals with three levels of business processes, namely internal, conceptual and external levels. eSML is provided with its schema, models and examples. SML model includes who, where and what blocks in which the company data, exchanged and resource data, process and its tasks and mapping of collaborating process' lifecycles are defined. [101]

2.3.1 Variability Management

2.3.1.1 Variability in Software Systems

Complexity and change require dynamicity and adaptation of software systems. One way to achieve this is to develop systems supporting large amounts of variability which represents the ability to be extended, changed, configured and adapted for specific contexts. Variability modeling is often closely associated with product lines. Software Product Lines (SPL) is famous about utilizing a systematic way for managing variability. Likewise self adaptive systems, open platforms, and Software as a Service (SaaS) applications are designed to be variable in order to fulfill user needs. By designing with variability, reusability of artifacts and productivity increase. However, complexity of variability management, a complicated task, requires more systematic approaches.[61, 129, 42] Several variability modeling approaches have been proposed in order to manage variability through all levels of software development; from requirements to source code. Though introduced modeling approaches have the same goal, they differ in modeling characteristics such as model choices, abstractions, modeling of quality models, tooling, guidance, and focusing on development activities. A classification of variability modeling techniques is given in [117].

Variability can be modeled in all phases of product family development which addresses traceability and automation issues ranging from requirements to implementation. Different modeling techniques focus different parts of development processes, for instance expressing requirement variability in terms of features; feature modeling with commonality and variability of product lines/families. Moreover, variability modeling supports evolution in which several evolution categories are (i) New product line, (ii) Introduction of new product, (iii) Adding new feature, (iv) Extend standards support, (v) New version of infrastructure, (vi) Improvement of quality attribute. [127, 128]

For modeling variability during product line development, variability points can be introduced in various levels of abstraction; namely architecture description, detailed design documentation, source code, compiled code, linked code, and running code[98]. Each variability point can be in one of the following states at each variability level

stated in [72]; implicit, designed and bound. When a variability point is introduced to a feature model, it is denoted as implicit. When its design is decided in the architectural design phase, it becomes designed. After the variability point is finally bound to a particular variant, it is bound. Binding, when a variability point is bound to a variant, can occur at product architecture, derivation time, compilation time, linking time, start-up time and runtime. A variability point can be either open or closed. If new variants can be added to a variation point, then the variation point is open. On the other hand, if there is no way to add new variants, then it is closed.

2.3.1.2 Variability Notion in SOA

In the context of service orientation, variability modeling comprises specification of variability points, constraints and dependencies between them, related variants, variant bindings and realizations as in SPL. Variability modeling approaches should be elaborated and applied to SOA context in that the needs of service variability is different from a general SPL. Types of variability which influences behavioral part of the architecture, namely orchestration and choreography, described in [133] are listed as follows:

- variation in the web service function,
- variation in the required parameters,
- variation in the protocols, and
- variability for coordinating web services.

Moreover, different views of variability in service-orientation is addressed by [25, 80, 100]. Variability support in all composition levels, namely choreography and orchestration, is needed in order to define variable interactions between services which forms behavior of possible composite services observed from a global viewpoint. Assuming that all granularity levels can be treated as services, variability can come from following structural and behavioral views of the system:

Structural. This view comprises services, service attributes and the way they linked to each other, connectors.

- **Service interface** including its functions and parameters: Interface variability requires a configuration mechanism specifying when and how to change its functions and parameters.
- **Service connectors**: Connector variability needs a relation mechanism to indicate when and which connector is used between two services.

Behavioral. This view specifies the way the services are gathered in order to achieve a goal.

- **Service composition**: Composition variability necessitates a tailoring mechanism to define in which order and how services are interacting with each other. There is an important consideration on how to achieve interoperability between services provided with different variability. Interacting services offer different functionalities regarding their variability bindings. Therefore, service composition is responsible for associating proper bindings of interacting services. In other words, service composition should specify when and how to bind which variation points of interacting services to which variants. By this way, consistent bindings of service variability can be achieved, meaning that services can provide required functionalities. This requires a mechanism and a configuration infrastructure to establish variability associations among interacting services.

2.3.1.3 Variation Support in Existing Approaches

Several approaches have been introduced to facilitate development of variable service architectures in the context of Service Oriented Software Product Lines (SOSPL)[126, 93, 25, 133, 66, 65, 111, 80, 85, 39, 108]. These approaches are evaluated with respect to the following criterias in Table 2.6 and Table 2.7:

Feature Modeling Support Defines whether the approach has feature modeling support to model variability. "Yes" indicates that the approach associates a feature model. "No" means that the approach does not utilize a feature model. "Not known" means that whether the approach uses a feature model is not known.

Table 2.5: Comparison of variation support in existing approaches

Approach	[126]	[93]	[25]
Feature Modeling Support	No	Yes	Yes
Variability Mapped From	FR and NFR requirements	Feature Model	UML stereotypes
Variability Mapped To	Decision model	Business Process Template	Services
Variability Specification in Composition	No	No	No
Type of Variability Support	None	None	Operations and parameters variability, transport variability, endpoint variability, discoverability and binding variability, error-handling variability
Tool Support	No	No	No
Additional Utilities	Decision model uses graph walking algorithm, query algorithm, variability dependency graph, entity/service mappings and composition search space	Feature modeler, business process modeler, feature mapper, and service manager for selection and binding	Design patterns recommended

Table 2.6: Comparison of variation support in existing approaches

Approach	[133]	[39]	[108]
Feature Modeling Support	No	No	Not known
Variability Mapped From	UML Class diagrams	Requirements	OVM Model
Variability Mapped To	UML Class diagrams with use of patterns	Process Variants and Context Profiles	UML Diagrams
Variability Specification in Composition	No	No	Yes
Type of Variability Support	Variation in the web service function, in the required parameters, in the protocols and for coordinating web services	None	Variation in the web service function, in the required parameters, in the protocols
Tool Support	No	No	Yes
Additional Utilities	Strategy,Decorator,Adapter, Iterator and Chain of responsibility	Variability management in the process level. Integration rules	2 level representation of architecture; such that the main representation excludes variants, and the secondary diagrams model realizations of variabilities separately

Table 2.7: Comparison of variation support in existing approaches - cont'd

Approach	[66, 65, 111]	[80]	[85]
Feature Modeling Support	Yes	Yes	No
Variability Mapped From	Components	COVAMOF Model	Requirements
Variability Mapped To	Plan	Orchestration Language	Rules
Variability Specification in Composition	No	Yes	No
Type of Variability Support	None	Variation in the web service function, in the required parameters and for coordinating web services	Variation in the web service function, in the required parameters and for coordinating web services
Tool Support	Yes	Yes	Yes
Additional Utilities	2 level OWL-S ontology, utility function, plans	Existing BPEL engine ActiveBPEL adapted and a configuration file specified	Extension to Jolie orchestration language with rule-based adaptation rules

Variability Mapped From Defines the models where the variability information of the system comes from.

Variability Mapped To Defines the models to where the variability information of the system is mapped.

Variability Specification in Composition Defines whether the approach enables to define variability in service composition. "Yes" indicates that the approach has models or constructs to specify variation in composition. "No" means that the approach does not variability in composition.

Type of Variability Support Defines supported variability types by the approach. "None" indicates that the approach does not support any variability specification.

Tool Support Defines availability of tools. 'No' means not available, 'Yes' means available.

Additional Utilities Defines a set of models, implementations, representations, files, algorithms or patterns utilized to support variability.

2.3.1.4 Existing Variability Models

Several variability models have been proposed over the years to capture, organize and represent variability which differ in the concepts. In [48], thirty-three approaches are reviewed, categorized into issue groups and analyzed. Within these, variability models dealing with architecture and product derivation are taken into account and explained separately.

xADL proposed in [134] is an architecture description language devised for modeling product line architectures based on XML syntax. The language uses (i) *Structure* and *Types* schema to define modeling architectural constructs for capturing a single architecture which includes components and connectors, (ii) Options, Variants, and Boolean Guard schemas to model variability in space through explicit variation points in the architecture, (iii) Boolean Guard schema to guard optional and variant elements by boolean expressions which decides

inclusion or exclusion of optional elements and selection of particular variants, and (iv) Versions schema to enable evolution of the architecture building on top of the other schemas.

Koalish proposed in [28] is an architectural description language extending Koala with variability, intended to model product line architectures. It is a product configuration based and an architectural centric approach. The language models and configures components and interfaces to form a logical structure of the system, which extends Koala with explicit variability constructs and resolves variability in compile time. Alternative and optional components and constraints on how components, their attributes and interfaces can be used are specified explicitly. The reasoning for configuration of component models comes from an existing inference tool for Weight Constraint Rule Language (WCRL). The Koalish configurator facilitates to construct valid products with regard to bound variables.

Systematic Integration of Variability into Product Line Architecture Design proposed in [131], an architecture centric approach, deals with variability in multiple views. It is an extension to the IEEE P1471 recommended practice for architectural description which uses variation points to model variability in the architecture description.

Divide and Conquer Variation Management proposed in [82], is a configuration-based approach following two dimensional view of variation management which operates on the file system level and is neutral to architecture, design, and language. The approach divides variation management into nine issues and then conquers them by addressing each of these sub-problems. The issues are categorized under three clusters; (i) Basic Configuration Management (version management, branch management, baseline management, branched baseline management), (ii) Component Composition (composition management, branched composition management), and (iii) Software Mass Customization (variation point management, customization management, customization composition management). Mass customization suggests a path starting from the file, then a customized component and finally to a customized product. Variation points are defined and selection logic is implemented with a set of file

variants. Instantiation of a customized component and a product is achieved by instantiating each variation point in the components of a product.

COVAMOF proposed in [118] is a framework for variability modeling which represents a variation point as a first class entity in all abstraction layers. The framework allows hierarchical organization of the variability by specifying realization relations between variation points. It facilitates to represent dependencies among variation points and modeling dependency relations. COVAMOF Variability view comprises two views over all abstraction layers (feature model, architecture and component implementation); variation point view and dependency view. In variation point view, variation points, variants, realization relations and dependencies are specified. In dependency view, dependencies and dependency interactions are handled in order to provide a strategy to resolve dependencies.

OVM proposed in [30] stands for Orthogonal Variability Model which intends to represent variability in architecture through establishing dependency relations between development artifacts which is used to document variability in design and realization artifacts. The model captures product line variability by specifying external and internal variation points, variants, constraints between them and supporting optional, and alternative variation points.

VSL proposed in [36] stands for Variation Specification Language which distinguishes variability at the specification and at the realization level. The specification level comprises user choices under *variabilities*, on the other hand the realization level variation points are mapped to assets depicting the places where the choices are implemented by taken actions. The language supports two sets of variation point; Dynamic Variation Points (runtime variability) and Static Variation Points (preruntime variability).

Kumbang proposed in [27] is a domain ontology for representing variability in software product families and combining Koalish with concepts from feature modeling. The ontology is developed as a profile extending the UML metamodel. It incorporates components and features with compositional structure and attributes, the interfaces of components and connections, and constraints.

Model-driven approach for SPLs proposed in [97] introduces a single metamodel comprising whole development process in which relationships and constraints among all artifacts are specified and decisions on variant features are propagated consistently throughout all artifacts. The variation points are located in one place which captures relationships and constraints among variation points, or decisions, without a direct link to a development artifact's variation point. Each variation point is related to a decision which constrain other variation points and that can explicitly be related to a domain concept.

Variability Expression proposed in [112] explains a process, methods and techniques to express the variability and its usage to derive new products, based on Software Product Line Integration Technology (SPLIT). The approach comprises a global framework for software product lines, having variability and decision modeling support, binding and instantiation of products and facilitating asset storage and evaluation within domain and application engineering activities. It provides a multi-level decision model covering levels for variation points, assets and to core assets. However, global consistency of the model is an open issue.

First class feature abstractions for product derivation proposed in [23] introduces an approach focusing on design and implementation level with formal description of features in the scope of software product families (SPFs). At the design level, features are formally expressed as a collection of roles which can be realized by different base components. Then these base components become actors of the system, satisfying a required set of functionality. A specific product is derived by selecting a number of features which result in a set of derived components and the mapping from the actors to the derived component implementations.

FDL proposed in [57] is a feature description language which is used to address variability management in product line architectures where software products are constructed from customer-specific feature selections. Specified features are mapped to software packages whose sources are then merged by the source tree composition technique to construct software components. The software packages can realize a feature or implement a functionality shared by other

packages.

Staged Configuration Using Feature Models proposed in [84, 83] indicated a car dinality-based notation for feature modeling in software product lines. Cardina lity-based feature models are cast to context-free grammars to provide formal-ism. Staged configuration approach satisfies the need for collaborating different stakeholders by stepwise specialization of feature models, staged configuration of platforms, components, and services. At each stage a feature model is taken and a specialized feature model is yielded.

CVL proposed in [74] stands for Common Variability Language facilitating to ex-press a generic and standardized variability model which brings together a vari-ation and a resolution model and transformations to resolved domain models. Revised version of CVL is submitted to OMG in 2012, on the way to be an OMG standard. CVL Architecture incorporates variability points, variability specification, constraints and resolutions. Variability Specification is similar to features in feature modeling, likewise a tree structure where variation points can be bound to. Constraints expressed by sub-language of OCL intricate re-lations between Variation Specifications. Variation Points refer to base objects and define their modifications precisely.

Variability Model for SOA proposed in [99] introduces a mechanism to modify ser-vices to suit business process requirements. The model and approach are based on VOE methodology and VOSD service derivation algorithm, targeting ser-vice orientation. Variations in service, data graphs of services and variants are formally modeled. An algorithm is provided to check whether the variant of a service is a "legal variant". The prototype is built on IBM's Rational Software Architect modeling tool.

Modeling Variability in Component and Connector View of Architecture using U nified Modeling Language (UML) proposed in [108] introduces a modeling method which extends OVM fulfilling and addressing realization and trace-ability issues. Variation in components, connectors and interfaces is expressed using a UML 2.0 profile. A two-level representation of architecture is pro-posed; (1) A simplified model exhibiting big picture of the system with abstract

components, connectors and interfaces regardless of variability and (2) a more detailed variability model including variability points, variants and realizations.

Managing service variability proposed in [100] introduces exposed, composition, partner, partner exposed variability types and their relationships in the context of service-orientation. Dynamic and recursive variability communication among service providers, service composers and service consumers are pinpointed. Inherited characteristics from software and service variability management are discussed; namely types of variation points, constraints, their realizations, variability at different levels of abstraction. The variability type needs and their relationships are explained in detail with a case study.

2.4 Comparison Framework

Architecture modeling is expected to end up with the architectural components, their connectors and the composition context. Based on this principle, service-oriented architecture defines services, messaging protocols, connections and service compositions by capturing interactions between services. Reusing existing service-oriented artifacts requires a systematic approach one of which is modeling variability in service-oriented architecture. To this end, existing orchestration and choreography languages are compared through the comparison framework. The framework is based on three basic components; Variability modeling, Composition and configuration of models, and Tool Support. The framework aims at elaborating and revealing capabilities of existing languages focusing on variability support in particular which requires different mechanisms to handle at the language level.

1 Variability modeling Evaluation questions related to modeling variability component are explained one by one.

1.1 Types of Variation Point and Variants Which types are supported by the approach which enables to specify variation point and variant?

1.2 Constraints Which mechanisms is supported for defining constraints between variations?

1.3 External and Internal Representation of Variation Point Does the app roach have a support to express external and/or internal variation points? External (Ext.) variation point stands for externalization to outer world to be configured by others, whereas internal (Int.) variation point means internalized to be bound by itself. "None" indicates that the model does not specify variation points as external and internal.

1.4 Realization Which mechanism is supported to define a relation in which (i) a high level variation point is specified by means of other variation points as configuration purposes or (ii) a higher level abstraction that enables to bind low-level variation points? Here, low-level variation point stands for a variation point directly related with development artifacts.

1.5 Design Artifact Mapping Which mappings can be defined among variation points and variants to design artifacts such as development artifacts, assets or products?

2 Composition and Configuration of Models Evaluation questions has two sub-components; (i) Composition which defines the way to gather service interactions and (ii) Configuration of Models which defines handling of service oriented model configurations according to variability bindings if the approach support variability.

2.1 Composition includes how existing languages capture interactions between a services and their environment even from a global or local perspective.

2.1.1 Composition Approach Which approach does the language follow, either choreography or orchestration or both?

2.1.2 Modeling Approach Which adopted modeling approach does the language follow? The language can be either based on interaction or interconnection or both. Modeling based on interaction represents definition of one building block (document or specification) for the whole system, whereas interconnection suggests modeling control flow logic per participant.

2.2. Configuration of Models comprises the mechanisms to support variability in service interfaces (parameters and functions), connectors and/or

composition and variability associations among variable services. Variability association defines the relation of variability bindings between services in any level, namely choreography, orchestration or atomic service levels. Here, service concept consists of choreography, orchestration, and atomic services.

2.2.1 Variability Support Does the language support variability? 'Yes' indicates that the language provides explicit language mechanisms for variability. 'Implicit' indicates that although the language does not provide explicit mechanisms, variability is supported implicitly. 'No' means that there is no variability support.

2.2.2 Variability in Service Interface Does the language has constructs or a mechanism to express variability in interfaces, namely in parameters and functions?

2.2.3 Variability in Connectors Does the language have constructs or a mechanism to express variability in connectors which services are connected with?

2.2.4 Variability in Composition Does the language have constructs or a mechanism to express variability in service composition?

2.2.5 Variability Association Specification Does the language have constructs or a mechanism to express variability associations?

3 Tool Support Evaluation questions related to aiding tools to specify, analyze, verify and generate code from related specification.

3.1 Specification Does the approach provide a tool to specify language constructs? "Yes" indicates the approach is supported by a tool. "No" means there is no tool support for specification.

3.2 Analysis Does the approach provide a tool to analyze language models or specifications? "Yes" indicates the approach is supported by a tool. "No" means there is no tool support for analysis.

3.3 Verification Does the approach provide a tool to verify language constructs or specifications? "Yes" indicates the approach is supported by a tool. "No" means there is no tool support for verification.

3.4 Code Generation Does the approach provide a tool to generate code from language constructs or specifications? "Yes" indicates the approach is supported by a tool. "No" means there is no tool support for code generation.

3.5 Configuration Does the approach provide a tool to configure language constructs or specifications? "Yes" indicates the approach is supported by a tool. "No" means there is no tool support for configuration.

3.6 Tool What is (are) the name (s) of tool that which carries out supported features? "Not in Use" means the tool is out of use. "None" indicates that the approach does not have any implemented tool support.

In the following sections selected orchestration and choreography languages are compared according to the comparison framework components one by one.

2.4.1 Variability Modeling

The variability models explained in Section 2.3.1.4 are evaluated with respect to the variability modeling component in Table 2.8 and Table 2.10 where the abbreviations "VP", "V", "Ext." and "Int." stand for "Variation Point", "Variant", "External" and "Internal" respectively.

Variability Approach and Variation Point Specification In xADL, by schema mechanisms, the language enables component interface and connector variability. The language provides variability both in space (binding in design time, invocation time and runtime) and time (versioning). Although variation points are specified explicitly, the conceptual difference between external and internal variation points does not exist. The language is generic and dedicated to Software Product Lines approach.

The Koalish component model targets configuration by means of models, however variation point and variants are not specified. Configuration component enables to define possible structural inclusions (the number and type of components) with constraints which brings about optional, mandatory, and alternative components. The component model and Koalish language enable component,

Table 2.8: Comparison for Variability Modeling Component

Approach	**Types of VP and V**	**Constraints**	**Ext. and Int. Rep. of VP**	**Realization**	**Design Artifact Mapping (To)**
xADL	Optional, Mandatory, Alternative	Logical Constraints by Boolean Guard	None	None	Architectural elements; component (interface) and connector
Koalish	None	Constraints by Weight Constraint Rule Language (WCRL)	None	None	Components, their attributes and interfaces
[131]	Not known	Decision model	None	None	Artifacts
[82]	Optional, Mandatory	Not clear	None	None	Files
COVAMOF	Optional, Value, Alternative	Dependency via Variation Point Interaction Diagram	None	Realization Relations between VPs	Product
OVM	Mandatory, Optional, Alternative (cardinality)	Requires, Excludes	Ext. and Int. VP	None	From VP and V to development artifacts

Table 2.9: Comparison for Variability Modeling Component-cont'd

Approach	**Types of VP and V**	**Constraints**	**Ext. and Int. Rep. of VP**	**Realization**	**Design Artifact Mapping (To)**
VSL	Not specified	Dependency	None	Realization Relation between Variability and VPs	VP and V to assets
Kumbang	None	Constraints by Kumbang Constraint Language	None	None	From feature model to components, their attributes and interfaces
[97]	Optional, Mandatory	Constraints among VPs in decision model	None	None	All artifacts of each development phase
[112]	Optional, Mandatory, Alternative	Constraints by decision models	None	Multi-level decision models	All artifacts of each development phase
CVL	Optional, Mandatory, Alternative	Constraints by sub-language of OCL	None	Configurable units	UML models

Table 2.10: Comparison for Variability Modeling Component-cont'd

Approach	Types of VP and V	Constraints	Ext. and Int. Rep. of VP	Realization	Design Artifact Mapping (To)
[23]	None	None	None	None	To actors, functions and base componentsl
FDL	None	None	None	None	Componentsl
[84]	None	None	None	None	Platforms, components,and services
[99]	Optional, Mandatory	None	None	None	Services and business processes
[108]	Mandatory, Optional, Alternative	Requires, Excludes	Ext. and Int. VP	two level architecture representation	Architectural components
[100]	exposed, composition, partner, partner exposed variability	require, exclude	exposed as external VP, composition and partner as internal variability	None	Architectural components

interface and connector variability in space. The language is dedicated to Configurable Software Product Families approach. Bindings denote the flow of function calls in a network of components connected through bindings between their interfaces.

In Systematic Integration of Variability in to Product Line Architecture Design (SIVPLAD) approach, the language enables variability in space. However, variation point concept and dependencies are not first class entities in the modeling approach. Although product architecture is represented as a decision model, modeling language is not clearly defined.

In Divide and Conquer Variation Management (DCVM) approach, the language provides variability both in space (domain space) and time (sequential, parallel). The approach is neutral to architecture, design, and language, instead file-based.

COVAMOF is a general variability model in space applied for all software artifacts.

OVM model is dedicated to software product lines which interrelates variation information to software artifacts.

VSL language is a generic variability model in space, gathering specification and realization levels together and supporting runtime and preruntime variation.

Kumbang model targets configuration by means of Koalish component model incorporating with feature model, however variation point and variants are not specified. Kumbang enables component, interface and connector variability in space.

Model-driven approach for SPLs (MDAS) approach facilitates to specify and manage variation point and variants in a single model throughout development process. The approach enables to define component and interface, code variability in space.

In Variability Expression (VE) approach, even though the approach does not assume a particular notation, UML is used for demonstration and experimentation purposes. Therefore, variation points and variants are represented as classes.

First class feature abstractions (FCFA) approach employs a formal feature model

for software product families (SPFs) to configure domain components. Therefore, no variation point and variant specification exists. FDL approach employs a formal feature description language for software product lines. Therefore, no variation point and variant specification exists.

Likewise, Staged Configuration Using Feature Models (SCUFM) approach includes a novel cardinality-based notation for software product lines. Therefore, no variation point and variant specification exists.

Variability Model for SOA (VMS) approach enables to define service interface variability and to integrate it with business process variability. However, integration with business process variability is not pinpointed enough. Besides, there is no explicit mechanism to specify a variation point and a variant.

In Modeling Variability in Component and Connector View (MVCCV) approach, types of variation points, variants and their realizations are addressed with the proposed variability modeling approach. Variability in architectural constructs, namely components, connectors and interfaces are specified.

Managing service variability (MSV) approach is important for elaborating different characteristics of service variability which covers service, service interface and composition variability concepts. External and internal variation point concepts are mapped to newly introduced variability types.

In Divide and Conquer Variation Management (DCVM) approach, COVAMOF model, VSL language, FCFA approach and CVL, although variation points are specified explicitly, the conceptual difference between external and internal variation points is missing.

Realization Higher level configurations, variability mappings are not taken into account in xADL, Koalish, SIVPLAD, DCVM, OVM, Kumbang model, MDAS, FCFA, FDL, SCUFM, VMS and MSV approaches. On the other hand, in COVAMOF, variations on higher-level abstractions are realized by variation points in lower-levels of abstraction by realization relation mechanism. In VSL, realization of variabilities in specification layer is achieved by variation points in the realization level which provides a higher-level configuration. In VE approach, a multi-level decision model facilitates to configure the product line from higher level (core asset level) to lower-level (variation point level). In

MVCCV approach, two views are provided for abstracting complexity in variable architecture. The first one represents the big picture of the system with abstract components which includes variation details. The second one includes system variability details and their realizations.

Variability in composition In xADL, Koalish, Kumbang model, MDAS, VE, FCFA, CVL and MVCCV approaches, although the infrastructure is convenient to specify variability in composition, process view of the architecture is not explicitly mentioned. In SIPLAD, process view of the architecture is not explicitly mentioned. In DCVM, it is stated that variability in composition is achieved by instantiation of variation points, however how it is achieved is not clear.

2.4.2 Composition and Configuration of Models

Obviously for small systems in which variability is limited we could handle configuration of the composition of services and the corresponding orchestration specifications using traditional approaches such as interaction diagrams. Variable parts and their relations can be modeled and implemented by data and through 'if' control structures. However, for integration of large scale systems soon the traditional approaches are less expressive and not tractable. Therefore, existing orchestration and choreography languages explained in Section 2.2.2 are evaluated with respect to composition and configuration of models component in Table 2.11.

BPML, BPEL, VxBPEL, Jolie, Jorba, PML, RBXPDL and EPML follow interconnection approach providing constructs for service orchestration. The only orchestration language which supports variability explicitly with a separate variability model is VxBPEL. As a rule-based approach built on top of Jolie language, Jorba enables dynamicity in orchestrations with an adaptation mechanism.

BPEL abstract processes, BPEL4Chor, Let's Dance,WS-CDL, MAP, CL, ScriptOrc, eSML, BCL and WS-Coordination facilitate to define service choreography specifications. Among them, BPEL4Chor follows interconnection approach by scattering service interaction definitions to each of the service. Global message links are gathered in a single Topology model. However, the language does not fulfill interaction

approach completely because global control flow logic is missing. MAP provides an interaction model through separation of choreography definition to service peers.

The languages which targeting both orchestration and choreography are WSMO, BPMN, Reo and AB-WSCL. WSMO is different than BPMN and Reo in following solely interconnection model.The other two comprise interaction and interconnection models. Among them, Reo provides a hyper-graph transformation mechanism in order to support variability.

Although VxBPEL, Jorba and Reo languages have variability support, no one enables to specify connector variability. Both of them targets service composition variability, whereas none of them provides a variability association mechanism. The only one providing interface variability is Jorba by reconfiguration with adaptation interfaces. The only explicit variability model, COVAMOF, integrated with the orchestration language is VxBPEL in which BPEL is extended with variability constructs. The language does not provide a global view of variability when a set of orchestrations are interacting with each other. Jorba defines adaptation interfaces and manager which reconfigures the interfaces whenever a change in a function or a parameter is required. As the variability logic is hidden between rules and implicit, the management of rules and variability is usually difficult.

Comprising orchestration and choreography models in one place, Reo proposes a hyper-graph transformation mechanism to reconfigure service interactions. In this mechanism, services are treated as nodes which are connected via edges. Reconfiguration of edges enables reconfiguration of service interactions meaning composition variability as an internal part of the system. Therefore, composition variability cannot be specified as an explicit variability model.

Based on the results of Table 2.11 variability in both orchestration and choreography is not supported in any of the languages. That is, there is no explicit representation of variability and variability dependency, and a single variability model incorporated with all granularity levels, namely choreographies, orchestrations and atomic services. Moreover, none of the languages addresses variability associations which enable proper variation point bindings of interacting services.

Table 2.11: Comparison for Composition and Configuration of Models Component

Language	Composition Approach	Modeling Approach	Variability Support	Variability In
BPEL 2.0	Orchestration	Interconnection	No	None
VxBPEL	Orchestration	Interconnection	Yes	Composition
Jolie	Orchestration	Interconnection	No	None
Jorba	Orchestration	Interconnection	Implicit	Interface, Composition
BPML	Orchestration	Interconnection	No	None
PML	Orchestration	Interconnection	No	None
RBXPDL	Orchestration	Interconnection	No	None
EPML	Orchestration	Interconnection	No	None
WSMO	Orchestration, Choreography	Interconnection	No	None
WS-CDL	Choreography	Interaction	No	None
Let's Dance	Choreography	Interaction	No	None
BPEL4Chor	Choreography	Interconnection	No	None
WS-Coordination	Choreography	Interaction	No	None
WSCI	Choreography	Interaction	No	None
CL	Choreography	Interaction	No	None
eSML	Choreography	Interaction	No	None
ScriptOrc	Choreography	Interaction	No	None
BCL	Choreography	Interaction	No	None
MAP	Choreography	Interconnection, Interaction	No	None
AB-WSCL	Orchestration, Choreography	Interconnection, Interaction	No	None
BPMN 2.0	Orchestration, Choreography	Interconnection, Interaction	No	None
Reo	Orchestration, Choreography	Interconnection, Interaction	Implicit	Composition

2.4.3 Tool Support

Existing variability models, orchestration and choreography languages, explained in Section 2.3.1.4 and Section 2.2.2 respectively, are evaluated with respect to the tool support component in Table 2.12 and Table 2.13.

Table 2.12: Comparison of Tool Support Component for Existing Variability Models

Approach	**Spec.**	**Anal.**	**Verif.**	**Code Gen.**	**Conf.**	**Tool**
xADL	Yes	Yes	No	Yes	Yes	xArch and Menage
Koalish	Yes	Yes	No	No	Yes	Koalish Tool
[131]	Yes	Yes	No	No	No	None
[82]	No	No	No	No	No	None
COVAMOF	Yes	Yes	No	No	No	COVAMOF-VS Tool Suite
OVM	No	No	No	No	No	None
VSL	No	No	No	No	No	None
Kumbang	Yes	Yes	No	No	Yes	Kumbang Configurator
[97]	No	No	No	No	No	None
[112]	Yes	Yes	No	No	Yes	UML
CVL	Yes	Yes	No	No	Yes	CVL Tools
[23]	Yes	Yes	No	No	No	Prototypical Tool
FDL	Yes	Yes	No	No	No	autobundle Tool
[84]	Yes	Yes	No	No	No	Prototypical tool
[99]	Yes	Yes	No	No	Yes	Prototype
[108]	Yes	No	No	No	No	UML2.0
[100]	Yes	No	No	No	No	UML2.0

xADL Tools, xArch and Menage, enable to specify, analyze, generate Java classes, and configure xADL models. Components are specified, analyzed and configured by the Koalish Tool. SIVPLAD Tool[131] only facilitates to specify and to analyze file variants. Variation points, variants, variation realizations and constraints are specified and analyzed by the COVAMOF-VS Tool Suite. Components are specified, analyzed and configured by Kumbang Configurator with the help of Koalish code. A prototypical tool for VE [112] facilitates to specify UML diagrams with stereotypes, to analyze and configure models via decision models. Autobundle tool for FDL, a prototypical tool for SCUFM [84] and a tool for FCFA [23] enable to specify and analyze feature diagrams and to configure the product line models. A prototype for VMS [99] has ability to specify variation, to analyze whether variants are"legal" or not, and to con-

figure via service interface customization for business process creation. In MVCCV [108] and MSV [100] approaches, UML is used for specification via stereotypes. DCVM [82], OVM, VSL, and MDAS [97] approaches do not provide any tools.

Table 2.13: Comparison of Tool Support Component for Existing Orchestration and Choreography Languages

Approach	**Spec.**	**Anal.**	**Verif.**	**Code Gen.**	**Conf.**	**Tool**
BPML	No	No	No	No	No	None
BPEL 2.0	Yes	Yes	Yes	No	No	BPEL Tools
VxBPEL	Yes	Yes	No	No	Yes	Adapted ActiveBPEL engine prototype and ValySec
Jolie	Yes	No	No	No	No	Jolie Interpreter Engine
Jorba	Yes	No	No	No	Yes	Jorba Prototype
PML	Yes	Yes	Yes	No	No	PMLCHECK
Reo	Yes	Yes	Yes	Yes	No	Reo Tools
WSMO	Yes	Yes	Yes	No	No	WSMX
BPMN 2.0	Yes	Yes	Yes	Yes	No	BPMN Tools
WS-CDL	Yes	Yes	Yes	Yes	No	Not in use
Let's Dance	Yes	No	No	No	No	None
BPEL4Chor	Yes	Yes	Yes	Yes	No	BPEL4Chor Tools
MAP	Yes	Yes	Yes	Yes	No	MagentA
WS-Coordination	Yes	No	Yes	No	No	XML Tools
WSCI	Yes	No	No	No	No	XML Tools
CL	No	No	No	No	No	None
ScriptOrc	No	No	No	No	No	None
BCL	Yes	No	No	No	No	Visual Studio IDE
AB-WSCL	No	No	No	No	No	None
RBXPDL	Yes	No	No	No	No	XML Tools
EPML	Yes	No	No	No	No	EPML engine
eSML	Yes	No	Yes	No	No	eSML Verification Tool
pi4soa	Yes	Yes	Yes	Yes	No	pi4SOA Tools Suite

BPML is not supported currently. For BPEL and BPMN language, several tools are available even for industrial level. Existing approaches introduce different models for verification of BPEL and BPMN models. For VxBPEL, ValySec analysis tool is implemented and ActiveBPEL engine is adapted to process variability information. The tools are not in use; will be open sourced soon. Jolie is currently supported by an interpreter implemented in the Java language, which can be run in multiple operating systems including Linux-based operating systems, Apple OS X, and Microsoft Windows. Jorba tool as a prototype is very limited, and it will need a lot of work to become fully able to deal with complex adaptation scenarios available for use. PML comes along with its specification, analysis and verification tool, PMLCHECK which translates a process model into a graph representation. For the Reo language, BPMN modeller, BPMN2Reo converter, Reo graphical editor, Reo reconfiguration plug-in, Reo simulation engine, Reo validation plug-in (to Eclipse) and java code generation engine are developed. WSMO approach has several tools for modeling and an execution environment for dynamic matchmaking, selection, mediation and invocation of semantic web services based on WSMO; WSMX. For WS-CDL, a visualization tool was implemented in Erlang; however it is currently unavailable [77]. Maestro tool is implemented for Let's dance, however, not available and SAP ended its support in 2006. BPEL4Chor has several tools [73]; Oryx editor for specification, BPEL to BPEL4Chor and BPEL4Chor to BPEL converter tools which can be run inside Eclipse. Verification of BPEL4Chor constructs are achieved by BPEL2oWFN [87]. MagentA, an implementation of MAP which provides a concrete, and open-source framework for the enactment of distributed choreographies. Besides, verification of MAP is achieved via its model transformation to PROMELA language. CL, ScriptOrc, and AB-WSCL have no tool support. BCL is developed based on Microsoft Domain-Specific Language Tools (DSL Tools) on top of the Visual Studio IDE. For EPML, process specifications in XML representation is executed by its enactment engine written in Java. pi4soa tools suite includes tools from Choreography Description Designer to Choreography Validation Framework which are released (and supported) through JBoss Tools[43].

2.4.4 Discussion and Problem Statement

State of the art variability support in orchestration and choreography languages are evaluated under the comparison framework. The analysis of the existing languages shows that variability in both orchestration and choreography is not supported in any of the languages. Besides, interface and composition variability support is not explicitly addressed with a single variability model incorporated with choreographies, orchestrations and atomic services. Only VxBPEL language has an explicit variability specification based on the COVAMOF model which enables definition of variability in service composition. However the language can not cover all service variability needs, namely variability in interface and connector. Variability associations can be defined by means of CVV dependency view, but it still needs an additional mechanism to specify which interacting services of which orchestration is dependent each other. Because variability association logic lies in revealing correlations of variation point bindings of interacting services within an orchestration.

No single orchestration or choreography language addresses all variability needs. Several variability models are proposed to capture, organize and represent variability which differ in emphasizing concepts since 2000. A review of variability model categorizations can be found in [48]. COVAMOF [118], OVM [30]and CVL [74] like variation models can be used to represent service variability, however still they are lack of representing all variability needs. Beyond that, there is no specified mechanism to map orchestration and choreography variability consistently.

The analysis of the current orchestration and choreography languages shows that none of the current languages supports variability both at orchestration and choreography level. None of them provides interface, connector and composition variability in one place with variability association mechanism. Moreover, there is no language supporting interface and composition variability in a single model explicitly addressing at all levels; namely choreography, orchestration and atomic service. The following problems are identified concretely:

- **Lack of explicit expressiveness of variability in choreography specifications.** There is no language that explicitly represents variability in choreog-

raphy in order to integrate orchestration specifications. Moreover, variability modeling in choreography, orchestration and atomic services as a whole is not explicitly covered in one single model. This impedes the consistent configuration of choreography and orchestration specifications with respect to variability. The lack of explicit abstractions for variability easily leads to the scattering of variability concerns over service compositions. Likewise, enabling or disabling a variability results in reorganization of the composition. This complicates the understanding of variable parts, relations amongst them and the overall goal for business process engineers and developers. Tracing these scattered variations can be achieved to a certain degree, but in large scale systems traceability and understandability decrease gradually. As a result, this scattering reduces the maintenance of the system.

- **Lack of explicit specification of variability associations among interacting services.** A choreography interrelates a set of orchestrations, atomic services and establishes connection with other choreographies. Interacting service variability constraints and shapes possible choreography abilities and composition. Likewise, variability of choreography dictates proper service variability bindings and specified configurations which bring along service interfaces with different functionality and parameters. In order to reveal these dependencies and relations between choreography and services, an explicit association of variation point bindings should be defined. In other words, configuring choreography requires configuring other services in order to consistently collaborate with each other. Therefore, configuration and binding of service variability requires an integrated model comprising choreography, orchestration specifications, and atomic services with variability. There is no language supporting such integrated configuration model dealt with variability of all granularity levels.

- **Lack of support for reusing existing choreographies.** The importance of reusing existing choreographies is addressed in some approaches, but reusing as a part of the other choreography is not emphasized sufficiently. There are ways to handle choreography-to-choreography relationships such as collaborating via exposed choreography interfaces. In case of variability, it is more difficult to utilize choreography specifications with proper bindings. Therefore,

the way to bind to other choreographies should be specified.

In order to fully support variability, needs of each level namely choreography, orchestration and atomic service should be addressed. As each view has different needs, variability specification and management at all views are related but different. The challenges of variability representation needs within service-oriented context lie in determination of following items:

- the types of variation points and variants,
- associations between variation points and variants,
- the parts where and how variability associations is stated,
- the effect of variation points to shared elements of choreography,
- the parts where and how choreography variability in composition is stated or referenced from outside,
- the relationships of service and choreography capabilities with variability and where they are specified.

The target variability model faces with following obstacles:

- External specification of variability opened to outer world in order to configure in an intended manner,
- Depiction of composition variability realizing external variability while dealing with internal variability, and
- Association of external service variability with the composition variability

All in all, all interacting services should define their external variability if exists, which can be either orchestration and atomic services. While configuring interacting services, their variability bindings should be consistent with each other, if one of the service variability binding affects that of the other one's. Therefore, a comprehensive variability model from the global point of view is missing to manage variability in choreography level which can then be mapped to the variability of interacting services in order to achieve a consistent architecture.

CHAPTER 3

VARIABILITY IN CHOREOGRAPHY LANGUAGE: XCHOR

This chapter introduces a new metamodel and its realization XChor language which addresses variability support at choreography level in order to integrate variable orchestrations and atomic services consistently. A variability model integrated with choreography model is proposed as a single model so as to systematically manage variability from a global point of view. After variability modeling requirements are revealed, two real life case study are explained for demonstrating variability in choreography level. Afterwards, the proposed metamodel and XChor language are explained with language constructs and exemplified via case studies. XChorS Tool is that supports specification, analysis, and configuration of XChor models is represented with its capabilities. Then, validation of the XChor language is studied for modeling service variability and choreography specification.

3.1 Variability Modeling Requirements for Choreography Languages

A choreography comprises a set of orchestrations, atomic services and collaborates with other choreographies. All interacting services knows only their variability information regardless of other ones'. Variability binding of a service can entail a proper variability binding of another one. Associations can be established one to one, however, the whole such associations can not be observed at their level, instead it should be managed in a high level abstraction, namely choreography. By this way, the choreography establishes a mechanism that interacting services can behave as expected by means of consistent configurations. In fact, the choreography forms a context when

and how variability of interacting services bound along with variable service interaction specification. Therefore, a variability model addressing all levels should be elaborated indicating which types of variability is supported and how. To this end, requirements for variability model in choreography language and its relation with other choreographies, orchestrations and atomic services are listed as follows:

- Variability should be uniform and treated as a first class concept at all abstraction levels of architecture, namely atomic service, orchestration, and choreography. In other words, multi-level variation representation should be specified.
- Variability should be represented in an hierarchical organization explicitly which reduces complexity of variability points and eases variability management. A global view on variability points helps product developers to understand ultimate goals of the system.
- Constraints on variability should be specified in all levels of architecture.
- Variability associations defined by means of explicit variability mappings should be treated as first class entities in order to realize a valid set of products.
- In order to establish variability associations, orchestrations and atomic services should offer and open their variability explicitly to the choreography in a global view.
- Variability of a choreography specification should be specified explicitly for utilization. The choreography can be used by other ones, which requires explicit variability representation in choreography level.
- Both centralized (choreography) and decentralized (orchestration and atomic services) variability management should be conducted at the same time for consistency. A centralized view is based on decentralized ones.
- Variability should be correlated with design artifacts, namely service interfaces and choreography specifications.
- A process for specification and management of variability on architecture should be defined for developers and a set of tools should be provided to ease variabil-

ity management ranging from variability specification, analysis, configuration, verification to code generation.

3.2 Case Study

In order to demonstrate variability in choreography, two case studies are explained expressing variable parts with UML Sequence Diagrams. The first one is Travel Itinerary System where several services are composed under one choreography specification. The second one is Adaptable Security System which comprises three choreography specifications with interrelated services. These case studies are specified using the metamodel and the XChor language constructs.

3.2.1 Case Study: Travel Itinerary System

Travel itinerary system is an online booking facility which organizes travelers' trip plans even for complex trips with multiple stops and changes. The system can optionally provide hotel and flight booking, car rental, booking of activities, and vacation packages. Vacation packages contain at least hotel and flight bundle and optionally additional activities. Traveler can choose any booking type (flight and/or hotel), arrival and departure dates of his/her trip, the destination place and other details such as traveling with pets. Travel agency gathers convenient hotel and flight options from available hotels and airlines which are presented with detailed information to the traveler with regard to his/her selected booking type. According to traveler's choice, additional activities, cruise options and advantageous vacation packages are offered. Traveler can choose one among them and can book hotel and/or flight. After getting booking confirmations, travel agency sends the trip plan to the traveler with all required information. In case of any booking problem, travel agency sends a message indicating the booking cancellation and directs to him/her rearrange her/his itinerary.

A UML diagram describing how the travel itinerary system works is depicted in Figure 3.1. Traveler, Travel Agency, Hotel, Airline, Cruise, Car Rental and Activity Provider are the main actors of the Travel Itinerary System. Hotel and Airline booking are processed with regard to selection of booking type by the Traveler. Therefore,

the parts of flow belonging to airline or hotel are covered by if clauses. Likewise, optional cruise, car rental and/or activities are represented by if clauses in the flow.

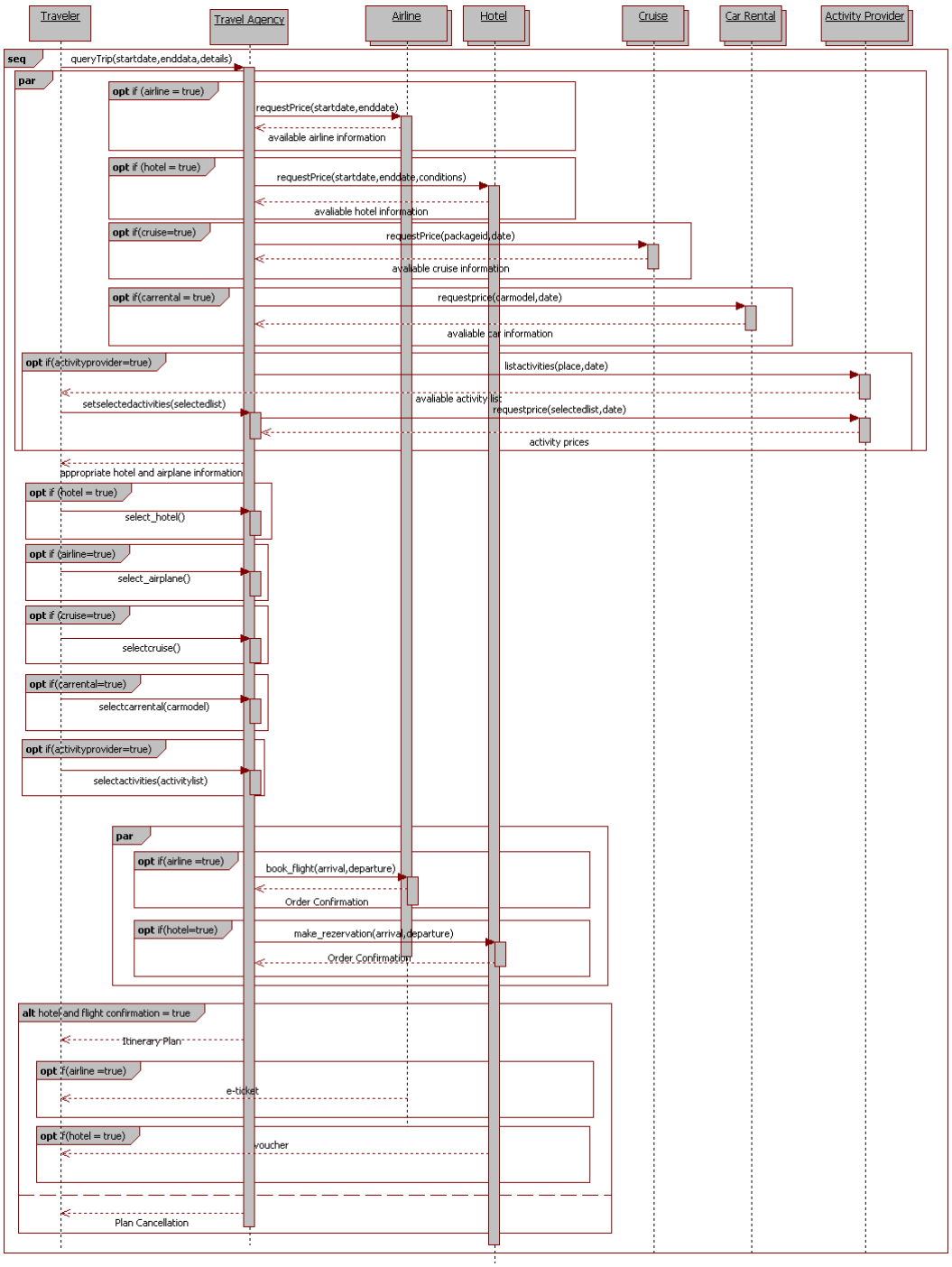

Figure 3.1: UML Sequence Diagram for Travel Itinerary System.

The flow of the booking changes with regard to booking type; only flight, only hotel,

or flight and hotel. Therefore, booking type causes travel itinerary system behavior change whose variants are the hotel and the airline. Besides, inclusion of additional features such as cruise, car rental, and activities alters the way services interact with each other. These are also variants of the travel itinerary plan.

3.2.2 Case Study: Adaptable Security System

The Adaptable Security System is an authentication system residing between customers and third party applications or institutions that support different authentication types of data, including software and hardware (biometric device) parts. The system has the ability to be integrated and applied on a military installation or to a banking system, which requires fulfilling different stakeholder needs. Applicability to different stakeholder systems requires different functionality support and behavior. In other words, the adaptable security system has the ability to comprise all structures and compositions in order to fulfill application needs for different stakeholders.

The system includes two basic functions; user enrollment and verification which can be offered by offline or online by a third party authority such as web services or certain devices like: PDA, PC, ATM, or mobile phone. The third party authority gets different types of data as required user credentials: (1) user name and password, (2) user name and password with instant mobile text, (3) e-sign, (4) biometric data; fingerprint, finger vein, and/or iris. According to the verification result, the system will allow or ban users entering the integrated application.

Device support is important as different devices have different capabilities. ATM, PDA and mobile phone can be used with (1), (2) and (3). PC supports (1), (2), (3) and (4). Therefore, the system should change authentication processing functions according to used devices. Moreover, users can combine different data types in order to authenticate; for instance in the PC case user can enter his/her user name, password and biometric data. Feeding the system with different possibilities of user credentials should be considered, as (i) internal encryption algorithm is affected by changed parameters and (ii) biometric processing algorithm usage should be configured according to the type of biometric data.

A UML interaction diagram is used to define the various configurations for verification of a user as shown in Figure 3.2. Adaptable Security System, as itself a choreography, interacts with credential manager and alert choreographies colored with dark blue, the others are treated as orchestration or atomic services. Adaptable Security System choreography realizes two functionalities; verify and enroll. Verification of a user generally includes four steps: (i) processing of his/her credentials; (ii) encryption of processed data, (iii) comparison of encrypted processed data and pre-existing encrypted data, (iv) showing the verification result to the end user. Processing of third step changes with regard to authentication types; online or offline.

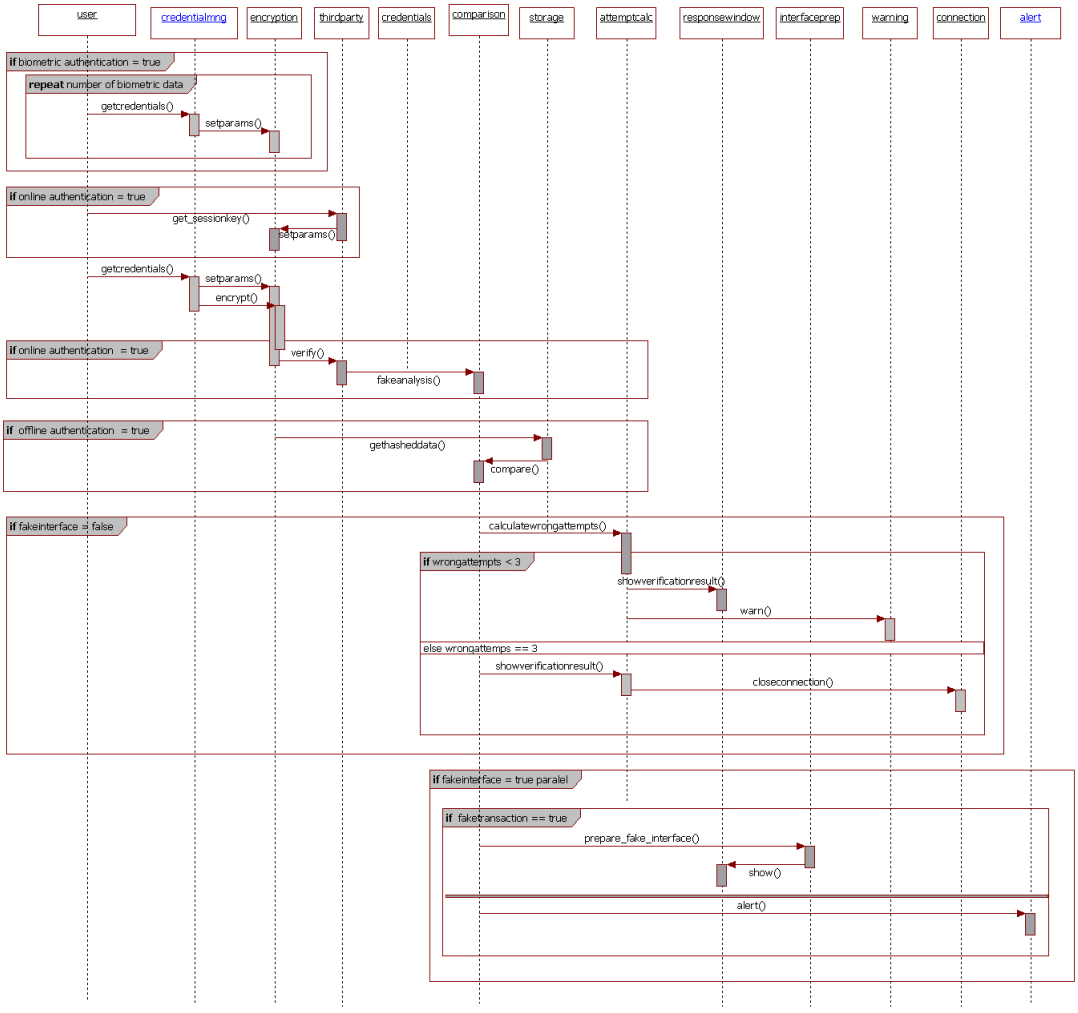

Figure 3.2: UML Sequence Diagram for User Verification in Adaptable Security System.

Online verification requires gathering comparison result from a third party application, whereas offline verification needs data comparison within the device by establishing connection to storage. Likewise, fake transaction support of the system changes the way to respond user, in our case the system prepares a fake interface and alerts bank or police station. Credential manager choreography gathers user's biometric data from biometric readers and extracts biometric features, which is reused by verification and enroll functions of adaptive security system. Alert choreography can take pictures, record a scene video and send them to bank and/or police station along with place and date information in case of emergency.

Given different user credential types such as biometric authentication, supported authentication modes (online and/or offline), transaction types (real or fake transaction) the system's behaviors need to be configured differently. As such we will need to compose different choreography and orchestration specifications. The composition of the services depends on the selected items of the orchestration and choreography elements. The selected orchestration elements will typically have an impact on the choreography specification. Likewise it is important that this be done in a consistent way.

3.3 A Metamodel for Variability Management in Choreography

To enable integration of orchestrations and atomic services in the scope of choreography, we propose a metamodel in which atomic services and orchestrations are evaluated under the service concept. The main difference between specifications of orchestration and atomic service comes from revealing external behavior to service environment. That is, an orchestration can define its external interactions with other services if required. Moreover, there is no constraint that an atomic service can not specify its interactions. Therefore, atomic services and orchestrations are treated as services in our metamodel. The metamodel basically enables to define choreographies and services, to specify variability of each one and to integrate these variabilities in order to provide a consistent collaboration. Figure 3.3 depicts an overview of service and choreography relations based on our metamodel so as to support interface and composition variability. Two main blocks are depicted; choreography and

service.

Internal, external variations and internal variability bindings for configuration purposes are specified for choreography. On the other hand, only external variations can be defined for services. Choreography and service interfaces without variation are identified which fulfill all possible functional requirements and then are configured by variations. Choreography interface is merely configured with regard to its own variability specification, whereas service interface is configured by both its own variability and variability specification of choreography that takes part in. Configuration of a service is achieved by activating/deactivating functions and setting/unsetting parameters. With this mechanism, different choreographies utilize different interfaces of the same service which brings service reusability.

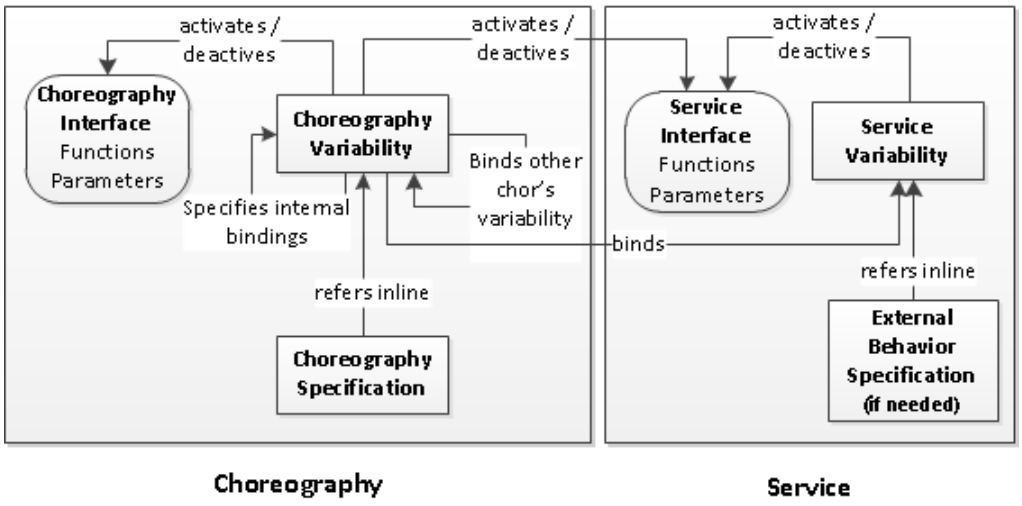

Figure 3.3: Overview of the approach based on the Metamodel.

Choreography variation leads to proper bindings of variations of other choreographies and services via mapping to provide interacting interface consistency. Choreography and external behavior specification of services include inline references of their own variability to point out the changeable parts. By this way, choreographies and services include a set of possible required behavior in order to fulfill different composition needs, which enables reuse of choreography and services.

The analogy between metamodel and model instance with metamodel of XChor is represented in Table 3.1.

Table 3.1: Mapping of Metamodel and XChor Metamodel Concepts

Metamodel	XChor Metamodel
Model	XChor Models: choreography, service/choreography interface, configuration interface model
Model instance	choreography specification, service interfaces, choreography interfaces, configuration interfaces

XChor metamodel with all construct are depicted in Figure 3.4. The metamodel is separated logically in three blocks for understandability purposes; Choreography Specification, Choreography to Variability Mapping and Variability Specification.

3.3.1 Variability Specification

The right most part of the metamodel in Figure 3.4 presents the variability specification constructs. The variability metamodel has been defined based on variability needs of service-orientation and existing variability metamodels in the literature, which comprises standard entities for modeling variability as described in Chapter 2, Section 2.3.1.4.

Choreographies and services reveal their hidden variability as internal and expose them as external variability. Internal variation points are invisible to outer context. Whereas, external variation points are explicit to users of choreography and services in order to be referenced, utilized and configured with a set of variants. A special variation point, configuration variation point (CVP) is responsible for reducing complexity of internal variation bindings. It provides a high level understanding for configuration purposes while hiding details of how the internal bindings are done.

Variants play the role of activating/deactivating functions and/or setting parameters of a function belonging to a choreography or a service interface for configuration purposes. For different variation point relationships, constraints provide a mechanism to establish a convenient binding and selection by defining numerical and logical constraints.

3.3.2 Choreography Specification

The left most part of the metamodel represents the elements to specify a choreography composition and interfaces of choreography and services. Choreography comprises a set of interacting services and other choreographies and identifies composability rules via service interactions. Here service interaction specifies the way how the services collaborate by means of atomic and composite interactions. Composite Interaction defines an interaction between services and choreographies with/without a guard. It can be either a selection of an interaction among others (SelectInt), repetition of a set of interactions (RepeatInt), parallelization of a set of interactions (ParallelInt) or flowing down in a sequence (SequenceInt) with a basic fault specification. The building block of a composite interaction is the atomic interaction which is a specification of a basic interaction between two services with variability attachment.

As choreography has its own interface, choreography specification comprises a set of interacting service and choreography interfaces. Choreography and service interfaces expose a set of functions without variability specifications. Different from a service, a choreography interface declares required functions from other services and choreographies.

3.3.3 Choreography to Variability Mapping

The middle part of the metamodel represents the concepts to define the mapping between choreography and variability constructs. Mainly these constructs are responsible for configuration of interfaces, establishment of variability associations and indicating variability references in composition.

Variability Configuration Model of service and choreography includes a set of variation points, constraints among them and service interactions (for services only). *Variability Association* facilitates choreography to identify proper bindings of utilized service and choreography variability. The need for associations comes from choreography variability to support different compositions and provide consistent variability bindings. For this purpose, firstly variation points are mapped and then each variant of related choreography variation point is mapped to that of service or that of utilized

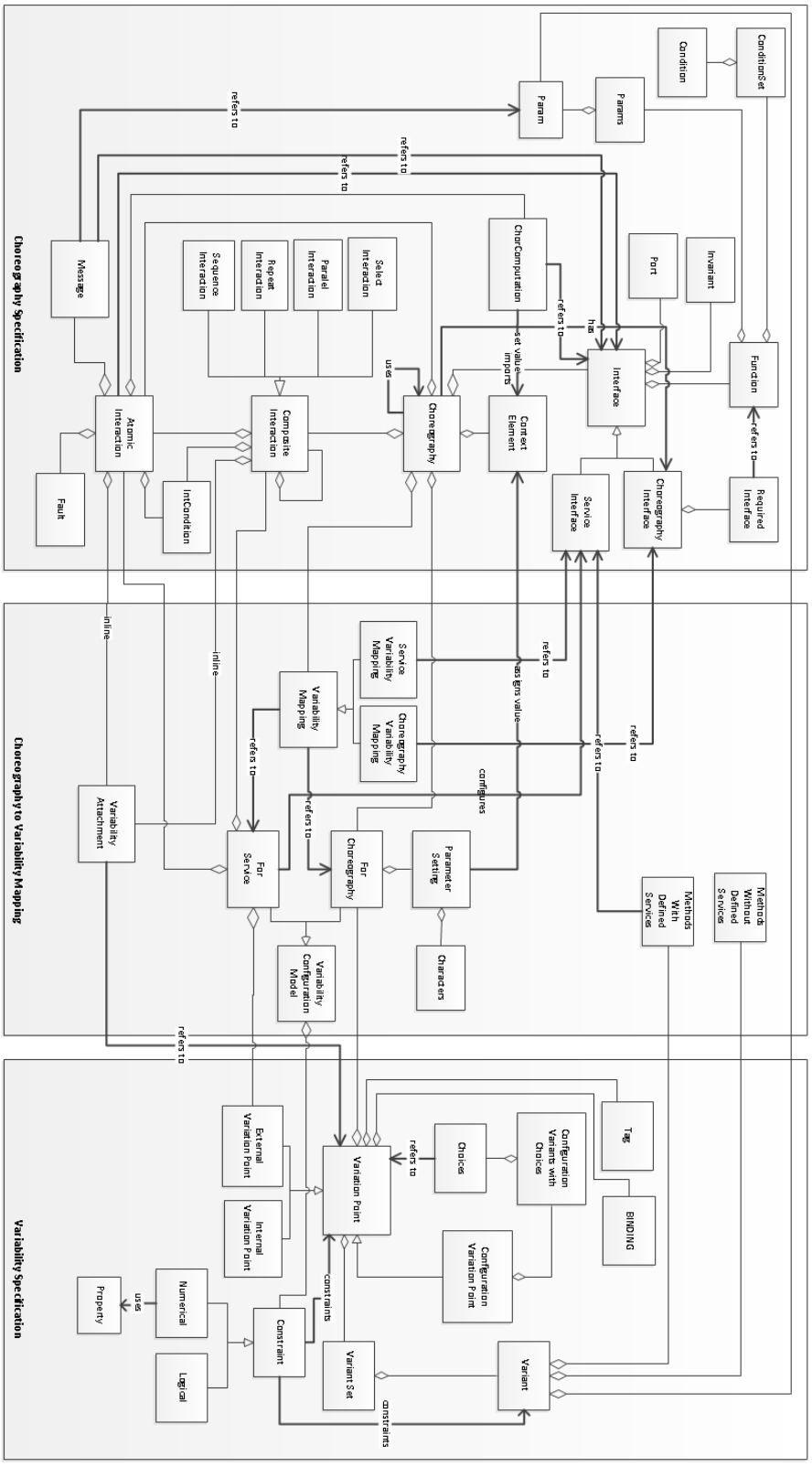

Figure 3.4: XChor Metamodel for Variable Choreography Specification.

choreography variation point.

Methods And Parameter Activation for Configuration provides a configuration mechanism to define method activation/deactivation and parameter setting/unsetting of the referred service interface.

Variability Attachment specifies conditions of variation point and variant selections used in choreography composition, namely in composite and atomic interactions. Tagging with variability attachment specifications, the parts of the composition gains dynamicity that changes the behavior of choreography. When conditions are satisfied, the part is added to the final composition. The conditions including variation point and variants are: (i) one of the variants in a variant set is selected, (ii) all of the variants in a variant set are selected and (iii) some of the variants in a variant set are selected.

3.4 XChor Language

The metamodel that we have described in the previous section has been realized as a new domain specific language that we call XChor. XChor[122] has been implemented using Xtext[22] in the Eclipse development environment.

3.4.1 XChor Language Constructs

3.4.1.1 Variation Specification Constructs

Explanations of each variation specification construct are given in detail with Xtext specifications. Variation point specification constructs of XChor metamodel is depicted in Figure 3.5.

VarPoint is a representation of a variable property of an item which identifies one or more locations at which the variation will occur. VarPoint is an abstraction of three types of variation point: internal, external and configuration variation point.

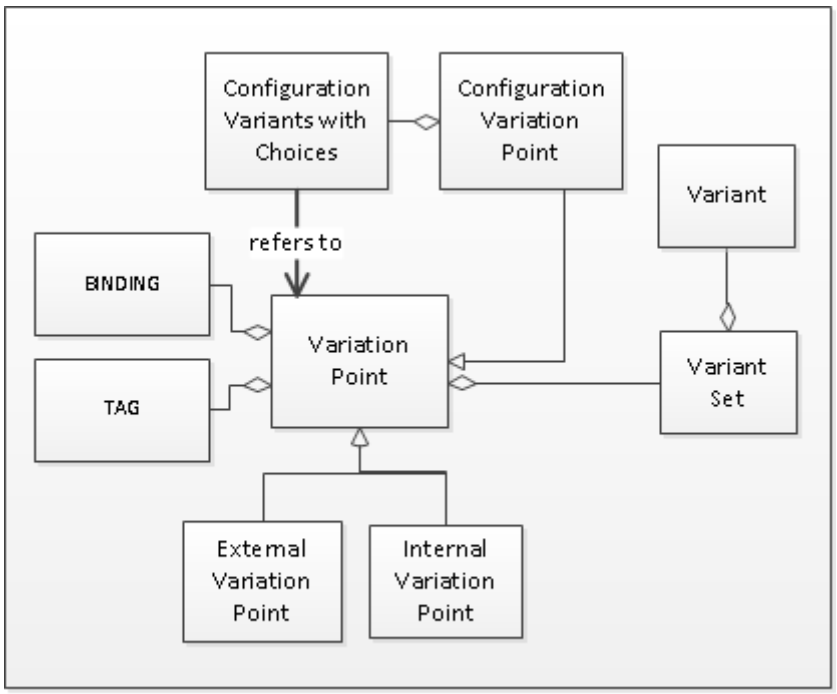

Figure 3.5: Variation Point Specification Constructs of XChor Metamodel.

```
VarPoint:
ConfigurationVarPoint | InternalVarPoint | ExternalVarPoint
;
```

InternalVarPoint is hidden from users of choreographies, orchestrations and atomic services. InternalVarPoint defines a variation point which is invisible to outer context so as to describe a variability with a VariantSet and specified binding time. Internal Variation points can be specified inside the configuration interfaces of choreography.

```
InternalVarPoint:
vt = "internalVP" name=ID ':'
variants = VariantSet
"bindingTime" btime =BINDING
;
```

ExternalVarPoint is explicit to users of choreographies, orchestrations and atomic services in order to reference, utilize and configure with a VariantSet and specified binding time. External Variation points can reside in configuration interfaces

of choreography and services. "externalVP" should be used while defining a variation point for choreography in a configuration interface and "vp" should be used for service variation in the configuration interface.

```
ExternalVarPoint:
( vt = "externalVP" | vt2 = "vp")  name=ID ':'
variants = VariantSet
"bindingTime" btime =BINDING
;
```

VariantSet is a set of defined set of variants grouped as mandatory, optional and alternative. Alternative variants are specified with minimum and maximum number of selections.

```
VariantSet:
{VariantSet} ("mandatory" (variants += Variant)* )?
("optional " (variants += Variant)*)?
("alternative " (variants += Variant)*
"(min:"INT",max:"INT")")?
;
```

Variant is a representation of a particular instance of a variable property. Variant is a variable definition of a variation point which can activate functions of services (MethodsWithoutDefinedServices) or its functions stated in the interface (MethodsWithoutDefinedServices) and/or can set a parameter (Function) to a function (Function) stated in service interface if required.

```
Variant returns Variant :
"variant" name = ID  ( (":activateMethods(" (
m1 = MethodsWithDefinedServices |
m2 = MethodsWithoutDefinedServices ) ")")?
(":setParameter(toFunct:" f = [Function] ",parameter:"
pars = Param (
```

```
";toFunct:" func += [Function]
",parameter:" fpars += Param)*
 ")" )?
)
;
```

Tag is an ID assigned to variation points whether they reside in composition or take part in configuring service interfaces via mapping. It can reside in composition (@composition) or take part in configuring service interfaces (@vconfservice) or take part in configuration variation point realization (@vconfrealization)

```
Tag:
"@" name = ID
;
```

Binding is a set of specified times indicating when a variation point can be bound to a set of variant.

```
BINDING:
devt = "devtime"| derv = "derivation"
| comp = "compilation" | link = "linking"
| strt= "start-up" | runt ="runtime"
;
```

In configuration interfaces of orchestration and atomic services, **vp** is used instead of **externalVP**. <vpname> indicates a unique name of the variation point, <varname> is also a unique name representing variant descriptor. <number> is an integer used to specify the minimum or maximum amounts of the variant to be selected within alternative variants. <binding> indicates the time when the variation point is bound to one or more of its variants and can be one among the set; devtime, derivation compilation, linking, start-up, and runtime.

Examples and syntaxes of internal and external variation points are given in Table 3.2. The example variation points are taken from the configuration interface of the

adaptive security system choreography. Developers can choose to internalize or externalize selection of authentication type supported by the adaptable security system choreography. Authentication type (i_auth_type, auth_type) is a variation point which changes the behavior of the choreography, hence the composition of the system. It has a mandatory variant which is username and password (line 4). Two optional variants are one time password (line 6) and e-sign (line 7).

Table 3.2: Internal and External Variation Point Syntaxes and Examples

```
1  internalVP  vpname :
2    mandatory
3      variant  varname
4    ...
5    optional
6      variant  varname
7    ...
8    alternative
9      variant  varname
10   ...
11   (min: number ,max: number
         )
12   bindingTime  binding
```

XChor Language - Internal Variation Point Syntax

```
1  internalVP i_auth type:
2    mandatory
3      variant username_passw
4    optional
5      variant onetimepassw
6      variant esign
7    alternative
8      variant fingerprint
9      variant fingervein
10     variant iris
11     variant face
12     (min:1,max:2)
13   bindingTime devtime
```

XChor Language - Internal Variation Point Example

```
1  externalVP  vpname :
2    mandatory
3      variant  varname
4    ...
5    optional
6      variant  varname
7    ...
8    alternative
9      variant  varname
10   ...
11   (min: number ,max: number
         )
12   bindingTime  binding
```

XChor Language - External Variation Point Syntax

```
1  externalVP auth_type:
2    mandatory
3      variant username_passw
4    optional
5      variant onetimepassw
6      variant esign
7    alternative
8      variant fingerprint
9      variant fingervein
10     variant iris
11     variant face
12     (min:1,max:2)
13   bindingTime devtime
```

XChor Language - External Variation Point Example

Alternative variants range from fingerprint (line 9), finger vein (line 10), iris (line 11) and face (line 12) among which minimum one and maximum two are selected. All these variants can be bound at development time (line 14).

ConfigurationVarPoint is a higher-level variation point including type and information about its variation points which are realized by low level variation points. It maps its variants to a set of internal variation points with their variant selections. It can be either internal or external which is specified by the "vartype" keyword. It defines a set of variants (VariantSet) and their realization (ConfVariantWithChoices), default variant (Variant) selection and binding time (BINDING).

```
ConfigurationVarPoint returns ConfigurationVarPoint:
"configuration" (
{InternalVarPoint} name=QualifiedName ':'
 "varType"  vt = "internalVP" |
{ExternalVarPoint} name=QualifiedName ':'
 "varType"  vt = "externalVP" )
(variants = VariantSet)
("realization" rea = STRING)
((confvariants += ConfVariantWithChoices)+ )
("defaultVariant" defaultVariant = [Variant])
("type" type= CONFTYPE
 "bindingTime" btime = BINDING )
;
```

ConfVariantWithChoices is a variant of a configuration variation point including a set of choices for realization.

```
ConfVariantWithChoices:
"confvariant" name = ID "mapping"
(choices += Choice)+
;
```

Table 3.3: Configuration Variation Point Syntax

```
configuration <vpname>:
  varType <externalVP | internalVP>
  mandatory
    variant <varname>
                ...
  optional
    variant <varname>
                ...
  alternative
    variant <varname>
                ...
    (min:<number>,max:<number>)
  realization <explanation>
  confvariant <confvarname> mapping
    VPName <referencedvpname> selectedVariants(<refvarname_1>,
        <refvarname_2>,...)
                ...
            ...
  defaultVariant <oneofconfvarname>
  type <type>
  bindingTime <binding>
```

Choice. It is a selection definition of a variation point among defined ones and related selected variants for the realization of a configuration variation point. Minimum and/or maximum number of variant selections can optionally be specified.

```
Choice:
"VPName" vp = [VarPoint] "selectedVariants(
" (vars += [Variant])+  ("; min:" INT)? (", max:" INT)?
 ")"
;
```

Syntax of the configuration variation point is given in Table 3.3 where <vpname> indicates a unique name of the configuration variation point. <confvarname> and <varname> are also unique names representing variant descriptors. <number> is an integer used to specify minimum or maximum counts of a variant to be selected within

alternative variants. <explanation> is a string clarifying realization of the configuration variation point. <referencedvpname> is a reference of an already defined variation point. <refvarname_1>,<refvarname_2>,... is a set of variants of referenced variation point. <oneofconfvarname> is one of the variants defined for configuration variant as default selected variant. <type> depicts the aim of configuration which can be substitution or parameterization. <binding> indicates the time when the variation point is bound to one or more of its variants and can be one among the set; devtime, derivation compilation, linking, start-up, and runtime.

Table 3.4: Configuration Variation Point Example

```
1  configuration authentication_type:
2    varType externalVP
3    optional
4      variant userinfo
5      variant biometrics
6    realization "it is realized by i_encryption_parameters and
         i_auth_type variability points"
7    confvariant userinfo mapping
8      VPName i_encryption_parameters selectedVariants(
           defaultparams)
9    confvariant biometrics mapping
10     VPName i_auth_type selectedVariants(fingerprint
           fingervein iris face; min:1, max:1)
11     VPName i_encryption_parameters selectedVariants(setparams
           )
12   defaultVariant userinfo
13   type parameterization
14   bindingTime devtime
```

authentication type configuration variation point is indicated as an external variation given in Table 3.4. The variation point hides encryption parameter and authentication type variation point binding logic and presents a high level configuration structure. It has two optional variants specified as (lines 4-5); “userinfo” and “biometrics”. “userinfo” variant is realized (line 8) by selection of “defaultparams” variant of “i_encryption_parameters” variation point. For “biometrics”, the realization requires two selections at the same time: (i) minimum one variant among “fingerprint fin-

gervein iris face" set should be selected from "i_auth_type" variation point (line 10) and "setparams" variant of *i_encryption_parameters* variation point (line 11). Default variant of the "authentication_type" configuration variation point is "userinfo" (line 12). Configuration type is parameterization and it is bound at development time depicted as "devtime" (line 14).

Constraint is a description of a relation among variation points and variants, as an abstraction of two types of constraints: LogicalConstraint and NumericalConstraint depicted in Figure 3.6.

```
Constraint:
LogicalConstraint | NumericalConstraint
;
```

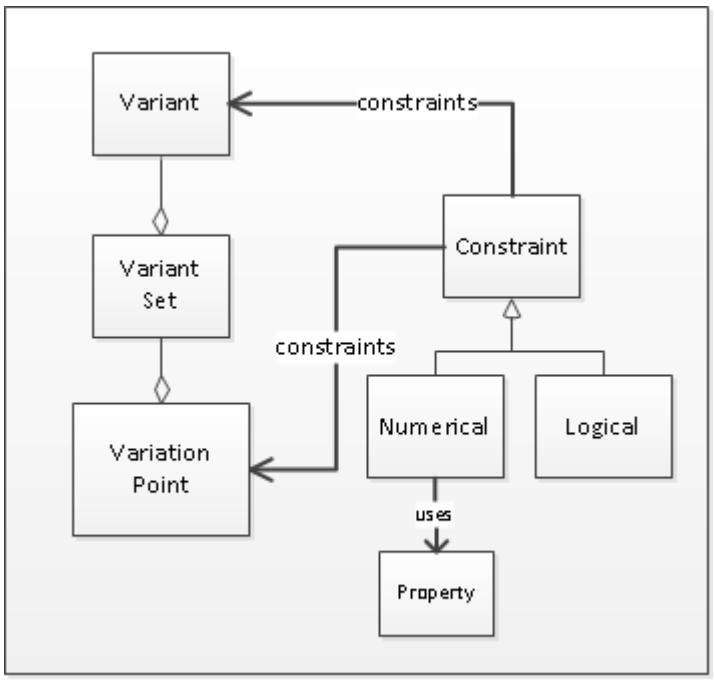

Figure 3.6: Constraint Specification Constructs of XChor Metamodel.

LogicalConstraint. It is a definition depicting a constraining relationship in which a variation point and/or related variants decide another variation points and/or its selected variants status as either excluded, implied, required or negated. Logical relationships are requires, excludes, negates and implies. Logical relationships are applied:

- Between two variation points.
- Between a variation point and a variant which is not related with the variation points. In other words, let's assume a variation point vp1 that has v1 and v2 as variants and vp2 has v3 and v4. A logical relationship can be represented between vp1 and v3 and/or vp4 as well as vp2 and v1 and/or v2.
- Between two variants.

```
LogicalConstraint:
( p1 = [VarPoint] (p2 = [Variant])? ) c =CONST
  p3 = [VarPoint] (
"selectedVariants("(vars += [Variant])+
(", min:" INT)? (", max:" INT)? ")"
)?
;
```

The syntax and an example of logical constraint are given in Table 3.5.

Table 3.5: Logical Constraint Syntax and Example

```
1 <vpname1> <varname1> <logicalconstraint> <vpname_2>
    selectedVariants(<varname_2-1>,<vpname_2-2>,...,min:<number
    >,max:<number>)
```

XChor Language - Logical Constraint Syntax

```
1 i_auth_type face requires i_auth_mode selectedVariants(mode
    _online)
```

XChor Language - Logical Constraint Example

NumericalConstraint is a definition depicting a constraining relationship in which a variation point and related variant result in an assignment of a value to another variation point and related variant or to a property with expressions (greater than, less than, greater than or equal, less than or equal, equal, not equal). The syntax and an example of logical constraint are given in Table 3.6.

```
NumericalConstraint:
p1 = [VarPoint] p2 = [Variant] nconst = NUMCONST
(p11 = [VarPoint] p3 = [Variant] | p4 =Property)
exp = EXPR (STRING | p5 = Property
| "valueOf{" (vars += [Variant])* "}")
;
```

Property is a specification of a system property with its name.

```
Property:
name = ID
;
```

Table 3.6: Numerical Constraint Syntax and Example

```
1 <vpname_1> <varname_1> const  (<vpname_2> <varname_2> |
     <property>) <expr> (<string> | <property>)
```

XChor Language - Numerical Constraint Syntax

```
1 i_auth_mode mode_online const protocol = "https"
2 i_auth_type esign const i_encryption_parameters defaultparams
     = valueOf username_passw esign
```

XChor Language - Numerical Constraint Example

<vpname_1> and <vpname_2> are already defined variation points. <varname_1> is a variant of <vpname_1>, whereas (<varname_2-1>,<vpname_2-2>,.. are variants of <varname_2>. <number> is an integer used to specify minimum or maximum amounts of variant to be selected within alternative variants. <logicalconstraint> is a constraint which can be one among the set, requires, excludes, implies, and negates. <property> is a string depicting a system property.

One of the authentication types, face needs authentication mode to be online. In other words, if authentication of users is done through face recognition, the system should operate online. This constraint is represented as numerical logical constraint.

If the system is operated online, the protocol should be “https” which is depicted as a numerical constraint in line 1. Username, password and e-sign should be set as the default parameters of encryption if e-sign is used for user authentication, and is represented as a numerical constraint in line 2.

3.4.1.2 Choreography Specification Constructs.

Explanations of each choreography specification construct are presented in detail with Xtext specifications.

Choreography includes its configuration interface (VConfModelImport), imports interacting choreographies (ChorImport) and services (ServiceImport), defines shared variables (Context Elements), maps choreography and service variability over related variation point and variant specifications (VMMapping). It also defines the choreography composition with in-line variation point and selected variants as a guard to execute the piece of choreography (Composition). Choreography specification constructs of XChor metamodel is presented in Figure 3.10.

```
Choreography:
"choreography" name=ID
(vconfmodelimport = VConfModelImport)?
(cimports  += ChorImport)*
(simports += ServiceImport)+
("Context Elements" (contexts += ContextElement)*)?
("Choreography Variability Mapping"
(mappings += VMMapping)*)?
("Function" func += [Function] ":" comp += Composition)+
;
```

VConfModelImport is an import mechanism to include choreography’s configuration Interface. There can be more than one configuration Interface of the same choreography.

```
VConfModelImport:
```

```
"import configuration"
 importedNamespace = [VarConfigurationModel4Chor]
;
```

ServiceImport is an import mechanism to include utilized services (ServiceInterface) in choreography composition with specified service configuration interface (VarConfigurationModel4Service) if required. There can be more than one configuration Interface for the same service.

```
ServiceImport:
"import service" s = [ServiceInterface] (
 "with configuration"
 importedNamespace = [VarConfigurationModel4Service]
)?
;
```

ChorImport is an import mechanism to include other choreographies (ChorInterface) interacting with the current choreography.

```
ChorImport:
"use choreography" name = [ChorInterface]
;
```

ContextElement is a shared element definition used in choreography composition.

```
ContextElement:
name = QualifiedName (defaultvalue = INT | STRING | ID
| BOOLEAN)
;
```

ChorComputation is an assignment of the return value of a service (Interface) function (Function) to a ContextElement which is a shared variable of choreography.

```
ChorComputation:
"%comp" name = [ContextElement] "="
     s = [Interface] "." f = [Function] "%"
;
```

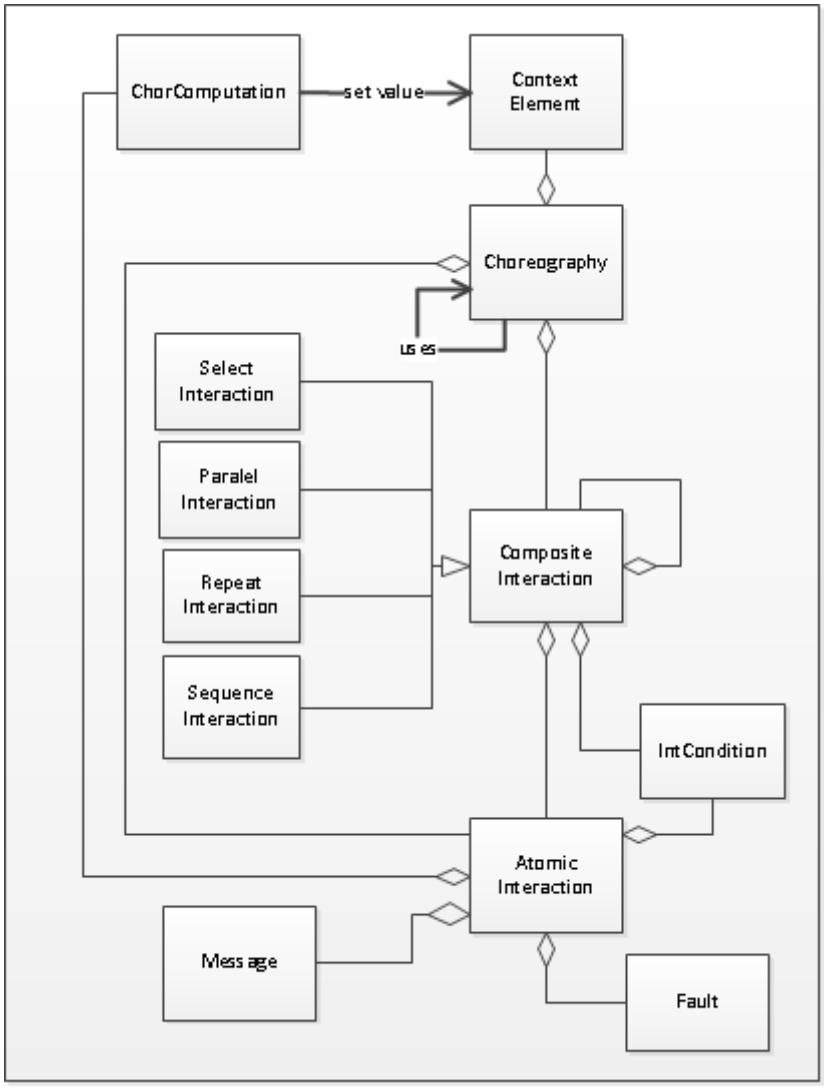

Figure 3.7: Choreography Specification Constructs of XChor Metamodel.

Composition is a definition of a set of interactions in order to realize a common goal via one or more atomic (AtomicInteractions) and/or composite (CompositeInteraction) interactions tangled with each other.

```
Composition:
(interactions +=
```

```
(AtomicInteraction | CompositeInteraction)
(WS interactions +=
( AtomicInteraction | CompositeInteraction))* )+
;
```

Message is a definition of message including set of parameters (Param), semantical description, referring service (Interface) and its function (Function).

```
Message:
"message" name = [Function]
("(" (par += [Param] ("," par += [Param] )*)? ")")
("refers" (service += [Interface]
  "." funct += [Function])* )?
("semantic (" s = STRING ")")?
;
```

IntCondition is a specification of a condition used to guard a part of an interaction. It can be either a definition of a condition with expression and numerical/non-numerical values or a specification of number.

```
IntCondition:
p1 = GUARDTEXT ((exp = EXPR (STRING | INT | ID
| BOOLEAN) ) | "times")?
;
```

AtomicInteraction is a specification of a basic interaction between two services with-/without variability attachment (VariabilityAttachment). It is written

- with/without a guard condition (IntCondition),
- with depiction of source and destination services (Interface) with "send" or "receive" actions,
- with a message (Message),

- with/without a computation effect to a ContextElement (ChorComputation) and
- with other additional constructs.

If the action is "receive" from a set of services and one should be selected then "pickOne" is added. If more than one receive is accomplished from a source to a destination, then "multiple times" should be added. If the "send" action requires notification from destination, then "withNotification" is added. If an atomic action is limited with a duration, then the "wait" keyword with a time specification should be provided. When an atomic action wants to explicitly depict a fault when a problem occurs, a "fault" should be defined. If interaction is "send" that is willing to get a request from one of the available destinations with a limited duration, then "callingSequence" is defined with a sequence of destinations. If the interaction causes another interaction; sending the value of the computation to another service/services, then "referedDestinations" is defined. If the AtomicInteraction causes one or more changes in ContextElement's values, then a set of ChorComputation is defined.

```
AtomicInteraction:
(va = VariabilityAttachment)?
("guard (" guard = IntCondition")")?
(source = [Interface]
type = "send" "{" (destionation += [Interface])+ "} "
("in-sequence")? ("atomic")? ("viewer")? |
 destination = [Interface]
type = "receive"("from{"(rsource += [Interface])*"}")?
 ("multiple times")? ("pickOne")? )
(message += Message)
("stopmessage from" stopservice = [Interface] )?
("wait" (t = Time)? ("until" INT "messagescame")?)?
("inactivity-interval" inact = Time)?
("referedDestinations(" refpart += [Interface]
((",," refpart += [Interface])*)? ")" )?
("withNotification"
("(min:" min = INT ",max:" max = INT")")?)?
```

```
(f += Faults ("toreferrals")?)?
(comp +=ChorComputation)*
;
```

Syntaxes and examples of send and receive atomic interactions are presented in Table 3.7.

Table 3.7: Send and Receive Atomic Interaction Syntaxes and Examples

```
1  <service_A> send <service_B>  message <function>(<params>)
```

XChor Language - Atomic Interaction - Send Syntax

```
1  thirdparty send encryption  message setparams(parameters)
```

XChor Language - Atomic Interaction - Send Example

```
1  <service_A> receive from  <service_B>  message <function>(
       <params>)
```

XChor Language - Atomic Interaction - Receive Syntax

```
1  imageretrieval receive message extractfeatures(
       biometricdata)
```

XChor Language - Atomic Interaction - Receive Example

CompositeInteraction is a definition of an interaction between services with/without a guard (IntCondition) including a set of selection of an interaction among others (SelectInt), repeating a set of interactions (RepeatInt), parallelization of a set of interactions (ParalelInt) and flowing down in a sequence (SequenceInt).

```
CompositeInteraction:
("guard (" guard = IntCondition ")" )?
 ("precedent")? (
interaction = SelectInt |
interaction = RepeatInt |
interaction = ParalelInt |
```

```
interaction = SequenceInt)
("timeout" INT)?
;
```

SequenceInt is a definition of a sequence of a set of interactions between services which can be atomic (AtomicInteraction) or composite (CompositeInteraction) with-/without variability attachment (VariabilityAttachment). It is written in such a way that the block is started with "sequence". Interactions are surrounded with parenthesis. The syntax and an example of sequence interaction is presented in Table 3.8.

```
SequenceInt:
(va = VariabilityAttachment)?
"sequence ("
          (interactions +=
          (AtomicInteraction | CompositeInteraction))+
")"
;
```

Table 3.8: Sequence Interaction Syntax and Example

```
1 sequence (
2   <Composite or Atomic Interaction>
3   <Composite or Atomic Interaction>
4   ...
5   <Composite or Atomic Interaction>
6 )
```

XChor Language - Composite Interaction - Sequence

```
1 sequence (
2   thirdparty receive from encryption  message getconnection()
3   thirdparty send encryption  message setparams(parameters)
4 )
```

XChor Language - Composite Interaction - Sequence Example

SelectInt is a definition of a selection between a set of interactions among services

which can be atomic (AtomicInteraction) or composite (CompositeInteraction) with-/without variability attachment (VariabilityAttachment). It is written in such a way that the block is started with "select", interactions are surrounded with parenthesis. The syntax and an example of sequence interaction is presented in Table 3.9.

Table 3.9: Select Interaction Syntax and Example

```
select (
  <Composite or Atomic Interaction>
  <Composite or Atomic Interaction>
  ...
  <Composite or Atomic Interaction>
)
```

XChor Language - Composite Interaction - Select

```
select (
  sequence (
    itineraryplanner receive from hotel message
        rejectbooking(customeID)
    itineraryplanner send customer message showresult(
        bookingrejected)
  )
  sequence (
    itineraryplanner receive from hotel message
        acceptbooking(customerID)
    itineraryplanner send customer message makepayment(
        bookingID)
  )
)
```

XChor Language - Composite Interaction - Select Example

```
SelectInt:
(va = VariabilityAttachment)? "select"
 (cond = IntCondition)?
"("
           interactions +=
           (AtomicInteraction | CompositeInteraction)+
")"
```

```
;
```

RepeatInt is a definition of a repetition of a set of interactions between services which can be atomic (AtomicInteraction) or composite (CompositeInteraction) with an exit condition and with/without variability attachment (VariabilityAttachment). It is written in such a way that the block is started with "repeat" following a condition and a set of interactions are surrounded with parenthesis. The syntax and an example of repeat interaction is presented in Table 3.10.

```
RepeatInt:
(va = VariabilityAttachment)? "repeat"
cond = IntCondition
"("(interactions +=
(AtomicInteraction | CompositeInteraction))+ ")"
;
```

Table 3.10: Repeat Interaction Syntax and Example

```
1 repeat condition (
2    Composite or Atomic Interaction
3    Composite or Atomic Interaction
4   ...
5    Composite or Atomic Interaction
6 )
```

XChor Language - Composite Interaction - Repeat

```
1 repeat noofbiometricauthtype times (
2   imageretrieval receive message extractfeatures(
      biometricdata) refers imageretrieval.extractfeatures
3 )
```

XChor Language - Composite Interaction - Repeat Example

ParalelInt is a definition of a paralelization of a set of interactions between services which can be atomic (AtomicInteraction) or composite (CompositeInteraction) with-/without variability attachment (VariabilityAttachment). It is written in such a way

that the block starts with "paralel", and interactions are surrounded with parenthesis. The syntax and an example of parallel interaction is presented in Table 3.11.

```
ParalelInt:
(va = VariabilityAttachment)? "parallel ("
(interactions +=
(AtomicInteraction | CompositeInteraction))+ ")"
;
```

Faults is a system failure with its name and explanation and sends fault notification to corresponding senders. Termination condition can be specified if required.

```
Faults:
"fault(" fname1 = FAULTTYPES ("," fname2 += FAULTTYPES)*
(",terminateIf" number = INT "fails")? ")"
;
```

Table 3.11: Parallel Interaction Syntax and Example

```
1 parallel (
2    Composite or Atomic Interaction
3    Composite or Atomic Interaction
4   ...
5    Composite or Atomic Interaction
6 )
```

XChor Language - Composite Interaction - Parallel

```
1 parallel (
2   thirdparty receive from encryption  message getconnection()
3   credentials receive message getcredentials() refers
       credentials.getcredentials
4 )
```

XChor Language - Composite Interaction - Parallel Example

Interface is an abstraction of two types of interface: Choreography and Service

which depicts provided functionalities to be used by other choreographies and services whose constructs are presented in Figure 3.10.

```
Interface:
ChorInterface | ServiceInterface
;
```

ChorInterface is a definition of interface for a choreography, the opened face to other choreographies including invariants (Invariant), externalized functions (Function), port description (Port) and required interfaces from other choreographies (RequiredInterface).

```
ChorInterface:
"Choreography interface" name = QualifiedName
"of " chorname = ID
(invariants += Invariant)*
((functions += Function)+)
port += Port
("required interfaces"
(reqints += RequiredInterface )*)?
;
```

RequiredInterface is a definition of demanded functions from other choreographies separated by semicolon (;).

```
RequiredInterface:
"from" name = QualifiedName "function" "{"
( f += [Function]) ("," (f += [Function]))*
"}"
;
```

ServiceInterface is a definition of a service interface including invariants (Invariant), functions (Function) and port specification (Port).

```
ServiceInterface:
"Service interface" name = ID
((invariants += Invariant)*)
((functions += Function)+)
port += Port
;
```

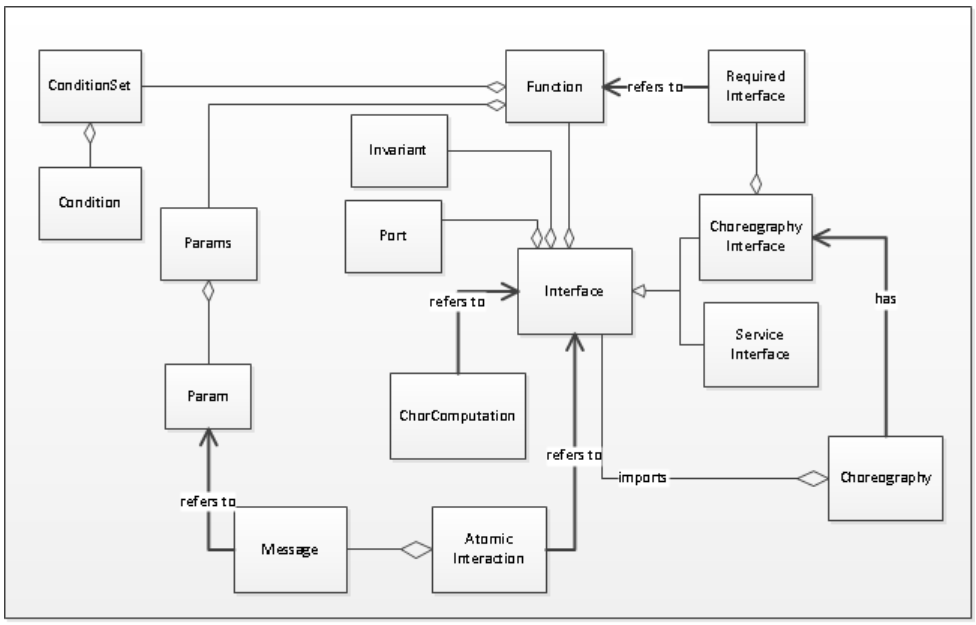

Figure 3.8: A part of XChor Metamodel for Interface Specification.

Function is a definition of a function including its name, pre and post conditions (ConditionSet), input parameters (Params) and its output (Param).

```
Function:
"function" name = ID
("precondition" precond += ConditionSet)?
("postcondition" postcond += ConditionSet)?
("input" ipars = Params)?
("output" opar = Param)?
;
```

ConditionSet is a set of conditions (Condition) composed via "or" and "and" logical relationships.

```
ConditionSet:
"(" c1 += Condition (("or" | "and") c2 += Condition )* ")"
;
```

Condition is a definition of a property of an object with a boolean value (true or false).

```
Condition:
name = ID ("==" | "!=") BOOLEAN
;
```

Invariant is a variable definition in choreography/service interface assigned to a boolean value which is valid throughout the choreography/service composition.

```
Invariant:
"invariant" name = ID "==" BOOLEAN
;
```

Port is a binding definition of other services with a defined host to current service/-choreography.

```
Port:
"portName" name = ID "binding" host = TEXT
;
```

Params is a set of parameters separated by comma and surrounded with parenthesis.

```
Params:
pars = "(" p1 = Param ("," p2 += Param)* ")"
;
```

3.4.1.3 Variation and Choreography Mapping Constructs.

Explanations of variation and choreography mapping constructs are given in detail with Xtext specifications and presented in Figure 3.9.

VarConfigurationModel is an abstraction of two types of configuration models namely VarConfigurationModel4Service for services and VarConfigurationModel4Chor for choreographies.

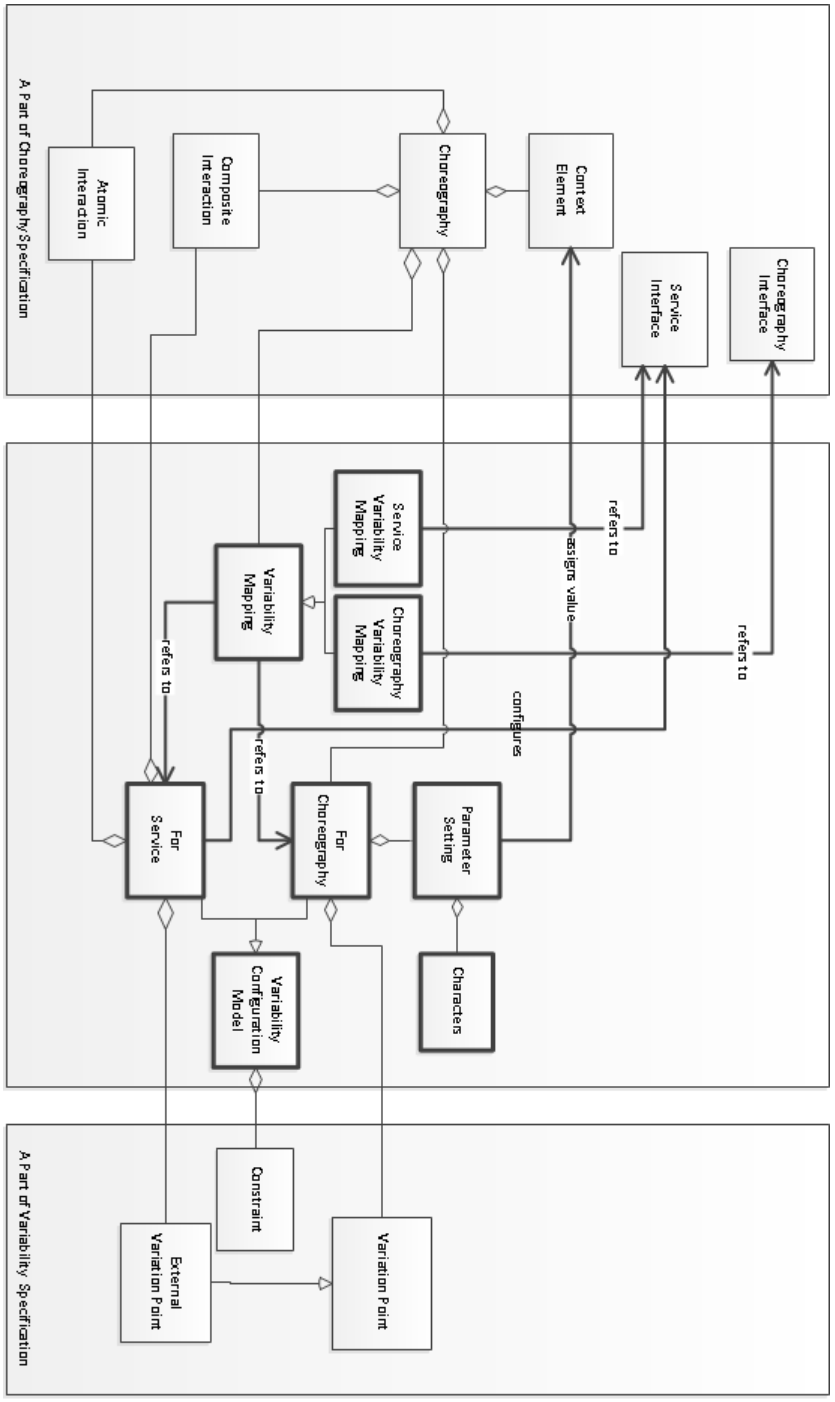

Figure 3.9: Configuration Model Specification Constructs of XChor Metamodel.

```
VarConfigurationModel:
VarConfigurationModel4Service | VarConfigurationModel4Chor
```

```
;
```

VarConfigurationModel4Service is a definition of a configuration interface for a service including

1. a set of external variation points (ExternalVarPoint) with a tag (Tag) defining the role of it if required,
2. constraints (Constraint) among external variation points and
3. its abstract process definition (Composition) which specifies external behavior of the service with other services.

```
VarConfigurationModel4Service:
"Configuration interface" name = ID "of service"
servicename = [ServiceInterface]
((tag += Tag)? vars += ExternalVarPoint)*
("Constraints"
(constraints += Constraint )*)?
("abstract process definition"
processdef = Composition
)?
;
```

VarConfigurationModel4Chor is a definition of a configuration interface for a choreography including

1. a set of internal, external and configuration variation points (VarPoint) with a tag (Tag) defining the role of them if required,
2. constraints (Constraint) among variation points and
3. parameter settings (ParameterSetting) which includes a set of defined parameters used in choreography.

```
VarConfigurationModel4Chor:
"Configuration interface" name = ID  "of choreography"
chorname = QualifiedName
((tag += Tag)?  vars += VarPoint)*
("Constraints"
(constraints += Constraint )*)?
("Parameter Settings"
(parametersetting += ParameterSetting )*)?
;
```

ParameterSetting is an assignment of a value to ContextElements residing in choreography specification.

```
ParameterSetting:
"parameter" name = [ContextElement] "=" ("
#ofSelectedVariants{"
    (vars += [Variant])+
"} Of "
vp = [VarPoint] |
"valueOf("
    var += [Variant] ("," vars += [Variant])*
")" )
;
```

VMMapping is an abstraction of two types of variability mapping VMServiceMapping for service and VMChorMapping for choreography. The syntax and an example of the construct are presented in Table 3.12 and Table 3.13.

```
VMMapping:
 VMServiceMapping | VMChorMapping
;
```

VMServiceMapping is a structural mapping from choreography variation to service

variation. First variation points are mapped and then each variant of related choreography variation point is mapped to that of service variation point.

```
VMServiceMapping:
"VP" vp = [VarPoint] "maps service"
service = [ServiceInterface]
"VP" svp = [VarPoint]
( "Variant"  vars += [Variant]
   "maps Variant"
(mvars += [Variant] )+)+
;
```

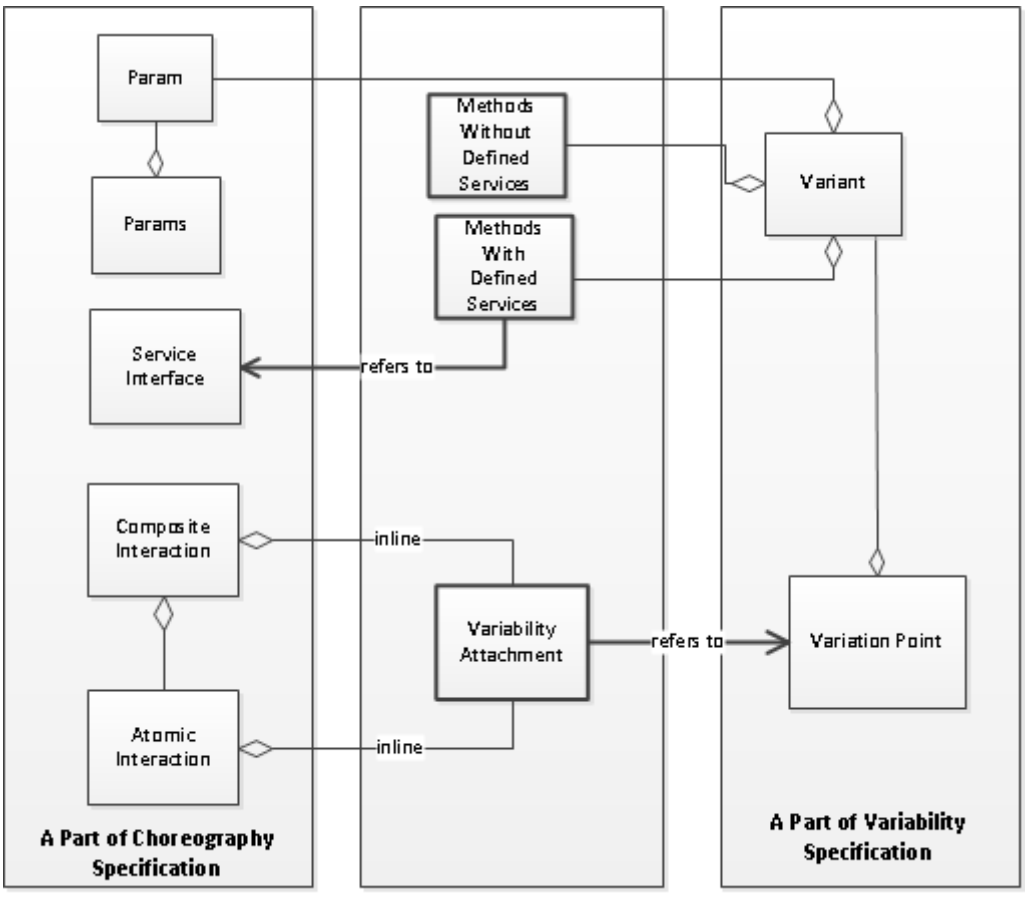

Figure 3.10: A part of XChor Metamodel for Variability Attachment Specification.

VMChorMapping is a structural mapping from choreography variation to utilized choreography variation. First variation points are mapped and then each variant of

related choreography variation point is mapped to that of utilized choreography variation point.

```
VMChorMapping:
"VP" vp = [VarPoint] "maps choreography"
chor = [Choreography]
"VP" cvp = [VarPoint]
( "Variant"  vars += [Variant]
   "maps Variant"
 (mvars += [Variant] )+)+
;
```

Table 3.12: VMMapping Syntax

```
1 VP <chorvpname> maps service <servicename> VP <servicevpname>
2   Variant <chorvarname_1> maps Variant <servicevarname_1>
3   ...
4 VP <chorvpname> maps choreography <chorname_1> VP <chorvpname
    _1>
5   Variant <chorvarname_1> maps Variant <chorname_1varname_1>
6   ...
```

In Table 3.12, <chorvpname>, <chorvpname_1> and <servicevpname> are variation points of current choreography, other interacting chorography and interacting service in composition respectively. <servicename> and <chorname> are names of interacting service and choreography. <chorvarname_1>, <chorname_1varname_1> and <servicevarname_1> are variants of <chorvpname>, <chorvpname_1> and <servicevpname> respectively.

In Table 3.13, adaptive security system choreography associates its internal variation points and related variants to those of utilized services' in order to configure service interface variability. The association between lines 1-3 ensures that when "i_encryption_parameters" variation point is bound to one of its variants, "encryption_params" variation point of encryption service is bound accordingly to provide a

consistent interaction. With this, when defaultparams is selected, encryption service interface is configured with regard to withdefaultparams variant (line 2).

Table 3.13: VMMapping Example

```
1  VP i_encryption_parameters maps service encryption VP
       encryption_params
2    Variant defaultparams maps Variant withdefaultparams
3    Variant setparams maps Variant withparams
4  VP i_transaction_type maps service comparison VP analysis
5    Variant faketransaction maps Variant fake
6    Variant realtransaction maps Variant real
7  VP i_auth_type maps service thirdparty VP user_device
8    Variant username_passw maps Variant ATM Mobile PDA PC
9    Variant onetimepassw maps Variant ATM Mobile PDA PC
10   Variant esign maps Variant ATM Mobile PDA PC
11   Variant fingerprint maps Variant PC
12   Variant fingervein maps Variant PC
13   Variant iris maps Variant PC
14   Variant face maps Variant PC
```

VariabilityAttachment is a definition of an attachment to choreography composition in order to define the conditions of variation point and variant selections. Relationships between variation point and variants used are:

- "ifOneSelected" if one of the variants in a variant set is selected.
- "ifAllSelected" if all of the variants in a variant set is selected.
- "ifSelected" if some of the variants in a variant set is selected.

"excl:"is used when a set of variants needed not to be selected. The composition segment tagged with VariabilityAttachment is added to the composition if the selections are realized.

```
VariabilityAttachment:
"#vp" vp += [VarPoint]
```

```
("ifOneSelected(" | "ifAllSelected(" | "ifSelected(")
(vs += [Variant])+
 (";excl:" (vsexc += [Variant])+)? ")"
(("and" | "or") vp2 += ID
("ifOneSelected(" | "ifAllSelected("| "ifSelected(")
(vs2 += [Variant])+
 (";excl:" (vsexc2 += [Variant])+)? ")" )*
"#"
;
```

Table 3.14: Variability Attachment Syntax

```
1 vp <vpname> <condition> (<varname_1> <varname_2> ...
    <varname_n>) Composite or Atomic Interaction
```

The syntax of variability attachment is presented in Table 3.14 where <condition> can be one of among “ifOneSelected”, “ifAllSelected” and “ifSelected”, designating the selection condition of variants. <varname_1> <varname_2> ... <varname_n> is a set of variants of <vpname>.

Table 3.15: Variability Attachment Example

```
1 vp i_auth_type ifOneSelected( fingerprint fingervein iris)
    Interaction1
2 vp i_auth_mode ifSelected(mode_online) Interaction2
3 vp i_auth_mode ifAllSelected(mode_online model_offline)
    Interaction3
```

In Table 3.15, when at least one of the authentication types; fingerprint, fingervein and iris is selected, then Interaction1 resides in the composition (line 1). If authentication mode is online, then Interaction2 is taking part in the flow (line 2). Only if

online and offline modes are selected at the same time, Interaction3 will be in the composition (line 3).

MethodsWithDefinedServices is a set of functions (Function) with related services separated by comma.

```
MethodsWithDefinedServices:
"service:" s = [ServiceInterface]
",funct:" funct = [Function] ("," functs += [Function])*
("; service:" s2 += [ServiceInterface]
",funct:" funct2 += [Function]
("," functs2 += [Function])*)*
;
```

MethodsWithoutDefinedServices is a set of functions of its own separated by comma.

```
MethodsWithoutDefinedServices:
funct = [Function] ("," functs += [Function])*
;
```

3.4.2 XChor Models

XChor Language facilitates to create three different models which covers different parts of the metamodel; configuration interface, choreography, service and choreography interfaces. The coverage of three model is depicted in Figure 3.4. The basic elements of XChor under three model to cope with variability in choreography is shown in the following subsections.

3.4.2.1 Configuration Interface

Configuration interface model covers service and choreography variability specifications internally and externally to depict possible abilities, to configure others and to be configured by others. To depict possible abilities, a choreography can specify internal,

external and configuration variation points, whereas services can only depict external variation points. The external ones are used to be configured by choreographies and services. Capabilities to configure its own interface or other services' interfaces as activating/deactivating and setting/unsetting parameters are also specified in this model. Numerical or logical constraints among variability specifications are included.

In Table 3.16, Table 3.17, and Table 3.18 different user authentication types such as biometric authentication, supported authentication modes (online and/or offline), transaction types (real or fake transaction) are the system's behaviors that need to be configured differently. Each is treated as variability in configuration interface of adaptive security system choreography.

To enable authentication variability, both types of authentication and parameters used in encryption function are changed with regard to the usage of biometrics or not. For this purpose, an external configuration variation point named as "authentication_type" and two internal variation points "i_auth_type" and "i_encryption_parameters" are defined. Binding of "authentication_type" configures consistent bindings of "i_auth_type" and "i_encryption_parameters".

In Table 3.16, "i_auth_type" is specified with "internalVP" keyword (line 5). "username_passw" is a mandatory variant, whereas "onetimepassw" (line 9) and "esign" (line 10) are optional. At least one and at most two variants can be selected among the following alternatives: "fingerprint" (line 12), "fingervein" (line 13), "iris" (line 14), and "face" (line 15). The binding time of this variation point is runtime (line 17).

In Table 3.17, "authentication_type" is specified as external (line 8). The variation point has two optional variants specified (lines 11-12); "userinfo" and "biometrics". "userinfo" variant is realized (line 15) by selection of "defaultparams" variant of "i_encryption_parameters" variation point. For realization of "biometrics" variant two variation bindings should be done simultaneously. Minimum one variant among "fingerprint fingervein iris face" set should be selected from "i_auth_type" variation point (line 17) and "setparams" variant of "i_encryption_parameters" variation point (line 18) should be bound. Default variant of the "authentication_type" configuration variation point is "userinfo" (line 8). Configuration type is parametrization and it is bound at development time represented as "devtime" (line 21).

Table 3.16: Configuration Interface of adaptive security system

```
Configuration interface vconf_adaptablesecuritysystem of
    choreography adaptablesecuritysystem

  //determines number of different biometric authentication
      types
  @composition
  internalVP i_auth_type:
    mandatory
      variant username_passw
    optional
      variant onetimepassw
      variant esign
    alternative
      variant fingerprint
      variant fingervein
      variant iris
      variant face
      (min:1,max:2)
    bindingTime runtime

  //determines authentication mode
  @composition
  internalVP i_auth_mode:
    alternative
      variant mode_online:activateMethods(service:thirdparty,
          funct:getconnection,savehasheddata,verify)
      variant mode_offline:activateMethods(service:storage,
          funct:gethasheddata)
    (min:1,max:1)
    bindingTime devtime

  //determines transaction type
  @composition
  internalVP i_transaction_type:
    optional
      variant realtransaction
      variant faketransaction
    bindingTime devtime
```

Table 3.17: Configuration Interface of adaptive security system-contd'

```
internalVP i_encryption_parameters:
  alternative
    variant defaultparams
    variant setparams
    (min:1,max:1)
  bindingTime runtime

configuration authentication_type:
  varType externalVP
  optional
    variant userinfo
    variant biometrics
  realization "it is realized by i_encryption_parameters
     and i_auth_type variability points"
  confvariant userinfo mapping
    VPName i_encryption_parameters selectedVariants(
       defaultparams)
  confvariant biometrics mapping
    VPName i_auth_type selectedVariants(fingerprint
       fingervein iris face; min:1, max:1)
    VPName i_encryption_parameters selectedVariants(
       setparams)
  defaultVariant userinfo
  type parameterization
  bindingTime devtime

configuration authentication_mode:
  varType externalVP
  alternative
    variant online
    variant offline
  (min:1,max:1)
  realization "it is realized by i_auth_mode and i
     _encryption_parameters variability points, setting
     params for sessionkey"
  confvariant online mapping
    VPName i_auth_mode selectedVariants(mode_online)
    VPName i_encryption_parameters selectedVariants(
       setparams)
  confvariant offline mapping
    VPName i_auth_mode selectedVariants(mode_offline)
  defaultVariant offline
  type parameterization
  bindingTime devtime
```

Table 3.18: Configuration Interface of adaptive security system-contd'

```
1  configuration fake_transaction_enabling:
2    varType externalVP
3    optional
4      variant fake_trans
5      variant real_trans
6    realization "it is realized by i_transaction_type
       variability point"
7    confvariant fake_trans mapping
8      VPName i_transaction_type selectedVariants(
         faketransaction)
9    confvariant real_trans mapping
10     VPName i_transaction_type selectedVariants(
         realtransaction)
11   defaultVariant fake_trans
12   type addition
13   bindingTime devtime
14
15 Constraints
16       i_auth_type face requires i_auth_mode selectedVariants
           (mode_online)
17       i_auth_mode mode_online const protocol="https"
18       i_auth_type esign const i_encryption_parameters
           defaultparams=valueOf{username_passw esign}
19       i_auth_type esign const i_encryption_parameters Mobile
           =valueOf{Mobile PC}
20
21 Parameter Settings
22   parameter noofbiometricauthtypeselected =
       #ofVariantsSelected{fingerprint fingervein iris face}
       Of i_auth_type
23   parameter defaultparams = value(username_passw,
       onetimepassw,esign)
24   parameter fakeinterface existswhenselected{i_transaction
       _type.faketransaction}
```

Any variant can activate required functions in service and choreography interfaces. "i_auth_m ode", internal variation point (line 21) is responsible for activation of different functions of storage and thirdparty services when its related variants are selected. For instance, "mode_onli ne" variant activates "getconnection, savehashed-

data, verify" functions of thirdparty service when selected (line 23).

In Table 3.18 *Constraints* part includes a logical constraint (line 16), stating that "face" variant of "i_auth_type" variation point requires "mode_onli ne" variant of "i_auth_mode" variation point to be selected. In lines 17-18 numerical constraints are presented in one of which "mode_online" variant of "i_auth_mode" variation point constraints the "protocol" property to be set to "https".

Moreover, any variability in choreography configuration interface that affects context elements in choreography can be defined in Parameter Settings part. Their values are set when the choreography is configured. For instance, "noofbiometricauthtypeselected" in Table 3.18 (line 22) identifies the number of times for extracting features from biometric data. Its value is assigned when variants of "i_auth_type" are selected.

Table 3.19: Configuration Interface of comparison orchestration

```
1 Configuration interface vconf_comparison of service
    comparison
2     @vconfservice
3     vp analysis:
4   optional
5     variant fake :activateMethods(fakeanalysis ,compare)
6     variant real:activateMethods(compare)
7   bindingTime devtime
```

An example for configuration interface of comparison orchestration is listed in Table 3.19 where analysis in comparison of the user credentials varies as fake and/or real. Fake analysis enables system to enact the alarm state whenever a user uses his/her alarm finger being under threat. Real analysis neglects this type of alarm states; basically compares coming data with existing user credentials. In order to represent this kind of variability, a variation point (line 2) is defined with fake (line 5) and real (line 6) variants. Fake variant activates fakeanalysis and compare methods residing in comparison service interface and real variant only activates the compare method. When a method is activated, this method is included in the service interface. In other words, activation is used for configuring service interfaces, deciding which methods

and related parameters should take place.

3.4.2.2 Choreography

Choreography model includes composition constructs with variability attachments, context elements and variability mappings between interacting services and choreographies. Basic fault handling mechanism is supported by choreography. For this purpose, different types of faults are specified and generated. Predetermined fault types are delivery, parameter, notready, waittimeout, insufficientmessage, notavaliable and termination with condition.

As presented in Table 3.20, Table 3.21, Table 3.22, and Table 3.23 adaptable security system choreography firstly imports its configuration interface. This is used to relate its external and internal variability with composition variability and to map it to utilized service variability. Then interacting choreographies and services are imported with or without their configuration interfaces. This provides an opportunity to utilize services with different configuration interfaces, that is with different service interfaces.

Variables defined within Context Elements take part with their default values in composition and are shared within the choreography. For instance, in Table 3.20 “noofbiometricauthtypeselected” (line 23) is a referred variable, and its value is set in the configuration interface. “wrongattempts” is newly specified in here to store the number of wrong attempts to limit verification trials.

Adaptable security system choreography maps its internal variation points and related variants to those of utilized services’ in order to configure service variability. The mapping between lines 32-34 ensures that when “i_encryption_parameters” variation point is bound to one of its variants, “encryption_params” variation point of encryption service is bound accordingly to provide a consistent interaction.

Adaptable security system choreography carries out “verify” (line 15) in Table 3.21 and "enroll" (line 28) in Table 3.22 functionalities comprising a set of interactions. Atomic and composite interactions are tagged with variability attachments when variability in that part of the composition is needed.

Table 3.20: Adaptable security system choreography

```
choreography adaptablesecuritysystem
  import configuration vconf_adaptablesecuritysystem
  use choreography chor_alert
  use choreography chor_credentialmng
  import service connection
  import service encryption with configuration vconf
     _encryption
  import service credentials
  import service attemptcalc
  import service comparison with configuration vconf
     _comparison
  import service responsewindow
  import service interfaceprep with configuration vm
     _interfaceprep
  import service thirdparty with configuration vm_thirdparty
  import service user
  import service warning

  //Shared variables
  Context Elements
    //user s wrong attempts
    wrongattempts 0
    //fake interface content enabling
    fakeinterface false
    //biometric selected authentication type variants
       specifies the number
    noofbiometricauthtypeselected 0
    //defualt parameters for encryption
    defaultparams "username_passw"
    //user entered credential data
    usernamepass ""
    //extracted features of user biometric data
    processeddata ""

  Choreography Variability Mapping
      VP i_encryption_parameters maps service encryption VP
         encryption_params
       Variant defaultparams maps Variant withdefaultparams
       Variant setparams maps Variant withparams
    VP i_transaction_type maps service comparison VP analysis
       Variant faketransaction maps Variant fake
       Variant realtransaction maps Variant real
```

Table 3.21: Adaptable security system choreography-cont'd

```
VP i_auth_type maps service thirdparty VP user_device
  Variant username_passw maps Variant ATM Mobile PDA PC
  Variant onetimepassw maps Variant ATM Mobile PDA PC
  Variant esign maps Variant ATM Mobile PDA PC
  Variant fingerprint maps Variant PC
  Variant fingervein maps Variant PC
  Variant iris maps Variant PC
  Variant face maps Variant PC
VP i_auth_type maps choreography credentialmng VP devicecon
  Variant fingerprint maps Variant biometricdevice
  Variant fingervein maps Variant biometricdevice
  Variant iris maps Variant biometricdevice
  Variant face maps Variant biometricdevice

Function verify:
sequence (
  #vp i_auth_type ifOneSelected( fingerprint fingervein iris face) # repeat noofbiometricauthtypeselected times(
    user send{chor_credentialmng} message getcredentials(deviceparameter)
    %comp processeddata =chor_credentialmng.getcredentials%
    chor_credentialmng send{encryption} message setparams(parameters)
  )

  #vp i_auth_mode ifSelected(mode_online)# sequence (
    thirdparty receive message getconnection()
    thirdparty send{encryption} message setparams(parameters)
  )
  user send{chor_credentialmng} message getcredentials(deviceparameter)
  %comp usernamepass =chor_credentialmng.getcredentials%
  chor_credentialmng send{encryption} message setparams(parameters)
  encryption receive message encrypt(credentials)

  #vp i_auth_mode ifSelected(mode_online)# sequence (
    encryption send{thirdparty} message verify(data)
    #vp i_transaction_type ifSelected(faketransaction)# thirdparty send{comparison} message fakeanalysis(comparisonresult)
```

Table 3.22: Adaptable security system choreography-cont'd

```
1            %comp fakeinterface=comparison.fakeanalysis%
2        )
3        #vp i_auth_mode ifSelected(mode_offline)# sequence (
4            encryption send{storage} message gethasheddata()
                 referedDestinations(comparison)
5            #vp i_transaction_type ifSelected(faketransaction)#
                 storage send{comparison} message fakeanalysis()
6        )
7
8        guard(fakeinterface==false) sequence (
9          comparison send{attemptcalc} message calculate_wrong
               _attempts(result)
10         %comp wrongattempts=attemptcalc.calculate_wrong
               _attempts%
11         guard(wrongattempts == 3) parallel (
12           comparison send{responsewindow} message show()
13           attemptcalc send{connection} message closeconnection
                 ()
14         )
15         guard(wrongattempts   3) parallel (
16           comparison send{responsewindow} message show()
17           attemptcalc send{warning} message warn(response
                 _warning)
18         )
19       )
20       guard(fakeinterface==true) #vp i_transaction_type
             ifSelected(faketransaction)#parallel (
21         sequence (
22           comparison send{interfaceprep} message
                 prepareinterface()
23           interfaceprep send{responsewindow} message show()
24         )
25         comparison send{chor_alert} message alert()
26       )
27     )
28   Function enroll:
29     sequence (
30       user send{chor_credentialmng} message getcredentials(
             deviceparameter)
31       %comp usernamepass =chor_credentialmng.getcredentials%
```

Table 3.23: Adaptable security system choreography-cont'd

```
1       chor_credentialmng send{encryption} message setparams(
            parameters)
2       #vp i_auth_type ifOneSelected( fingerprint fingervein
            iris face)# repeat noofbiometricauthtypeselected
            times(
3            user send {chor_credentialmng} message
                 getcredentials(deviceparameter)
4            %comp processeddata =chor_credentialmng.
                 getcredentials%
5            chor_credentialmng send{encryption} message
                 setparams(parameters)
6       )
7       encryption receive message encrypt(credentials)
8       #vp i_auth_mode ifSelected(mode_online)# encryption send
            {thirdparty} message savehasheddata(hasheddata)
9       #vp i_auth_mode ifSelected(mode_offline)# encryption
            send{storage} message sethasheddata(hasheddata)
10      interfaceprep send{responsewindow} message show()
11    )
```

The lines 17-21, 23-26 and 32-34 in Table 3.21, lines 3-6 in Table 3.22 and lines 2-6,8, 9 in Table 3.23 include attachments referring to specified variation declarations in the configuration interface of the adaptable security system choreography. For instance, "# vp i_auth_mode ifSelected(mode_online)" to indicate the point which composition can change (line 64).

3.4.2.3 Service and Choreography Interface

Service and choreography interface model comprises only interface specifications without variability. Each choreography and service has its own interface including all possible functionalities to be configured by configuration interfaces. In Table 3.24, encryption service interface is shown with its exposed functionality as "encrypt" (line 3), and "setparams" (line 9) with pre-post conditions, input and outputs. Other services and choreographies can collaborate with it using "encryption" port (line 14).

Table 3.24: Encryption Service Interface

```
Service interface encryption

  function encrypt
    precondition(sessioncreated == true)
    postcondition(data_encrypted == true)
    input(credentials)
    output hasheddata

  function setparams
    precondition(params_required == true)
    postcondition(set_params == true)
    input(parameters)

  portName encryption binding hostname:8082
```

In Table 3.25, adaptable security system choreography interface named as “chor_adap tablesecuritysystem” declares its functionalities “verify” and "enroll" with pre-post conditions, and input and output parameters. Different from service interfaces, it explicitly states required choreographies with a list of functions.

3.5 Tool Support for XChor

Xtext is used to implement XChor Language which provides a development environment for domain specific languages to developers with Eclipse IDE integration. XChor files created from three models are: (i) choreography interface, (ii) service interface, (iii) configuration interface for choreography, (iv) configuration interface for service, and (v) choreography specification. These files are categorized under configuration, services, and choreographies packages respectively in order to increase understandability.

Choreography, orchestration and atomic services are specified with variability specifications in Xtext. Binding variability and revealing a consistent collaboration require analysis of variability specifications. This analysis requires considering constraints,

choreography and service configurations with regard to variation selections. For this purpose, XChorS tool is developed

- to analyze variability relations which reveal configuration effects on orchestration and service interfaces,
- to configure choreographies, services regarding variant selections, and
- to output configured XChor files in a specified destination folder.

Table 3.25: Adaptable Security System Choreography Interface

```
1  Choreography interface chor_adaptablesecuritysystem of
       adaptablesecuritysystem
2
3    function verify
4      precondition(authentication_mode_selected == true)
5      postcondition  (verification_result_set == true)
6      input(user_info)
7      output response
8
9    function enroll
10     output enrollmentnotification
11
12   portName verifyuser binding hostname:8082
13
14   required interfaces
15     from chor_credentialmng function { getcredentials }
16     from chor_alert function { alert }
```

Consequently, service architecture can be reused in constituting related orchestrations and services, such a reuse may be beneficial for different stakeholders, needs, and choreographies. XChorS tool employs parsing, dependency analysis, and configuration phases.

Parsing Phase In this phase all XChor files are parsed in order to be analyzed and configured afterwards. Parsed files are configuration interfaces of service and

choreographies, service interfaces, choreography interfaces and choreography specifications.

Analysis Phase This phase starts after variation bindings specified by tool users. Some analysis is done before configuration of files with regard to variation bindings. All selected variation points are stored in a list.

- **Missing Variation Point Bindings:** All development time variability of services are bound according to direct user selections or variability bindings coming from choreography variability mappings. Therefore, whether all required variation bindings are specified by tool users or not is analyzed with regard to variability of services and variability mappings defined in the choreography specification. The tool first tries to find proper variants, such as selection of variants among alternative ones. However, if the tool can not find a proper binding, it warns and asks for missing bindings.
- **Additional Variation Point Bindings:** Configuration Variation Points (CVP) facilitate a high level understanding for configuration purposes while hiding details of how low level bindings are done. In other words, realization of each CVP is described by a set of variation points and variant selections. For this reason, after all CVP's and their realization information are picked, required variation bindings realizing the selected variant for each CVP are added to the proper variation binding list. If variant selection of a CVP is not bound, then default variant is assumed to be selected and related required variation point bindings are done. However, if there are additional variation point bindings which should be done by tool users, then tool asks for these selections.
- **Constraints:** All constraints VP to VP or VP to V defined within choreographies are analyzed and checked whether these are satisfied by the bound variation. If they are not satisfied, the tool warns the user.
- **Binding Time Consistency:** All selected variation point binding times are gathered and checked whether the variation points can be bound in development/design time. If they are not bound in design time, the tool warns user about the inconsistency of binding times. Moreover, in CVP

binding case, realization of a CVP needs other VP bindings. Therefore, binding times of additional variation points which realize the selected variant of CVP should be analyzed and checked also.

- **Existence of Redundant Variation Points:** The variation point specifications which do not take part in realization of a CVP or does not reside in composition variation are analyzed, revealed and then shown to tool users.

Configuration Phase This phase is applied after analysis, after being sure all required VPs and variants are selected. All choreography, orchestration and atomic services are configured with regard to user variation point selections.

- **Proper VP Bindings:** Bindings of VPs to its variants are revealed.
- **Parameter Settings:** Existence or values of parameters, which are utilized within choreography, depend on selected VP bindings. The decision whether the parameter takes part in the composition is made with regard to VP bindings or the value of each is assigned after analyzing the binding.
- **Configuration of Service and Choreography Interfaces:** Interfaces of service and choreography can be tailored by configuration interfaces. Functions and related parameters are added or removed from interfaces according to VP bindings. For this reason, the configurations resulted from VP bindings are revealed and the changes are applied to interfaces. Then a set of configured interfaces are the outputs.
- **Composition Variation Bindings:** Choreography composition is formed according to VP bindings, the parts which are guarded by VPs are added if the variants are selected. All resolved VP information is removed from composition.

The analysis phase shows which variation points are related with which services and service functions. It helps in the configuration phase to determine which services interact with each other and which functions reside in their interfaces.

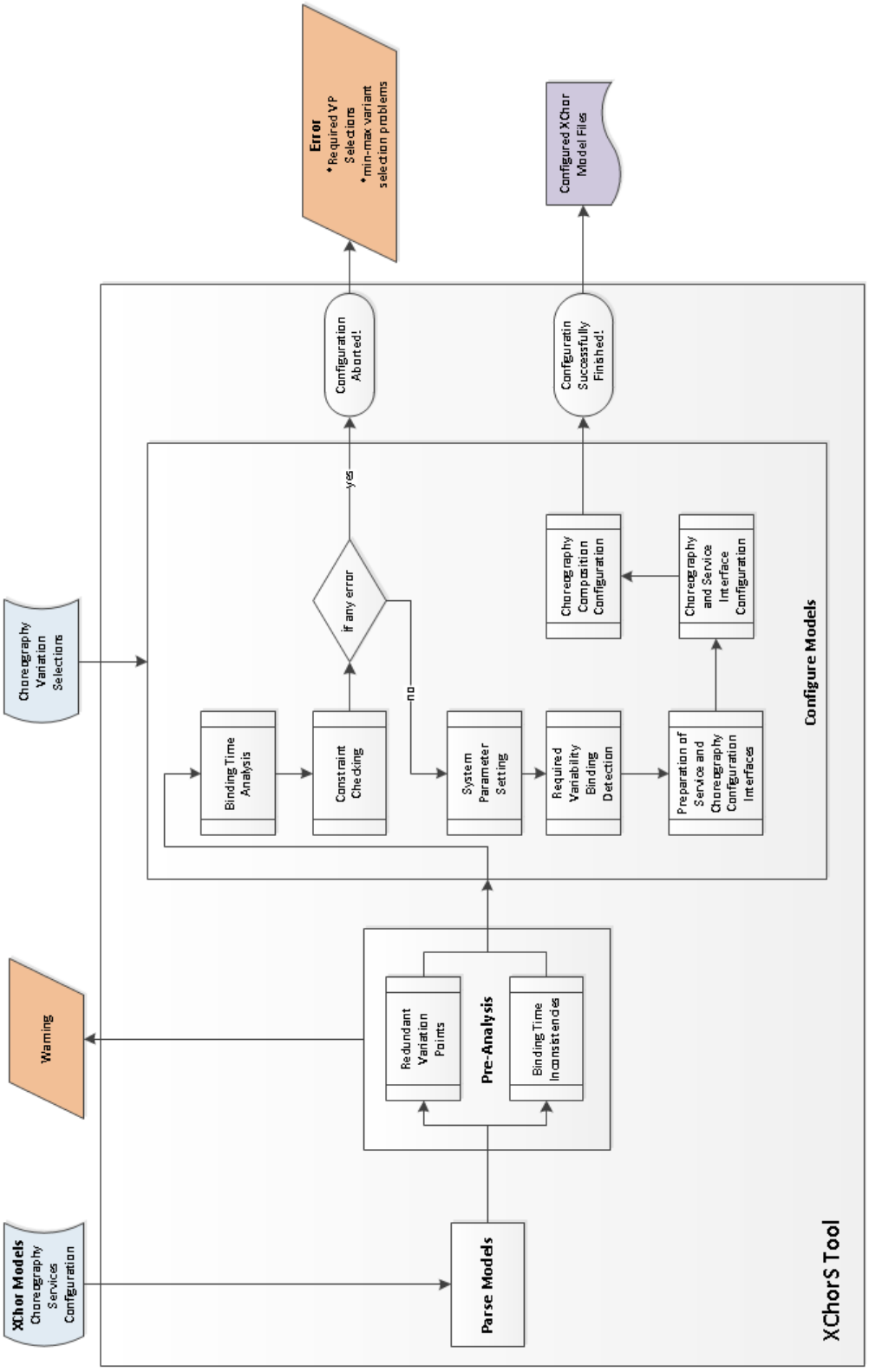

Figure 3.11: XChor Tool Execution Flow.

According to variation selections, the tool (i) configures interfaces by enabling and

disabling its functions and parameters, (ii) prepares choreography compositions and abstract process definitions of orchestration by examining whether the parts with variation attachments are included. Finally, the tool outputs configured choreography and related services and configuration interfaces if there are variation points that will be bound at runtime.

The flow diagram for representing execution of XChor Tool is presented in Figure 3.11. After parsing XChor models, warnings can be generated if there are binding inconsistencies between related variation points. These inconsistencies can arise from variation associations or variability mappings specified in choreography. Moreover, if any redundant variation point specified in configuration interfaces which does not take part in configuration variation point or is not referenced inside composition , a warning is shown to user including variation point information. After checking constraints and analyzing required variation point bindings, the tool can generate errors indicating additional variation point bindings or problems with minimum/maximum variant selections.

Adaptable security system choreography has been depicted in Table 3.20 before variation binding. Table 3.26 and Table 3.27 shows configured choreography after selection of (i) "onetimepassw","fin gervein" and "face" variants of "i_auth_type" variation point and (ii) "biometrics" variant of "authentication_type" variation point.

The tool includes three main blocks of functionality: Parse, Pre-analysis and Configuration. Algorithmic Complexities of Parse, Pre-analysis and Configuration are presented in Table 3.28, in Table 3.29 and in Table 3.30 respectively where "No of" stands for "Number of", "Fs" is "Files", "VP" represents "Variation Points", "VA" indicates "Variation Associations", "Consts" is "Constraints", "PS" stands for "Parameter Settings", "Ints" is "Interactions", "Invs" indicates "Invariants", "Funcs" represents "Functions", "UC" is "Used Choreographies", "CE" stands for "Context Elements", "VM" represents "Variation Mappings", "CI" is "Configuration Interfaces", "Servs" stands for "Services", "Chor" represents "Choreography", "Chors" indicates "Choreographies", and "CS" is "Choreography Specification".

Table 3.26: Configured adaptable security system choreography

```
choreography adaptablesecuritysystem
  use choreography chor_alert
  use choreography chor_credentialmng

  import service connection
  import service encryption
  import service credentials
  import service attemptcalc
  import service comparison
  import service responsewindow
  import service interfaceprep
  import service thirdparty
  import service user
  import service warning

  Context Elements
    wrongattempts 0
    noofbiometricauthtypeselected 2
    usernamepass ""
    processeddata ""

    Function verify:
    sequence (
      repeat noofbiometricauthtypeselected times(
        user send{chor_credentialmng} message getcredentials(
            deviceparameter)
        %comp processeddata =chor_credentialmng.
            getcredentials%
        chor_credentialmng send{encryption} message setparams(
            parameters)
      )

      sequence (
        thirdparty receive message getconnection()
        thirdparty send{encryption} message setparams(
            parameters)
      )

            user send{chor_credentialmng} message
                getcredentials(deviceparameter)
      %comp usernamepass =chor_credentialmng.getcredentials%
      chor_credentialmng send{encryption} message setparams(
          parameters)
      encryption receive message encrypt(credentials)
```

Table 3.27: Configured adaptable security system choreography-cont'd

```
            encryption send{thirdparty} message verify(data)

      sequence (
        comparison send{attemptcalc} message calculate_wrong
            _attempts(result)
        %comp wrongattempts=attemptcalc.calculate_wrong
            _attempts%
        guard(wrongattempts == 3) parallel (
          comparison send{responsewindow} message show()
          attemptcalc send{connection} message closeconnection
              ()
        )
        guard(wrongattempts   3) parallel (
          comparison send{responsewindow} message show()
          attemptcalc send{warning} message warn(response
              _warning)
        )
      )
    )

  Function enroll:
   sequence (
     user send{chor_credentialmng} message getcredentials(
         deviceparameter)
     %comp usernamepass =chor_credentialmng.getcredentials%
     chor_credentialmng send{encryption} message setparams(
         parameters)
     repeat noofbiometricauthtypeselected times(
          user send {chor_credentialmng} message
              getcredentials(deviceparameter)
          %comp processeddata =chor_credentialmng.
              getcredentials%
          chor_credentialmng send{encryption} message
              setparams(parameters)
     )
     encryption receive message encrypt(credentials)
     encryption send{thirdparty} message savehasheddata(
         hasheddata)
     interfaceprep send{responsewindow} message show()
   )
```

Table 3.28: Algorithmic Complexity of Parsing XChor Models

Pseudo-code	Complexity
FOR file IN XChor_file_directory:	1
read_file_content ()	n (No of Fs)
IF file is configuration_interface:	1
FOR variation_point IN variation_point_list:	m (No of VP
parse_variation_point()	1
IF variation_association in variation_point:	k (No of VA)
parse_variation_association()	1
FOR constraint IN constraints:	l (No of Consts)
parse_constraint()	1
FOR parameter_setting IN parameter_settings:	p (No of PS)
parse_parameter_setting()	1
IF abstract_process_definition EXISTS:	1
parse_abstract_process_definition()	r (No of Ints)
IF file IS service_interface OR choreography_interface:	1
FOR invariant IN invariants:	s (No of Invs)
parse_invariant()	1
FOR function IN functions:	t (No of Funcs)
parse_precondition()	1
parse_postcondition()	1
parse_input()	1
parse_output()	1
FOR port IN ports:	u (No of ports)
parse_port()	1
IF file IS choreography_specification:	1
FOR choreography IN used_choreographies:	v (No of UC)
parse_choreography()	1
FOR service IN imported_services:	y (No of IS)
parse_service()	1
FOR context_element IN context_elements:	z (No of CE)
parse_context_element()	1
FOR variability_mapping IN variability_mappings:	a (No of VM)
parse_variability_mapping()	1
FOR function IN functions:	b (No of Funcs)
FOR interaction IN function:	c (No of Ints)
parse_interaction()	1
FOR variability_attachment IN function:	d (No of VA)
parse_variability_attachment()	1

Table 3.29: Algorithmic Complexity of Pre-analysis of XChor Models

Pseudo-code	Complexity
FOR file IN parsed_files:	n (No of Fs)
IF file IS choreography:	1
Variability_mappings = get_variability_mappings ()	1
FOR variability_mapping IN variability_mappings:	a (No of VM)
analyze_binding_time_consistency()	1
get_configuration_interface_of_choreography()	1
FOR variation_point in variation_points:	m (No of VP)
is_taking_participation_in_composition()	1
is_taking_participation_in_configuration()	1

The complexity of pre-analysis functionality is calculated as follows:

```
O(PreAn) = n(1+1+a+1+2m)
                = O(nm + na)
```

Pre-analysis complexity directly relates with the product of number of XChor models, variation points and their associations and increases expeditiously.

The complexity of parse functionality is calculated as follows:

```
O(Parse)= (n+1)+(1+m(k+2)+l+p+r)+(1+s+4t+u)
                              +(1+v+y+z+b(c(d+1)))
            = O(n+mk+p+r+s+t+u+v+y+z+bcd)
            = O(n + mk + bc)
```

Parsing complexity increases with number of XChor models (n), the number of variation points (m) along with their associations (v), number of choreography functions and interactions within functions.

The complexity of configuration functionality is calculated as follows:

```
O(Conf) = e(m+3+a+3+l+3+f+a+g+h+i)
            = O(e(m+a+l+f+a+g+h+i))
           = O(e(m+f+g+i))
```

Table 3.30: Algorithmic Complexity of Configuration of XChor Models

Pseudo-code	Complexity
FOR choreography IN files:	e (No of Chors)
error = gather_all_required_vp_bindings()	m (No of VP)
IF error EXISTS:	1
show_error()	1
stop_configuration()	1
error = binding_time_analysis (user_variation_selections)	a (No of VM)
IF error EXISTS:	1
show_error()	1
stop_configuration()	1
error = check_constraints((user_variation_selections))	l (No of Consts)
IF error EXISTS:	1
show_error()	1
stop_configuration()	1
configure_choreography_configuration_interface(user_variation_selections)	f (No of VP in CI)
variability_mappings = get_variability_mapping()	a (No of VM)
prepare_service_interfaces(variability_mappings)	g (No of Servs in Chor)
prepare_service_interfaces(variability_mappings)	h (No of CI of Servs)
prepare_choreography_composition(user_variation_selections)	i (No of Ints in CS)

Configuration complexity is dependent on the product of number of choreographies and the sum of variation points, interacting services and interactions within choreography specification and increases mercurially.

3.6 Application Development with XChor

To develop such variable service compositions by means of one or more choreographies and interacting services, top-down, bottom-up or both of these strategies can be applied.

Top-down development First choreography capabilities as choreography interface and its variability are determined in its configuration interface. Then, during choreography specification collaborating services and their functionality and possible service variability are specified. With these decisions, the service interface with its possible functionalities without variability is created. If a service provides variability, then its configuration interface is prepared. Afterwards, the newly created service is included by means of importing its interface with or without its configuration interface. If choreography variability bindings affect newly created service variability bindings, a variability mapping is added into the choreography specification. Required context variables are identified which are used for changing parts of choreography composition. The addition and change in choreography specification and services are done until reaching intended service-oriented application. Then the system is ready to be analyzed and configured by the user via XChorS tool.

Bottom-up development Contrary to top-down development, first possible service interfaces are created with their functionality. If required, their variations are specified in their own configuration interfaces. One or more choreography interfaces with possible functionalities are created. Then, with the help of defined services, choreography specifications are filled with service interactions along with required variability mappings. Shared information stored in context variables is defined for serving service interaction flow. Choreography functionalities that can be altered as service interactions are defined. By this way, as new

choreographies are defined, they are ready to be used by other choreographies specified beforehand. This addition and alteration process continues until the intended functionalities of choreography specification are achieved. Then, the system is ready to be analyzed and configured by user via the XChorS tool.

The two approaches can be used simultaneously as a hybrid approach. With regard to variability association specification, each service and choreography can change its interface with specified variants with the constructs *activatemethods* and *setParameters* defined in their configuration interfaces. Only variants can alter interface functions and parameters. However, only a choreography can change a service's interface which interacts with other services in the context of this choreography. Services are not allowed to change other service's interfaces.

In service and choreography interfaces, pre and post conditions can be defined but not analyzed by XChorS at this moment. In fact, the metamodel forms a structure which you can extend to be analyzed for these type of semantic information.

3.7 XChor Language Evaluation under Comparison Framework

XChor Language is evaluated according to the components of comparison framework explained in Chapter 2, Section 2.4. The metamodel of XChor comprises variability modeling, choreography modeling for service composition and mapping between these models. Some properties of the XChor language are evaluated in Table 3.31.

Variability modeling part of XChor is evaluated with regard to the Variability Modeling Component. Variability model of XChor enables to specify external and internal variation points with mandatory, optional and alternative variants. Configuration Variation Points are the structures where high level variation points can be mapped to low level variation points. Configuration variation points are not mapped to choreography composition, they are only used for abstracting details of variation point bindings in order to increase understandability and decrease complexity. Logical and numerical constraints are defined among variation points and variants.

Choreography modeling part of XChor is evaluated with regard to Composition and Configuration of Models Component. Choreography model of XChor facilitates the definition of service compositions as choreography specifications with variability support in a composition. With use of XChor variability model, choreography, orchestration and atomic services can define their own interface variability in their configuration interfaces where existing function and related parameters can be altered. By this way, choreography can associate its own variability bindings to that of interacting service's either specifying a one-to-one mapping or altering service interface functions or parameters by enabling and disabling.

XChorS Tool is evaluated with regard to Tool Support Component. An eclipse plugin Xtext enables to define XChor models depending on the XChor metamodel. Pre-analysis, parsing and configuration of these models with regard to user variability selections are supported by the XChorS Tool. After variability bindings, BPEL4Chor and VxBPEL specifications can be generated from the XChor models. Moreover, verification of variable XChor models is achieved by transformation to fPromela language.

3.8 Validation of XChor

The metamodel of XChor brings together two parts: variability management and choreography as service composition, each of which should be validated separately. The capability of XChor to cover variability is validated and examined with characteristics, types and needs of variability in service-oriented applications. The capability of XChor to realize recurring service interactions are validated with patterns defined in the context of service-orientation.

3.8.1 Modeling Service Variability through XChor Language

There have been approaches for modelling and managing variability within service-oriented [133, 80, 100, 106] and service-oriented software product line[25, 47, 78,

Table 3.31: XChor Evaluation under Components of the Comparison Framework

Approach	**Types of VP and V**	**Constraints**	**Ext. and Int. Rep. of VP**	**Realization**	**Design Artifact Relation**	
XChor	Optional, Mandatory, Alternative	Logical and Numerical Constraints	Ext. and Int. VP	Conf. VP	Mapping to Choreography, Atomic Services	Orchestration and

Language	**Composition Approach**	**Modeling Approach**	**Variability Support**	**Variability In**			**Variability Association**
				Interface	**Connector**	**Composition**	
XChor	Choreography	Interaction	Yes	Yes	None	Yes	Yes

Approach	**Specification**	**Analysis**	**Verification**	**Code Generation**	**Configuration**	**Tool**
XChor	Yes	Yes	Yes	Yes	Yes	XChorS and Xtext

114] contexts. Among them, [100] is a comprehensive study covering types, characteristics and needs of service variability in detail and presenting a review of works and challenges in variability management in service-oriented systems.

The study categorizes services into two groups; atomic and composite services. Our notion of service also covers atomic services and composite services realized as orchestration and choreography. According to the study, there are four kinds of variability that should be addressed to model variability completely: exposed variability, composition variability, partner variability and partner exposed variability.

XChor metamodel facilitates definition, organization, relation and binding of variability in service interfaces and compositions in choreography level. In our context, service covers choreography, orchestration and atomic service concepts. The realization of each type in XChor language is explained in detail in the following subsections.

3.8.1.1 Exposed variability

The variability revealed in a composite service's interface is the exposed variability. In our context, services have two types of interfaces: (1) Service interface without variability which includes all possible functionalities and (2) configuration interfaces of the service which covers variability aspects. Therefore, exposed variability of composite services (orchestration and choreography) is included in their configuration interfaces in XChor language.

To represent variability explicitly in configuration interfaces in XChor, internal, external, configuration variation points and constraints among them are specified. Internal variation point is invisible to outer context, namely service users so as to describe variability with a set of variants and specified binding time. External variation point is explicit to users of the service in order to be referenced, utilized and configured with a set of variants and specified binding time. Internal and external variation point syntax is the same except their type indication, that is either internal or external. The syntax and examples for variation point specifications are given in Table 3.2. Logical or numerical constraints among variation points can be represented in configuration interfaces whose syntaxes and examples are given in Table 3.5 and Table 3.6.

Configuration interface as a whole facilitates the representation of exposed variability for choreography, orchestration and atomic services, whose examples are given in Table 3.19 and Table 3.16.

3.8.1.2 Composition variability

Composition variability refers to variability specified in behavior, the way services are interacted with each other. In XChor scope, variability in composition is specified in choreography specification where a set of choreography functionality is defined. Variable parts of composition are tagged with variability attachment which enacts the parts surrounded with it when referred variants of a variation point are selected. In other words, the parts of composition can participate in the composition when specified conditions are met in variability attachments, otherwise the parts cannot

Table 3.32: A part of adaptable security system choreography

```
1  choreography adaptablesecuritysystem
2      ...
3
4      Function verify:
5      sequence (
6        #vp i_auth_type ifOneSelected( fingerprint fingervein
             iris face) # repeat noofbiometricauthtypeselected
             times(
7          user send{chor_credentialmng} message getcredentials(
               deviceparameter)
8          %comp processeddata =chor_credentialmng.
               getcredentials%
9          chor_credentialmng send{encryption} message setparams(
               parameters)
10       )
11
12       #vp i_auth_mode ifSelected(mode_online)# sequence (
13         thirdparty receive message getconnection()
14         thirdparty send{encryption} message setparams(
               parameters)
15       )
16   ...
```

participate. Variability attachment syntax and an example are given in Table 3.14 and Table 3.15.

A part of adaptable security system choreography including two composition variabilities in line 6-10 and line 12-15 are shown in Table 3.32. In the first variability, whenever one of the variants among fingerprint, fingervein, iris or face is selected, the "repeat" composite interaction is included in verify functions (line 6). As such, if online authentication is selected, then the "sequence" composite interaction should reside in the composition (line 12).

3.8.1.3 Partner variability

Partner variability includes its bound variability of the service within a composition context. In the XChor scope, interacting services take part in composition with their configuration interfaces. Therefore, services can join interactions with different configuration interfaces, meaning with different variability. Variability of interacting services playing role in a composition can be achieved in two ways:

1. Establishing association between choreography variability and interacting service variability. This enables to describe which variation points and related variants of choreography are related with those of the interacting service. It facilitates the establishment of a direct association between the variant of the variation point when selected, and the variant of the determined variation point. In this way, the service can be configured with regard to choreography variation binding. The way how to establish variability association is presented with its syntax in Table 3.12 and Table 3.13.

2. Establishing variability binding effect on a service interface while introducing a new variant within a configuration interface. In configuration interfaces, a variant can enable or disable functions of a service interface and can change the parameters of functions accordingly. By this way, variants can configure service interfaces with required consistency. This configuration occurs in a choreography configuration interface to configure interacting service interfaces.

Table 3.33: Newly Specified Variability Binding Effect on Configuration Syntax and Example

```
1 variant <varname>:activateMethods(service:<servicename>,funct
    :<function_1>,<function_2> ,...)
2                                   :setParameter(toFunct:
                                        <function>,parameter:
                                        <params>)
```

XChor Language - Newly Specified Variability Binding Effect on Configuration Syntax

```
1 internalVP iauthmode:
2   optional
3     variant mode_online:activateMethods(service:thirdparty ,
          funct:getconnection ,savehasheddata ,verify)
4     variant mode_offline:activateMethods(service:storage ,
          funct:gethasheddata)
5   bindingTime devtime
```

XChor Language - Newly Specified Variability Binding Effect on Configuration Example

In Table 3.33, authentication mode internal variation point is defined within the configuration interface of an adaptable security system choreography. When authentication is done online, the "thirdparty" service needs to support getconnection, savehasheddata and verify functions (line 3). If user is authenticated offline, storage service should have the function gethasheddata in its interface (line 4). With regard to authentication mode bindings "thirdparty" and "storage" service interfaces are configured to provide consistency.

3.8.1.4 Partner exposed variability

Partner exposed variability defines the offered variability of a service to other services which is strongly related with partner variability. In our context, all configuration interfaces of a service interface provides all possible functional variations of the service.

3.8.2 Modeling Choreography through XChor Language

There have been categorizations with regard to interactions between services [33, 24, 96, 68]. Among them, although informally described, Service Interaction Patterns[33] facilitates the assessment of choreography languages as a benchmark. The patterns represent a collection of patterns describing bilateral, multilateral, competing, atomic and causally related interactions. The patterns, revealing service functionality and behavior, are pertaining to service composition, namely orchestration and choreography. Service interactions define the collaboration among a number of interacting services with regard to predefined rules. Interaction patterns are categorized under four groups: Single-transmission bilateral, Single-transmission multilateral, multi-transmission and routing. Each group comprises a set of patterns. These recurring patterns are used to represent the fulfillment of XChor language constructs for service interactions. Atomic and composite interactions of XChor are utilized for depicting realization of the patterns described.

Realization of service interaction patterns are explained in detail as in the same way in [33]. The subtitles for each pattern (Pattern name, description, example, issues/design choices) are used as is. Solution subtitle is replaced by realization in XChor language which describes the use of XChor to realize the corresponding pattern. Two new subtitles: "graphical representation" and "explanation" are added. Graphical representation is to depict the pattern with box and line drawing without addressing all pattern details. Explanation is to describe the graphical representation.

3.8.2.1 Single-transmission bilateral interaction patterns

The first categorization is the single-transmission bilateral interaction patterns comprising send, receive, and send/receive which correspond to elementary interactions. Detailed explanations of each are given with graphical representations.

Pattern 1: Send

Description A service sends a message to another service.

Example

A user credentials gathering service sends user credentials as parameters to be encrypted to an encryption service.

Issues/design choices The send interaction pattern can be blocking or non-blocking and can result in faults.

- In blocking send pattern, the source service waits for acknowledgment from destination service and choreography does not progress until the acknowledgment is received. In the non-blocking send pattern, the source is only responsible for preparing and sending the message. The distinction between blocking and non-blocking interaction mode is determined at the level of the executable choreography language.
- Faults can be originated from delivery problems or from destination side. Delivery fault occurs when the message is not delivered to the destination service. Faults from the destination side are occurred when an error arises in destination and should be directed to the source service. For example, a purchase order sent to a supplier may result in a fault message from the supplier because the customer ID does not match the customer name.

Graphical Representation

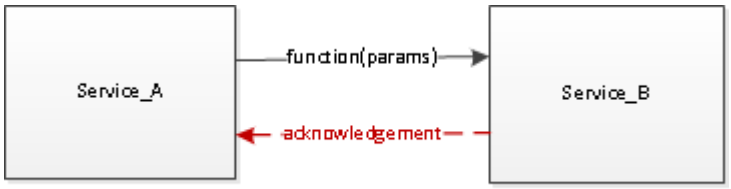

Figure 3.12: Send Pattern.

Explanation

In Figure 3.8.2.1, service_A sends a message comprising the function with parameters to Service_B. Acknowledgement is sent to Service_A only in the blocking case. Faults can be resulted from delivery problems or from destination side; however they are not depicted in this graphical representation.

Realization in XChor Language

1. **Basic send**

   ```
   <service_A> send{<service_B>}
   message <function>(<params>)
   ```

   where

   <*service_A*>: message sending service, source service

   <*service_B*>: message receiving service, destination service

   <*function*>: the name of function residing in service_B interface

   <*params*>: a list of parameters separated by comma

   Example

   ```
   credentials send encryption  message encrypt(
      credentialData)
   ```

2. **Non-blocking send** Same as basic send.

3. **Blocking send**

   ```
   <service_A> send{<service_B>}
               message <function>(<params>)
               withNotification
   ```

   where

   withNotification keyword enables service_A to waits the acknowledgement as notification.

   Example

   ```
   credentials send encryption  message encrypt(
      credentialData) withNotification
   ```

4. **Non-blocking send with fault**

   ```
   <service_A> send{<service_B>}
               message <function>(<params>)
               fault (<names>)
   ```

where

fault keyword enables to define a fault. <*names*> is a list of fault names.

Example

```
credentials send encryption  message encrypt(
   credentialData) fault (delivery)
```

5. **Blocking send with fault**

```
<service_A> send{<service_B>}
            message <function>(<params>)
            withNotification
            fault (<names>)
```

where

fault keyword enables to define a fault. <*names*> is a list of fault names.

Example

```
credentials send encryption  message encrypt(
   credentialData) withNotification fault(delivery)
```

Pattern 2: Receive

Description A service receives a message from another service.

Example

An image retrieval service receives a message indicating that features of biometric data are extracted.

Issues/design choices The send interaction pattern can be blocking or non-blocking and can result in faults.

- The destination service may wait for an acknowledgment from a called service.
- The destination service can be ready/not ready to consume the message. A fault is generated if the destination service is not ready to

process the message and no queuing system is applied. Otherwise, the message is appended to queue waiting to be processed after a predefined time.

Graphical Representation

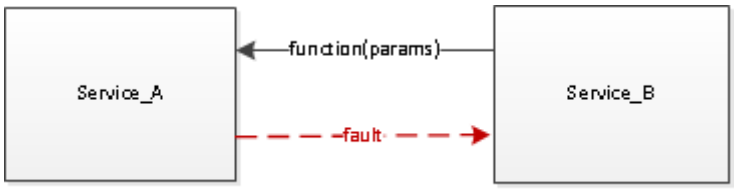

Figure 3.13: Receive Pattern.

Explanation

In Figure 3.8.2.1, Service_A receives a message comprising the function with parameters. A fault can result from delivery problems or readiness of the processing. Indication of the source service, Service_B is optional for realizing blocking patterns.

Realization in XChor Language

1. **Basic receive**

```
<service_A> receive from {<service_B>}
            message <function>(<params>)
```

where
<*service_A*>: message receiving service, destination service
<*service_B*>: message sending service, source service
<*function*>: the name of function residing in service_A interface
<*params*>: a list of parameters separated by comma

Example

```
1 imageretrieval receive message extract features (
    biometric data)
2 comparison receive from storage  message compare (data
    )
```

2. **Receive with fault**

```
<service_A> receive
            message <function>(<params>)
            fault (<names>)
```

where

fault keyword enables to define a fault. <*names*> is a list of fault names.

Example

```
1 imageretrieval receive message extract features (
    biometric data) fault (notready)
```

Pattern 3: Send/Receive

Description A service can send a message to another service then receives the response message or receives a message from another service and sends a response message.

Example

A storage service sends credentials to be encrypted to the encryption service and then receives encrypted data from it.

Issues/design choices The send interaction pattern can be blocking or non-blocking and can result in faults.

- Source service can wait for a response or for a fault indicator to enable blocking send pattern.
- Outgoing and incoming messages are correlated in order to provide consistency in data.
- Send, receive or both interactions can result in fault messages.

Graphical Representation

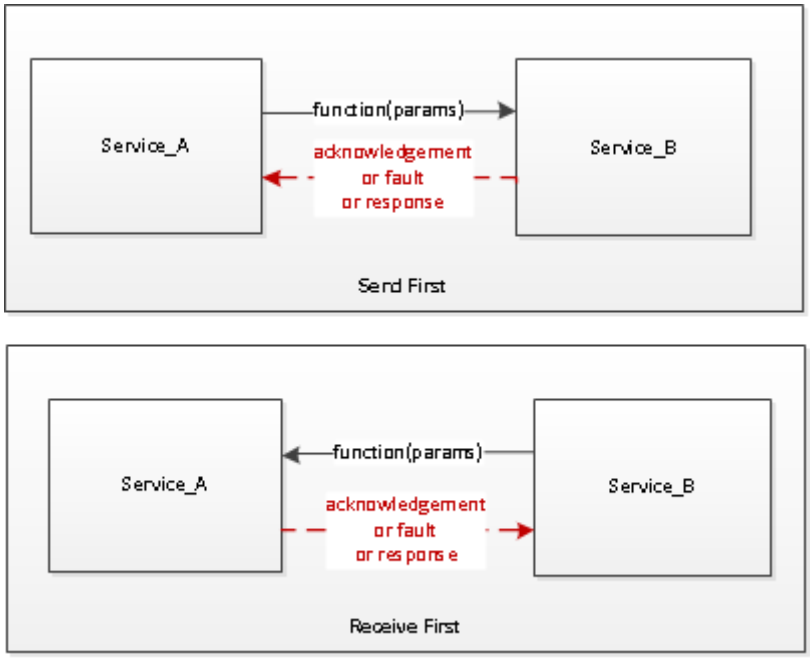

Figure 3.14: Send/Receive Pattern.

Explanation

In Figure 3.8.2.1, two cases are covered:

Send First Service_A sends a message comprising the function with parameters to Service_B. A fault, acknowledgement message or response can be generated by Service_B and received by Service_A.

Receive First Service_A receives a message comprising the function with parameters from Service_B. A fault, acknowledgement message or response can be generated by Service_A and received by Service_B.

Realization in XChor Language

Fault specification, blocking and non-blocking realization are the same in send and receive patterns. In this case, after fault is generated the sending and receiving services (Service_A and Service_B) are aborted.

1. **Basic Send First**

```
sequence (
<service_A> send{<service_B>} message
<function_B>(<params_B>)
<service_A> receive from {<service_B>} message
<function_A>(<params_A>)
```

)

where

<*service_A*>: message sending and receiving service

<*service_B*>: message sending and receiving service

<*function_A*>: the name of function residing in service_A interface

<*function_B*>: the name of function residing in service_B interface

<*params_A*>: a list of parameters pertaining to function_A separated by comma <*params_B*>: a list of parameters pertaining to function_B separated by comma

Example

```
sequence (
    storage send encryption message encrypt(data)
    storage receive from encryption message
        save encrypted data(encrypted data)
)
```

2. **Basic Receive First**

```
sequence (
<service_A> receive from {<service_B>} message
<function_A>(<params_A>)
<service_A> send{<service_B>} message
<function_B>(<params_B>)
)
```

where

<*service_A*>: message sending and receiving service

<*service_B*>: message sending and receiving service

<*function_A*>: the name of function residing in service_A interface

<*function_B*>: the name of function residing in service_B interface

<*params_A*>: a list of parameters pertaining to function_A seperated by comma <*params_B*>: a list of parameters pertaining to function_B seperated by comma

Example

```
sequence (
  thirdparty receive from encryption  message
      getconnection()
  thirdparty send encryption  message setparams(
      parameters)
)
```

3.8.2.2 Single-transmission multilateral interaction patterns

The second categorization is the single-transmission multi-lateral interaction patterns comprising racing incoming messages, one-to-many send, one-from-many receive, one-to-many send/receive where multiple services come into play. Detailed explanations of each are given with graphical representations.

Pattern 4: Racing incoming messages

Description A service expects to receive one among a set of messages. These messages may be structurally different, coming from different source services. The processing of messages is changed with regard to the source service.

Example

- Multiple messages in the same structure: A customer waits for a connection to receive a movie among a set of service providers. If one connection is established, the others are ignored.
- Multiple messages in different structures, exclusive: Hotel booking request is processed with regard to coming message indicating that it rejects the booking or accepts the booking. If one of the messages is received, the other one is neglected. One excludes the other.
- Multiple messages in different structures, not-exclusive: User authentication data is processed with respect to the device they use either be a keyboard, a fingerprint device or finger-vein device.

Issues/design choices

- Incoming messages are of same or different types.
- The processing that follows the message consumption may be different depending on the consumed message.
- When one of the expected messages is received, the corresponding continuation is triggered. The remaining messages may or may not need to be discarded.
- Depending on the underlying communication infrastructure, several of the expected messages may be simultaneously available for consumption. In this case, two approaches may be adopted: (i) let the system make a non-deterministic choice, or (ii) provide a "ranking" among the competing messages. In any case, only one message is chosen for consumption.

Graphical Representation

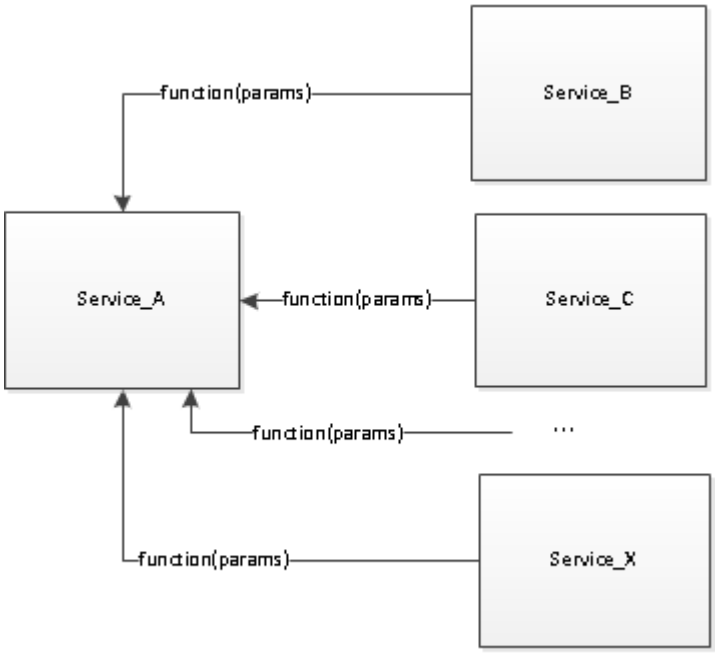

Figure 3.15: Racing Incoming Messages Pattern.

Explanation

In Figure 3.8.2.2, Service_A receives a set of messages from Service_B,

Service_C.., Service_X and selects one of them to continue to process.

Realization in XChor Language

1. **Multiple messages in same structure**

```
<service_A> receive from
    {<service_B> <service_C> ... <service_X>}
    pickOne message <function>(<params>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*> <*service_C*> ... <*service_X*>: a set of message sending services, source services

<*function*>: the name of function residing in service_A interface

<*params*>: a list of parameters separated by comma

Example

```
customer receive from provider1 provider2 provider3
    pickOne message establish connection(customer
    IP)
```

2. **Multiple messages in different structure**

```
select (
            <Proccesing_wrt_B>
            <Proccesing_wrt_C>
            ...
            <Proccesing_wrt_X>
)
```

where

<*Proccesing_wrt_B*>: Start with a receive interaction which refers to single transmission bilateral receive pattern. Then followed by a set of interactions including all types of patterns surrounded with possible sequence, parallel, repeat or select structures.

The *select* keyword chooses non-deterministically one of the branches

with regard to coming message which can be either from service_B, service_C ... or service_X.

Example

```
select (
  sequence (
    itineraryplanner receive from hotel message
        rejectbooking(customerID)
    itineraryplanner send customer message
        showresult(bookingrejected)
  )
  sequence (
    itineraryplanner receive from hotel message
        acceptbooking(customerID)
    itineraryplanner send customer message
        makepayment(bookingID)
    ...
  )
)
```

Pattern 5: One-to-many send

Description A service sends messages to a set of services having the same type of message.

Example

A travel itinerary sends a price quote to all available hotels to gather different offers. A travel itinerary sends price quote notification indicating that one of them is selected for reservation after traveler chooses one of them. The message content of the selected one is different than others.

Issues/design choices

- The number of destination services is known at design time. All instances of the same service get the message even if the function names on the service interfaces are different.

- For reliable delivery concern, each destination service can generate and send fault notification to source service. Fault handling mechanism depends on selection of the way how the application responds to one or more faults. The application can terminate even if one fault occurs or can tolerate all faults, and uses logging.

Graphical Representation

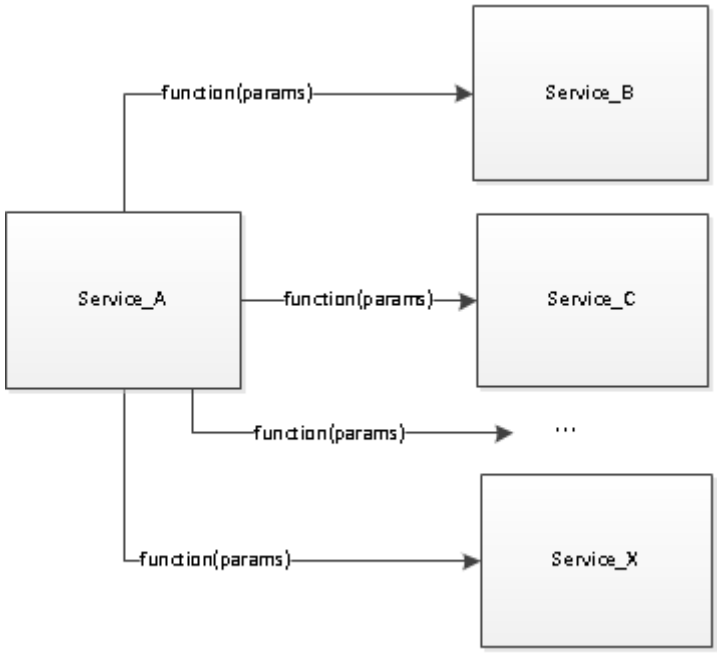

Figure 3.16: One to Many Send Pattern.

Explanation

In Figure 3.8.2.2, Service_A sends messages to Service_B, Service_C.., Service_X.

Realization in XChor Language

1. **Send without fault**

```
<service_A> send
    {<service_B> <service_C> ... <service_X>}
    message <function>(<params>) refers
    <service_B>.<function_B>
    <service_C>.<function_C>
```

```
...
<service_X>.<function_X>
```

where

<*service_A*>: message sending service, source service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message receiving services, destination services

<*function*>: the name of function residing in service_B, service_C,... service_X interface

<*params*>: a list of parameters separated by comma

refers keyword can be optionally used if the <function> names are not same as service_B, service_C and service_X interface. Most common function name is written in place of <function>, for each different function names following piece of code is added:

<*service_B*>.<*function_B*> is an example of the reference to <*function_B*> in <*service_B*> interface.

Example

```
1   travel itinerary send  sherton rixos
        jwmarriot fourseasons  message
        price quote(startDate, endDate,
        additionalRequests)
2   travel itinerary send  sherton rixos
        jwmarriot fourseasons  message
        price quote(startDate, endDate,
        additionalRequests) refers sherton.
        get price rixos.get quote
```

2. **Send with fault**

```
<service_A> send
    {<service_B> <service_C> ... <service_X>}
    message <function>(<params>)
    refers
    <service_B>.<function_B>
    <service_C>.<function_C>
```

```
...
<service_X>.<function_X>
fault (<names>, terminateIf <number> fails)
```

where

<*service_A*>: message sending service, source service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message receiving services, destination services

<*function*>: the name of function resided in service_B, service_C,... service_X interface

<*params*>: a list of parameters separated by comma

refers keyword can be optionally used if the <function> names are not same as service_B, service_C and service_X interface. Most common function name is written in place of <function>, for each different function names following piece of code is added:

<*service_B*>.<*function_B*> is an example of the reference to <*function_B*> in <*service_B*> interface. *fault* keyword enables to define fault. <*name*> is a list of fault names. *terminateIf* <*number*> *fails* represents termination condition where <*number*> is the amount of faults generated.

Example

```
travel itinerary send   sherton rixos jwmarriot
    fourseasons  message price quote(startDate, endDate
    , additionalRequests) fault (delivery, terminateIf
    1 fails)
```

Pattern 6: One-to-many receive

Description A service receives a number of logically related messages from a set of services which then are gathered as a single message. Because of this, each message should arrive in time. The success of the interaction depends on gathered messages and time constraint if exists.

Example

A travel itinerary receives a set of prices which belongs to previously requested price offers for fulfilling a customer request. Travel itinerary collects all received messages to present all available hotel prices to the customer. It can wait for a specified time frame for messages to arrive. It can wait only for a specified number of price offers to come, for instance if three price offers are enough out of five.

Issues/design choices

- A mechanism is required for correlating messages coming from different sources via the message content or not.
- Message correlation should be constrained with a specified time not to let destination service wait forever.
- A number of messages can be sufficient to proceed without waiting for other messages to come.
- A timeout can occur with insufficient number of messages. In this case, a fault can be generated by the destination service indicating that receive interaction ended unsuccessfully.

Graphical Representation

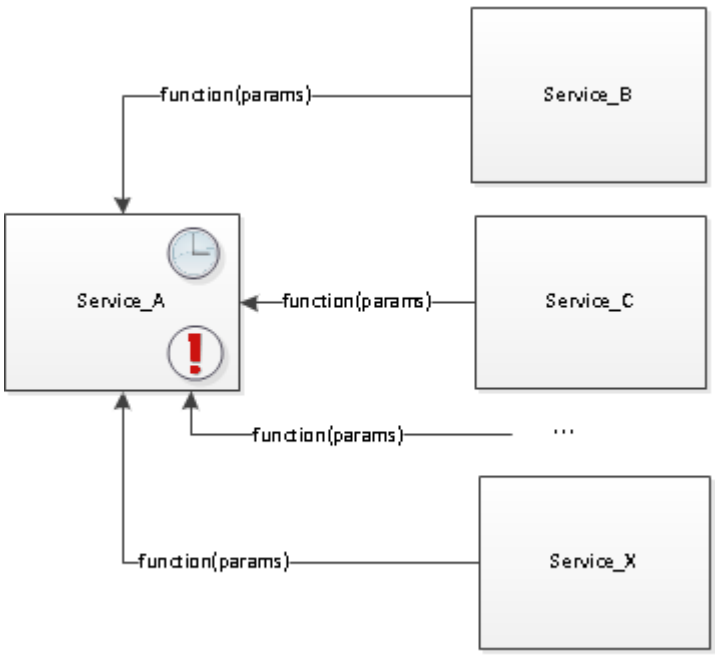

Figure 3.17: One to Many Receive Pattern.

Explanation

In Figure 3.8.2.2, Service_A receives messages from Service_B, Service_C,..., Service_X with a time and success condition.

Realization in XChor Language

```
<service_A> receive from
    {<service_B> <service_C> ... <service_X>}
    message
    <function>(<params>)
    wait <duration> <time>
    until <number>
    messagescame fault (<names>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message sending services, source services

<*function*>: the name of function residing in service_A, service_C,... service_X interface

<*params*>: a list of parameters separated by comma

wait keyword enables to define time and success condition specification. <*duration*> is an integer to indicate the waiting period. <*time*> is a string indicating the type of duration as second, minute, hour, day and month. *until* keyword enables to define the success condition. <*number*> is an integer with which *messagescame* keyword indicates the condition the destination service completes the receiving interaction.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 travel itinerary receive from  sheraton rixos jwmarriot
    fourseasons  message price quote(customer id) wait
    10 seconds
2 travel itinerary receive from  sheraton rixos jwmarriot
    fourseasons  message price quote(customer id) wait
```

```
    until 2 messagescame
3 travel itinerary receive from sheraton rixos jwmarriot
    fourseasons message price quote(customer id)
    wait 10 seconds until 2 messagescame
4 travel itinerary receive from sheraton rixos jwmarriot
    fourseasons message price quote(customer id) wait
    10 seconds fault(waittimeout)
5 travel itinerary receive from sheraton rixos jwmarriot
    fourseasons message price quote(customer id) wait
    10 seconds until 2 messagescame fault(waittimeout)
```

Pattern 7: One-to-many send/ receive

Description A service sends messages to a set of services having the same type of message and may be related. Then the service receives a number of logically related messages from a set of services which then are gathered as a single message. Because of this, each message should arrive in time. The success of the interaction depends on gathered messages and time constraint if exists.

Example

A travel itinerary sends a price quote to all available hotels to gather different offers. The travel itinerary sends price quote notification indicating that one of them is selected for reservation after traveler chooses one of them. The message content of the selected one is different from others. Then the travel itinerary receives a set of price offers for fulfilling the customer request. The travel itinerary collects all received messages to present all available hotel prices to the customer. It can wait for a specified time frame for messages to come. It can wait only a specified number of price offers to come, for instance if three price offers are enough out of five.

Issues/design choices

- The number of destination services is known at design time. All

instances of the same service get the message even if the function names on the service interfaces are different.

- For reliable delivery concern, each destination service can generate and send fault indications to the source service. Fault handling mechanism depends on the selection of the way how the application responds to one or more faults. The application can terminate even if one fault has occurred or can tolerate all faults, and only use logging.
- A mechanism is required for correlating messages coming from different sources via the message content.
- Message correlation should be constrained with a specified time not to let destination service wait forever.
- A number of messages can be sufficient to proceed without waiting for other messages to come.
- A timeout can occur with in sufficient number of messages. In this case, a fault can be generated by the destination service indicating that receive interaction has ended unsuccessfully.

Graphical Representation

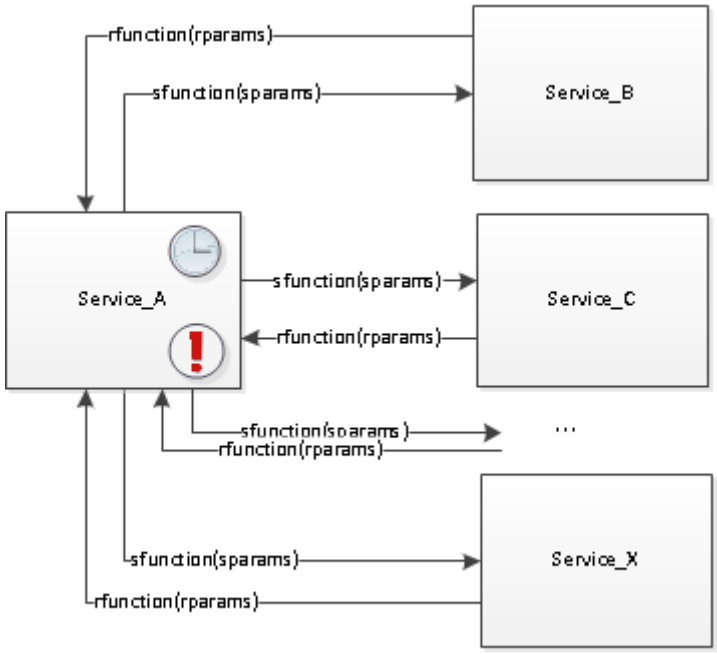

Figure 3.18: One to Many Send/Receive Pattern.

Explanation

In Figure 3.8.2.2, Service_A sends messages to and receives from Service_B, Service_C.., Service_X with a time and success condition.

Realization in XChor Language

```
sequence (
            <service_A> send
            {<service_B> <service_C> ... <service_X>}
            message <function>(<params>) refers
            <service_B>.<function_B>
            <service_C>.<function_C>
            ...
            <service_X>.<function_X>
            fault (<names>)

            <service_A> receive from
            {<service_B> <service_C> ... <service_X>}
            message <function_A>(<params_A>)
            wait <duration> <time>
            until <number>
            messagescame fault (<names>)
)
```

where

<*service_A*>: message sending and receiving service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message receiving and sending services

<*function*>: the name of function residing in service_B, service_C,...service_X interface

<*function_A*>: the name of function residing in service_A interface

<*params*>: a list of parameters of <*function*>, separated by comma

<*params_A*>: a list of parameters of <*function_A*>, separated by comma

wait keyword enables to define time and success condition specification. <*duration*> is an integer to depict waiting period. <*time*> is a string depicting the type of duration as second, minute, hour, day and month. *until* keyword enables to define success condition. <*number*> is an integer with which *messagescame* keyword depicts the condition the destination service completes the receiving interaction.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 sequence (
2     travel itinerary send   sheraton rixos jwmarriot
        fourseasons  message price quote(startDate, endDate
        , additionalRequests) fault (delivery)
3   travel itinerary receive from   sheraton rixos jwmarriot
      fourseasons   message price quote(customer id) wait
     10 seconds until 2 messagescame fault(waittimeout)
4 )
```

3.8.2.3 Multi-transmission interaction patterns

The third categorization is the multi-transmission interaction patterns comprising multi-responses, contingent requests and atomic multicast notification where multiple services come into play. Detailed explanations of each are given with graphical representations.

Pattern 8 : Multi-responses

Description A service sends a request to another service, then a set of responses are received from the service until no further response is required. The completion or the sign of no further response can come from the requestor service or the responder service, or specified by a duration or in a message content.

Example

A customer of Hurriyet online news service subscribed for latest news about the protests in Taksim, Istanbul. Hurriyet provides last-minute events when available. Customer can think received news are enough and decide to stop getting latest news or Hurriyet can stop posting more news updates after sending detailed events.

Issues/design choices

- The sign of no further messages can be in four cases:
  1. the requestor service sends a notification to stop,
  2. a relative or absolute deadline specified by the requestor service,
  3. an interval of inactivity during which the requestor service does not receive any response from the responder service,
  4. a message from the responder service indicating no messages are sent anymore.
- After observing the sign of no further messages, the requestor service does not accept any messages from the responder service.
- Requester service accepts multiple messages from the responder service whose number is determined in runtime
- Each received message is treated individually, that is the fault resulted from one received message does not affect the others.
- In the case of the requester service sending a notification, a mechanism should inform responder service about rejected messages sent between the time the requester sends notification and the time responder service receives it.

Graphical Representation

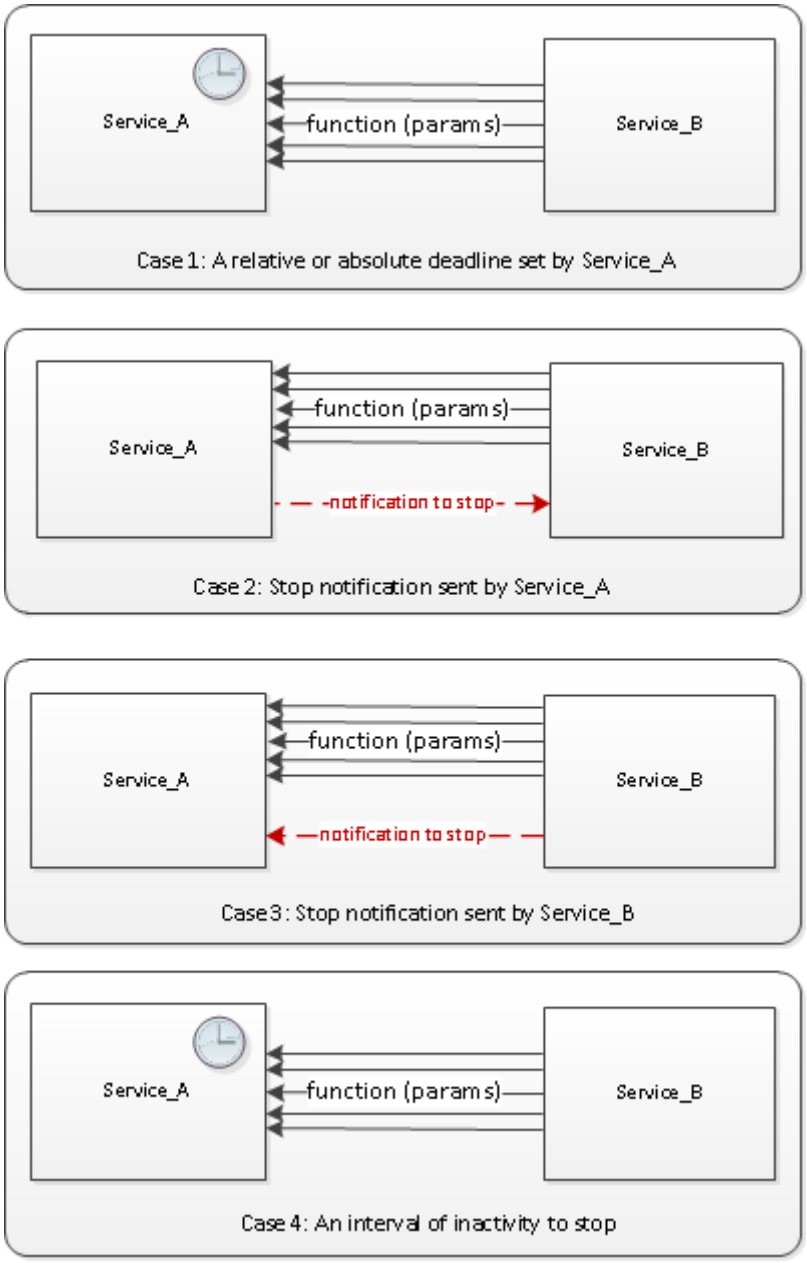

Figure 3.19: Multi Responses Pattern.

Explanation

Four cases are presented in Figure 3.8.2.3: *Case 1*: Service_A receives multiple messages from service_B during a specified time. Service_A specifies a deadline to stop reeving messages from Service_B.

Case 2: Service_A receives multiple messages from Service_B until Service_A sends a notification to stop.

Case 3: Service_A receives multiple messages from Service_B until Ser-

vice_B sends a notification indicating that it stops sending further messages.

Case 4: Service_A receives multiple messages from Service_B until a specified interval of inactivity occurs.

Realization in XChor Language

1. **Case 1 – A relative or absolute deadline set by Service_A**

```
<service_A> receive from {<service_B>}
            multiple times
            message <function>(<params>)
            wait <duration> <time>
            fault (<names>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*>: message sending service, source service

<*function*>: the name of function residing in service_A interface

<*params*>: a list of parameters separated by comma

multiple times keyword enables to specify that *service_A* receives more than one message from *service_B*.

wait keyword enables to define time and success condition specification. <*duration*> is an integer to depict waiting period. <*time*> is a string describing the type of duration as second, minute, hour, day and month.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 customer receive from   hurriyet   multiple times
    message latestnews() wait 60 seconds
```

2. **Case 2 – Stop notification sent by Service_A**

```
<service_A> receive from {<service_B>}
            multiple times message
```

```
<function>(<params>)
stopmessage from  <service_A>
fault (<names>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*>: message sending service, source service

<*function*>: the name of function residing in service_A interface

<*params*>: a list of parameters separated by comma

multiple times keyword enables to specify that *service_A* receives more than one message from *service_B*.

stopmessage from keyword enables to specify the service which sends the notification to stop the receive interaction. In this case service_A is the sender.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 customer receive from   hurriyet   multiple times
    message latest news() stopmessage from customer
```

3. **Case 3 – Stop notification sent by Service_B**

```
<service_A> receive from {<service_B>}
            multiple times
            message <function>(<params>)
            stopmessage from <service_B>
            fault (<names>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*>: message sending service, source service

<*function*>: the name of function residing in service_A interface

<*params*>: a list of parameters separated by comma

multiple times keyword enables to specify that *service_A* receives

more than one message from *service_B*.

stopmessage from keyword enables to specify the service which sends the notification to stop the receive interaction. In this case service_A is the sender.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
customer receive from   hurriyet   multiple times
   message latestnews() stopmessage from hurriyet
```

4. **Case 4 – An interval of inactivity to stop**

```
<service_A> receive from {<service_B>}
            multiple times
            message <function>(<params>)
            inactivity-interval <duration> <time>
            fault (<names>)
```

where

<*service_A*>: message receiving service, destination service

<*service_B*>: message sending service, source service

<*function*>: the name of function residing in service_A interface

<*params*>: a list of parameters separated by comma

multiple times keyword enables to specify that *service_A* receives more than one message from *service_B*.

inactivity-interval keyword enables to define time interval that the receive activity stops if the specified duration is exceeded. <*duration*> is an integer to indicate the waiting period. <*time*> is a string depicting the type of duration as second, minute, hour, day and month.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
customer receive from   hurriyet   multiple times
   message latestnews() inactivity interval 10
   seconds
```

Pattern 9 : Contingent request

Description A service, service_A sends a request to another service, service_B. If service_A can not get any response from service_B within a specified time frame, then service_A sends a request to service_C and waits for the response as it did in the service_B case. Service_A continues to sending requests to a number of services until getting a response from the current called service.

Example

A user of emergency service requests for ambulance due to a car crush. The the emergency service sends ambulance request from the nearest hospital. If the nearest one does not provide any response within a specified time, let's say 5 seconds, then the request is sent to the second nearest hospital. Request is sent until a response comes from within the set of hospitals.

Issues/design choices

- There can be issues such as response can come from previous request after the specified time frame; that is response comes late. In this situation, only the response coming from the current called service can be accepted and the other ones are discarded.

Graphical Representation

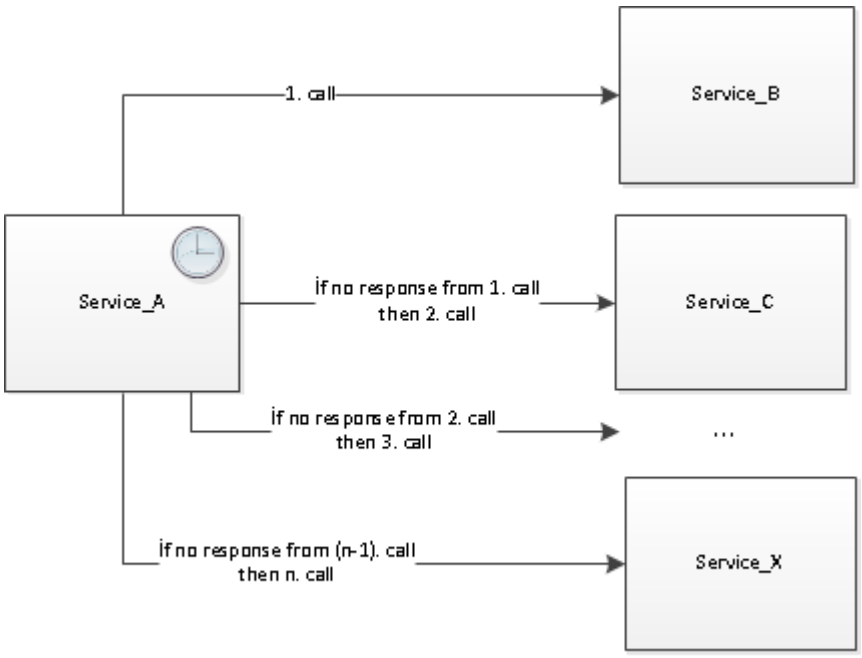

Figure 3.20: Contingent Request Pattern.

Explanation

In Figure 3.8.2.3, Service_A sends messages to Service_B, Service_C.., Service_X sequentially if Service_A cannot get any response from the current called service during a time frame.

Realization in XChor Language

```
<service_A> send
            {<service_B> <service_C> ... <service_X>}
            in-sequence
            message <function>(<params>)
            wait <duration> <time>
            fault (<names>)
```

where

<*service_A*>: message sending service, source service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message receiving services, destination services

<*function*>: the name of the function residing in service_B, service_C,...service_X

interface

<*params*>: a list of parameters of <*function*>, separated by comma

in-sequence keyword enables to specify that the list of services are called in sequence.

wait keyword enables to define time and success condition specification. <*duration*> is an integer to depict waiting period. <*time*> is a string indicating the type of duration as second, minute, hour, day and month. *until* keyword enables to define success condition. <*number*> is an integer used with *messagescame* keyword. The *messagescame* keyword defines the condition the destination service completes the receiving interaction.

fault keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 emergency service send   ataturk hospital medicana
    hospital bayindir hospital 100.yil hospital  in
    sequence message getambulance(coordinates,helpmessage)
     wait 5 seconds
```

Pattern 10 : Atomic multicast notification

Description A service, service_A sends notifications to a set of services which are expected to accept the notification within a specified time frame. The number of notification acceptances can have a minimum and maximum.

Example

A travel agent allows the booking of both flight and hotel travel requirements as part of a comprehensive travel packaging. Customers nominate their preferred flight carriers and hotel accommodation. Hotel and related flight details can be seen as an atomic group. Within this group, the flight carrier, and the booking agencies for hotels, identify the services contacted. All such atomic groups need to succeed in order for the interaction to succeed as a whole.

Issues/design choices

- The number of notified services can be known at design time or at run-time.
- The required minimum and maximum number of acceptance should be specified.
- The need for transactional support which enables notifying all related services as a group, selected or not. Identification of the group to be based on message content, such as customer id, request id or group id.
- The minimum number of services to accept the notification can range from one to the total number of services targeted, while the maximum number can range from the minimum number specified to the total number of services targeted.

Graphical Representation

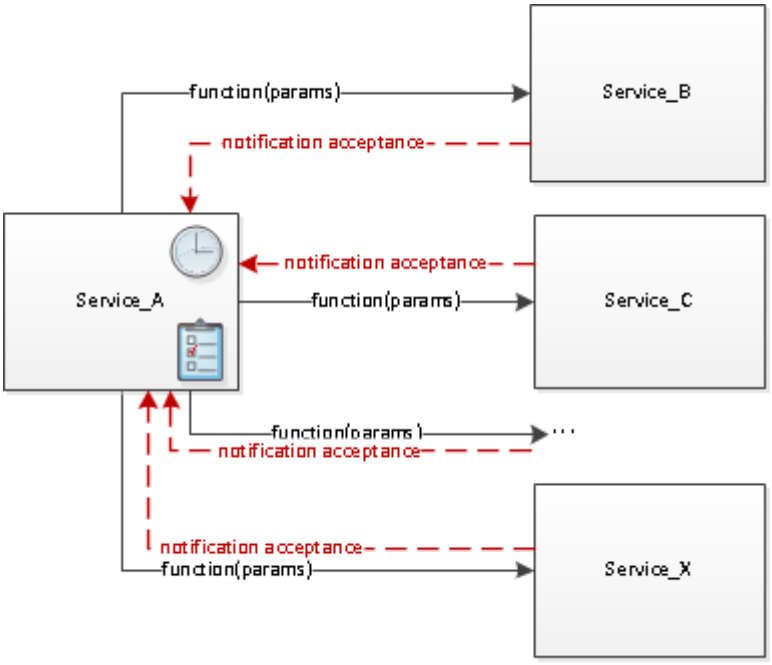

Figure 3.21: Atomic Multicast Notification Pattern.

Explanation

In Figure 3.8.2.3, Service_A sends a notification message to Service_B, Service_C.., Service_X and waits for acceptance of this notification within a time frame.

Realization in XChor Language

```
<service_A> send
    {<service_B> <service_C> ... <service_X>}
    atomic
    message <function>(<params>)
    wait <duration> <time>
    withNotification(min:<number> ,max:<number>)
    fault (<names>)
```

where

<*service_A*>: message sending service, source service

<*service_B*> <*service_C*> ... <*service_X*>: >: a set of message receiving services, destination services

<*function*>: the name of function residing in service_B, service_C,... service_X interface

<*params*>: a list of parameters of <*function*>, separated by comma

atomic keyword enables to specify transactional behavior.

wait keyword enables to define time and success condition specification. <*duration*> is an integer to define waiting period. <*time*> is a string depicting the type of duration as second, minute, hour, day and month. *withNotification* keyword enables service_A to wait the acceptance of notification sent by service_A. *min* and *max* keywords enable to define the least and the most amounts of acceptance notification from the destination services.<*number*> is an integer to indicate the minimum and maximum amounts. *fault* keyword enables to define fault. <*name*> is a list of fault names.

Example

```
1 travelitinerary send   sheraton rixos jwmarriot
    fourseasons pegasus anadolujet thy  atomic message book
    (startDate, endDate, additionalRequests, customerid)
    wait 60 seconds withNotification(min:2,max:7)
```

3.8.2.4 Routing patterns

The fourth categorization is the routing patterns comprising request with referral, relayed request and dynamic routing where multiple referred destination services come into play. Detailed explanations of each are given with graphical representations.

Pattern 11 : Request with referral

Description A service, service_A sends a message to another service, service_B which then sends the response to a set of services (service_1, service_2..., service_n). Faults can be sent to service_A or the set of services by service_B.

Example

A customer requests to make a reservation with the hotel between specified dates and with some conditions. Travel itinerary can send reservation request to the hotel and make the hotel respond to the customer directly. In other words, travel itinerary fades from the scene.

Issues/design choices

- The faults are sent to the source service, service_A by default if not specified otherwise.
- service_B may or may not have prior knowledge of the identity of the other services which are transferred by service_A.

Graphical Representation

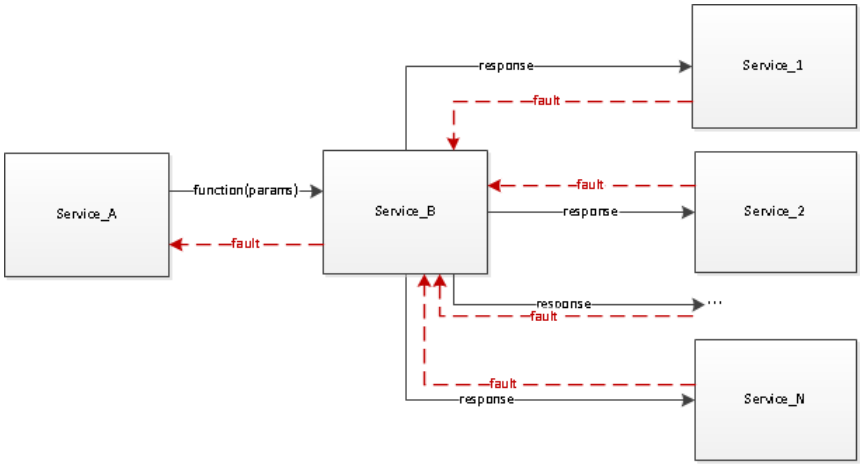

Figure 3.22: Request with Referral Pattern.

Explanation

In Figure 3.8.2.4, Service_A sends a message to Service_B which then sends related response to referred destination services(Service_1, Service_2.., Service_N). Faults can be sent to Service_B by destination services or to Service_A by Service_B.

Realization in XChor Language

```
<service_A> send {<service_B>}
            message <function>(<params>)
            referredDestinations(
               <service_1>,
               <service_2>,
               ...
               <service_N>
            )
            fault (<names>)
            toreferrals
```

where

<*service_A*>: message sending service, source service

<*service_B*>: message receiving service, destination service

<*function*>: the name of function residing in service_B interface

<*params*>: a list of parameters of <*function*>, separated by comma

referredDestinations keyword enables to specify a set of services that the response from service_B is sent. <*service_1*>,<*service_2*>,... <*service_N*> is a set of referred services.

fault keyword enables to define fault. <*name*> is a list of fault names.*torefer rals* keyword depicts that the fault is sent to referred services, *service_1, service_2....* This keyword is optional used to change destination of fault.

Example

```
1 customer send travelitinerary message reserve(
    startDate, endDate, additionalRequests)
2 travelitinerary send sheraton message reserve(
    startDate, endDate, additionalRequests)
    referredDestinations(customer) fault(notready)
    toreferrals
```

```
1 customer send travelitinerary message reserve(
    startDate, endDate, additionalRequests)
2 travelitinerary send sheraton message reserve(
    startDate, endDate, additionalRequests)
    referredDestinations(customer) fault(notavaliable)
```

Pattern 12 : Relayed request

Description A service, service_A sends a message to another service, service_B which then sends this request message to a set of services (service_1, service_2,..., service_n). After that, the set of service interact with service_A and service_B stays as a viewer of interactions and faults.

Example

A client seeking his/her debt status which can be queried from e-government services. Social Security Institution (SGK), tax department and credit dormitories institution are outsourced service providers of e-government services. The government authority stipulates that interactions between the client and debt status service providers be sent to it.

Issues/design choices

- The referred services may or may not have prior knowledge of service_A. It is the service_B's duty to inform about the source service, service_A.
- service_B can receive interested messages from referred destination services while playing viewer role.

Graphical Representation

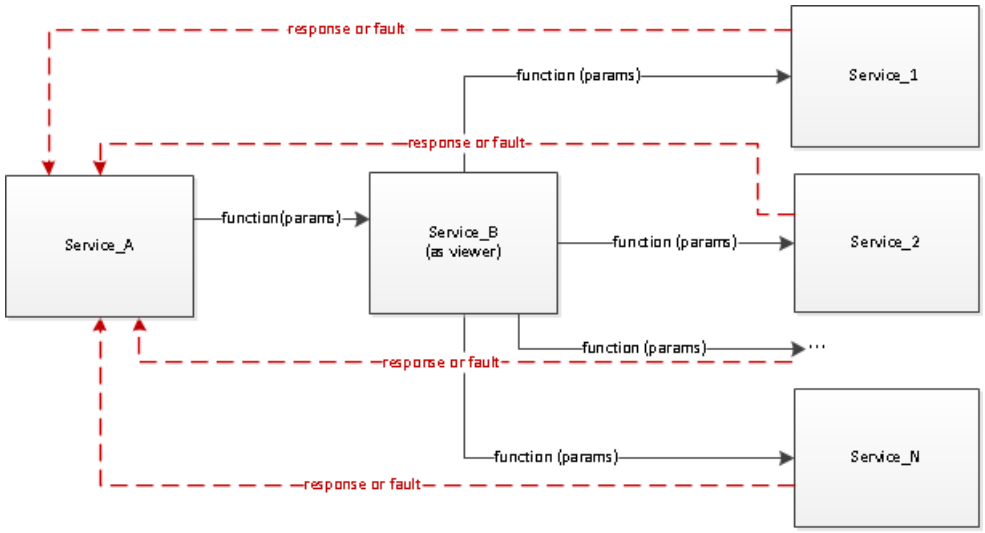

Figure 3.23: Relayed Request Pattern.

Explanation

. In Figure 3.8.2.4, Service_A sends a message to service_B which then sends the request to referred destination services(service_1, service_2.., service_N). After that, all responses or faults are directed to service_A.

Realization in XChor Language

```
<service_A> send {<service_B>}
            viewer
            message <function>(<params>)
            referredDestinations(
               <service_1>,
```

```
    <service_2>,
    ...
    <service_N>
)
fault (<names>)
```

where

<*service_A*>: message sending service, source service

<*service_B*>: message receiving service, destination service

<*function*>: the name of function residing in service_B interface

<*params*>: a list of parameters of <*function*>, separated by comma

viewer keyword enables service_B play the viewer role and responses are directed to service_A instead of service_B.

referredDestinations keyword enables to specify a set of services that the response from service_B is sent. <*service_1*>,<*service_2*>,... <*service_N*> is a set of referred services.

fault keyword enables to define fault. <*name*> is a list of fault names.*torefer rals* keyword depicts that the fault is sent to referred services, *service_1, service_2*.... This keyword is optionally used to change the destination of fault.

Example

```
customer send e government viewer message deptstatus(
    customerid) referredDestinations(sgk,taxdepartment,
    creditdormitoriesinstitution) fault(notavaliable)
```

```
parallel(
    customer send e government viewer message
        deptstatus(customerid) referredDestinations(sgk,
        taxdepartment,creditdormitoriesinstitution)
  e government receive from sgk,taxdepartment,
      creditdormitoriesinstitution message report(
      customerid)
)
```

CHAPTER 4

VERIFICATION OF XCHOR MODELS

This chapter is dedicated to representing a step by step approach in order to formally verify variable XChor models through model checking using Feature Transition Systems (FTS). Transformations from XChor to FTS model are introduced and required feature model of the variable choreography(ies) and related fPromela specification(s) are produced. Applying verification with the help of SNIP model is demonstrated and exemplified through case studies described in Chapter 3.

4.1 Need to Verify

XChor specification, which describes the behavior of service-oriented systems, supports variability in order to provide flexibility and to increase reusability of services. In variable-intensive service systems such as XChor choreography, verification is a costly and complex task due to having to check consistency of all possible choreographies and related services (orchestrations and atomic services). In XChor variable-intensive systems, variation points and related variants are scattered over variable choreography, orchestrations and atomic services. For small systems where variability is limited, consistency checking of variable choreography and related interacting services can be handled. Due to the complexity that comes with variability, deadlocks and incompleteness in composition behavior cannot be seen directly from variable choreography specification, even if variability is limited. For large scale systems this process is very complicated and error prune due to having to check all possible choreography against required variable behavior. To achieve consistent service ar-

chitecture and to reduce complexity of verification during adaptation to changes and maintenance, a formal model is required for quality assurance purposes.

4.2 Verification Approaches for Variable Systems

One of the approaches addressing formal verification of systems is model checking. System models and system properties are two main artifacts in model checking. System models describe the system behavior, while system properties define the specifications that are supposed to be satisfied by the system. Among these approaches, Modal Transition Systems (MTS)[64], PL-CSS (Product Line - Concurrent Communicating Systems, adaptation of the CCS process algebra) [71], and FTS[50] are the ones tackling formal verification of variable-intensive systems. In variable systems, it is important to relate product behavior to its properties in order to analyze where and how a property is violated. Focusing on software product lines, FTS enables such relations by linking possible product features with transitions which describes inner processes of possible products. To this end, FTS approach utilizes a feature model of a product line to reveal product features and constraints over them in TVL (Text based Variability Language) syntax[51]. It makes use of the product line behavior written in fPromela language along with the feature model [50]. fPromela language is an extension of Promela language which enables to guard statements with features and includes assertion statements. Assertion statements in fPromela specification are used for checking properties of the system through comparing the property with a value. SNIP model checker [50] is used as the verification tool which takes the feature model and fPromela specification of the product line. The tool verifies the fPromela specifcation against temporal properties and checks possible deadlocks. That the flow graph of choreography composition is not connected leads to a deadlock in fPromela. That is the situation when a variation binding causes a non-complete graph which is represented by fPromela specification. In that case, SNIP results with a deadlock along with violated product features and additional information about where the deadlock occurs. Likewise in assertion violation where the system property is not satisfied is provided with violated product features.

FTS approach is selected and applied for verification purposes of XChor variable

choreography in that

- Variability of a system is specified explicitly by means of a feature model.
- Explicit variability of the system, features are associated with behavior of the product line.
- Information about possible deadlocks and assertion violations are presented with the violated features which eases the error finding and correction.
- Even XChor variability model and feature model of the product lines are not mapped one-to-one, feature model construction is achievable from configuration interfaces of service and choreographies.
- Composition behavior specification in fPromela Language and XChor choreography specification are close to each other which makes transformation easier.
- Parallel composition is achievable which enables to run several choreographies simultaneously.

Along with its advantages to apply model checking to choreographies, the coverage of model checking is only for behavior variability. In XChor language, service and choreography interfaces can be changed by configuration interfaces and variability associations defined in choreography specifications. These interface changes lead to existence or non-existence of functions and parameters which are referenced within choreography specifications. In other words, due to service interface changes, one of the service function cannot be provided by a service even though it is stated that the service takes part in an interaction. Therefore, the service interaction with this functionality cannot occur within the choreography. Such interface consistency checking cannot be directly addressed by FTS. Therefore, an additional mechanism is needed to achieve interface variability checking which is left as future work.

4.3 Model Checking of Variable XChor Choreographies

The application of FTS model to XChor variable choreographies requires model transformation, because variability and behavior modeling are handled differently

Table 4.1: Variability and Behavior Models in XChor and FTS

	Variability Model	**Behaviour Model**
XChor	XChor Variability Model	XChor Choreography Model
FTS	Feature model in TVL	fPromela

which is shown in Table 4.1. The source model XChor, utilizes its own variability model in order to specify variation points and their associations between choreography and services. On the other hand, the target model, FTS utilizes TVL for representing product line's feature model as the input for verification of the product line behavior defined in fPromela. Therefore, a transformation needs to be conducted between these models while preserving their semantics.

According to Table 4.1, transformation is twofold; one for variability model explained in Section 4.3.1 and one for behavior model indicated with transformation rules in Section 4.3.2. After then, the usage of created FTS models for model checking of choreographies is explained in Section 4.3.3.

4.3.1 From Variability Model in XChor to TVL Feature Model

The target model, TVL employs a hierarchical structure including features, their attributes and constraints among them. The source model, XChor includes variation points (internal, external and configuration), related variants, constraints and variation point associations. Due to the semantic difference between TVL and XChor variability model, a mapping should be defined. Variability model in XChor requires all specified variation points to be added to the root of newly created feature model as mandatory. Moreover, TVL has no support for directly mapping configuration variation point logic to features. Variation points taking part in configurations should not be added to the root of the feature model, as the semantic relations among a set of variation point is hidden. The rest can be added to the root. A step by step description of constructing a TVL model from variability in XChor model is as follows:

Step 1 Variability Information Extraction Extract variation points stated as "externalVP", "internalVP", "configuration" and "vp" from configuration interfaces

of choreography and services. Extract related variants labeled as "mandatory", "optional" and "alternative".

Step 2 Root Construction Construct a root and name it with the application name.

Step 3 Feature Additions Assume that all variant and variation point names are distinct.

For Choreography For all choreographies

3.1 Add a mandatory feature to the root with choreography name, let's say chor1.

3.2 If and only if a VP does not participate in any configuration variation point, add it to chor1 as a feature and its related variants as sub-features. Otherwise, skip it.

3.3 If a VP is a configuration variation point, then add it to the chor1 as a mandatory feature and its variants as "optional", "mandatory" or "alternative" sub-features. For optional add someOf relation, for alternative add oneOf relation and for mandatory add allOf relation. For each of its variant, add a new feature as optional with new sub-features stated under "mapping" part as mandatory.

3.4 Repeat 3.2 and 3.3 steps until no variation point exists.

3.5 By use of variability association information gathered from Step 2, add each association as constraint.

Add all logical constraints of choreography as feature constraints.

Step 4 fPromela Feature List Construction Construct a list of leaf features residing in fPromela which are annotations for variability in behavior.

4.3.2 From XChor Behavior Model to fPromela

The process logic in XChor specification is transformed to the fPromela equivalent. The syntax of fPromela and Promela are the same. fPromela has almost all functionalities of Promela. The main difference between fPromela and Promela lies in the feature convention. fPromela has a new type called features that can be used to guard

statements with feature specific expressions. In [113] transforming VxBPEL variability logic to fPromela is explicitly defined for variable orchestrations. However, this approach does not cover choreography but sheds light to the way to transform.

Among variation point types, configuration variation point provides a high level understanding for configuration purposes while hiding details of how low level, internal binding logic is done. Therefore, these types of variation points do not take part in process specification, thus they do not need to be transformed to fPromela. All variation points except configuration should be transformed to fPromela in order to prevent information loss.

Each function realized by choreographies is transformed to fPromela equivalent separately. Then these functions are gathered in a single fPromela file as a pml file named with the application. The rules for transforming XChor behaviour specification and variability to equivalent fPromela constructs are listed in Table 5.2, Table 5.4 and Table 5.5 where P1, P2,.., Pn represent XChor Composite or Atomic Interaction specifications, fPromela-equvalent-P1, fPromela-equvalent-P2,. . . , fPromela-equvalent-Pn represent fPromela equivalent specifications of P1, P2,. . . , Pn.

Atomic interaction covers basic send and receive operations along with additional keywords; refers, in-sequence, atomic, viewer, multiple times, pickOne, stopmessage from, wait .. until, inactivity-interval, referedDestinations, withNotification. Among these keywords,

- *viewer* and *refers* keywords are not transformed to fPRomela in that it does not add any interaction to the behavior.
- *wait .. until* and *inactivity-interval* are not transformed to fPRomela equivalent because real time representation is not modeled in Promela. A relative counter can be set instead.

The gd...dg; part in Table 5.4 covers variable parts which is interpreted as if structures in Promela. Variation points and selected variants are references to features in the extracted feature model explained in Section 4.3.1. Transformation rules and feature model construction are implemented by using python programming language.

Table 4.2: Transformation Rules

XChor	fPromela
s1 **receive from**$\{s2, s3, ..sn\}$	for each service in s2,s3,..sn
message <funct>(par1,par2...parn)	create a channel *chan chan_s1sx<funct>* where x is in range(2...n)
	with the number of parameters stated in (par1,par2...parn)
	and <funct> is the function name
	For instance, if there are 3 parameters: chan chan_s1sx<funct> = [1] of type byte, byte, byte
	Insert *chan_s1sx<funct>?par1,par2...parn;* for each service s2,s3,..sn
s1 **send** $\{s2, s3, ..sn\}$	for each service in s2,s3,..sn
message <funct>(par1,par2...parn)	create *chan chan_s1sx<funct>;* where x is in range(1...n) if it does not exist
	Insert *chan_s1sx<funct>!par1,par2...parn;* for each service s2,s3,..sn
	and <funct> is the function name

Table 4.3: Transformation Rules-cont'd

XChor	fPromela
sequence(P1 P2)	$\{fPromela - equivalent - P1\};\{fPromela - equvalent - P2\};$
paralel(P1 P2)	fPromela-equvalent-P1; fPromela-equivalent-P2;
select expr (P1 P2)	if :: fPromela-equivalent-expr ➞ if ::fPromela-equivalent-P1; :: fPromela-equivalent-P2; fi; :: else ➞ skip; fi;

Table 4.4: Transformation Rules-cont'd

XChor	fPromela
repeat expr (P1 P2)	do :: fPromela-equivalent-expr → fPromela-equivalent-P1; fPromela-equivalent-P2; :: else → break; od;
guard(expr) P1	if :: fPromela-equivalent-expr → fPromela-equivalent-P1; :: else → skip; fi;
Variability Attachment	gd :: fPromela-equivalent-vpattachment → fPromela-equvalent-P1; :: else → skip; dg;

4.3.3 Model Checking After Transformation

After the transformation, fPromela specification and related feature model of the application which includes a set of choreography and services is ready to be checked against deadlocks and assertions. No assertion is added during the transformation. Developers and testers can add user defined assertions and required system properties to fPromela specification, which can afterwards be checked against these properties with the help of SNIP tool[50].

From the src folder of the model checker SNIP, user verification can be verified with following command line code, meaning that appname.pml fPromela file is checked with appname.tvl feature model file, where appname is the name of the application:

./snip -check -fm path/to/appname.tvl path/to/appname.pml

If there are no deadlocks or assertion violations, the model works fine. Otherwise, via SNIP output, the developer can investigate the source of the problem. The process flow might not be able to continue and reach the final state because of its feature model selections and restrictions. SNIP outputs the line where the process gets stuck in the infinite loop. For instance, if one of the interactions, send or receive, is absent which results in the whole choreography is not processing any further, then a deadlock occurs. It repeats itself waiting for a result and finally produces an error indicating where the deadlock occurs as an expression with features. The feature expression or absent interaction give a clue about what to change. Correction can be done directly in fPromela and feature model or in XChor models. If XChor models or variability specifications are tailored, fPromela and feature model files can be reproduced by

Table 4.5: An excerpt of feature list for fPromela specification

```
typedef features {
      bool Cruise;
      bool Carrental;
      bool Activities;
      bool Hotel;
      bool Airline
};
features f;
```

using transformation implementation which is a time saving approach for service oriented application developers. Moreover, for semantic flow of the application, developers can define new variables and write assertion statements in order to ensure the choreographies behave as intended.

4.4 Verification of The Case Study

Case Studies provided in Section 3.2 are used here to demonstrate the verification approach step by step.

4.4.1 Travel Itinerary - Single Choreography

As a single choreography specification, travel itinerary along with its configuration interface are transformed to FTS. A TVL model file is constructed by following the steps explained in Section 4.3.1 and PML file describing *planitinerary* function of the choreography with variability in Section 4.3.2. Excerpts from constructed feature

Table 4.6: An excerpt from constructed feature model in TVL

```
root Application{
      group allOf{
            Travelitinerary group allOf{
                  Itinerary group oneOf{
                        Vacationpackage group allOf{
                              Facilities group group[0..3]{
                                    Cruise,
                                    Carrental,
                                    Activities
                              },
                              Booking group allOf{
                                    Hotel,
                                    Airline
                              },
                        },
                        Regular group allOf{
                        ...
                        }
                  },
            },
      }
}
```

model is listed in Table 4.6 and a part of fPromela specification in Table 4.7.

Travel itinerary feature model includes only features related to "Itinerary" configuration variation point and related constraints. Feature list includes only the leaf nodes of the feature model which are used in PML file in order to specify variation in behavior. Full TVL and PML file contents can be found in Appendix D.

Travel itinerary choreography realizes only one function *planitinerary* which is transformed to fPromela as an *active proctype*. In the excerpt of generated fPromela code indicated in 4.7, the variable parts dependent on *Airline* feature (f.Airline) are covered with gd..dg;. Traveler sends travel agency a trip query with startdate, enddate and details parameters in the first *chan_travelertravelagency_querytrip!startdate,enddate,details;* line. Then if *Airline* features becomes true, *chan_travelagencyairline_requestprice!startdate,enddate;* line is executed; travel agency service sends a price request to airline service along with *startdate* and *enddate* parameters.

Table 4.7: An excerpt from generated fPromela code for travelitinerary choreography of Travel Itinerary System

```
...
active proctype planitinerary() {
      {
            chan_travelertravelagency_querytrip!startdate,enddate,details;
      };
      {
            gd
            ::f.Airline ->
            chan_travelagencyairline_requestprice!startdate,enddate;
            ::else -> skip;
            dg;
      ...
      {
            if
            ::if
                  ::if
                  ::( hotelbookingconfirmation == temp_hotelbookingconfirmation
                  && flightticketconfirmation == temp_flightticketconfirmation ) ->
                  gd
                        ::f.Airline ->
                              temp = temp+1;
                              {
                              chan_travelagencyairline_bookflight!arrival,departure;
                              chan_travelagencyairline_bookflightnot?notification;
                              };
      ...
}
```

4.4.2 Biometric Security System - Multiple Choreography

Biometric Security System has three choreographies interacting with each other, namely adaptablesecuritysystem, credentialmng and alert. These three choreographies along with their configuration interfaces are transformed to FTS.

Table 4.8: An excerpt from constructed feature model in TVL

```
root Application{
      group allOf{
            Adaptablesecuritysystem group allOf{
                  Authentication_type group someOf{
                        Biometrics group allOf{
                              I_encryption_parameters group allOf{
                                    Setparams
                              },
                              I_auth_type group [1..1]{
                                    Fingerprint,
                                    Fingervein,
                                    Iris,
                                    Face
                              }
                        },
                        ...
            },
            Credentialmng group allOf{
                  opt Biometricdevice,
                  ...
            },
            Alert group allOf{
                  Emergency_notification group someOf{
                        Telephonecall,
                        Mediasend
                  },
                  ...
            }
      }
      Setparams_4 -> Setparams;
}
```

A TVL model file is constructed by following the steps explained in Section 4.3.1 and fPromela specification file indicating *verify* and *enroll* functions of adaptablesecuritysystem choreography, *alert* function of alert choreography, and *getcredentials* function of credentialmng choreography with variability in Section 4.3.2. Excerpts from constructed feature model is presented in Table 4.8 and a part of fPromela specification in Table 4.10 and Table 4.11. Adaptable security system feature model includes features for each choreography indicating their configuration variation points

and external variation points and related constraints. Feature list includes only the leaf nodes of the feature model which are used in PML file in order to depict variation in behavior. Full TVL and fPromela specification as a PML file content can be found in Appendix D.

Table 4.9: An excerpt of feature list for fPromela specification

```
typedef features {
      bool Setparams;
      bool Fingerprint;
      bool Fingervein;
      bool Iris;
      bool Face;
      bool Defaultparams;
      bool Mode_offline;
      bool Mode_online;
      bool Faketransaction
      ...
};
features f;
```

Adaptable security system choreography realizes four functions *verify*, *enroll*, *alert* and *getcredentials* which are transformed to fPromela as an *active proctype*. In the excerpt of generated fPromela code depicted in Table 4.10,

Table 4.10: An excerpt from generated fPromela code for Adaptable Security System

```
active proctype verify() {
  {
      gd
      ::(( f.Fingerprint && !f.Fingervein && !f.Iris && !f.Face) ||
      ( f.Fingervein && !f.Fingerprint && !f.Iris && !f.Face) ||
      ( f.Iris && !f.Fingerprint && !f.Fingervein && !f.Face) ||
      ( f.Face && !f.Fingerprint && !f.Fingervein && !f.Iris) ) ->
        do
        ::(noofbiometricauthtypeselected!= 0) ->
          chan_userchor_credentialmng_getcredentials!34;
          chan_userchor_credentialmng_getcredentials?processeddata;
          chan_chor_credentialmngencryption_setparams!parameters;
          noofbiometricauthtypeselected = noofbiometricauthtypeselected - 1;
        ::else -> break;
         od
            ::else -> skip;
      dg;
  };
```

Table 4.11: An excerpt from generated fPromela code for Adaptable Security System-cont'd

```
  {
      gd
      ::f.Mode_online -> temp = temp+1;
        { chan_tempthirdparty_getconnection?temp;};
        { chan_thirdpartyencryption_setparams!parameters;};
      ::else -> skip;
        dg;
  };
  ...
}
active proctype enroll() {
  ...
  {
      gd
      ::f.Mode_online -> chan_encryptionthirdparty_savehasheddata!hasheddata;
      ::else -> skip;
      dg;
  };
  {
      gd
      ::f.Mode_offline -> chan_encryptionstorage_sethasheddata!hasheddata;
      ::else -> skip;
      dg;
  };
  ...
}
active proctype alert() {
      ...
      {
            gd
            ::f.Telephonecall -> chan_cameraalertsender_call!destination;
            ::else -> skip;
            dg;
      };
      ...
}
active proctype getcredentials() {
      {
            gd
            ::f.Biometricdevice -> temp = temp+1;
            {chan_tempconnection_connectdevice!deviceid;};
            ...
            ::else -> skip;
            dg;
      };
      ...
}
```

the variable part condition related to *Fingerprint*, *Fingervein*, *Iris* and *Face* features represents that if one of the features are selected, then the do..od; repeating part is executed until *noofbiometricauthtypeselected* is zero. After this part is executed or

skipped, if *Mode_online* feature is true, the variable part is executed sequentially; first *"chan_tempthirdparty_getconnection?temp;"* and then *" chan_thirdpartyencryption _setparams!parameters;"*. In other words, after thirdparty service receives a getconnection message, it sets the key information as parameters to encryption service.

After TVL and fPromela files are generated, they are ready to be verified by the SNIP tool. Travel itinerary system has no deadlocks or assertion violations, so the output of the SNIP tool is as follows indicating that 772 states are explored while executing possible choreoraphies:

```
No never claim, checking only asserts and deadlocks..
No assertion violations or deadlocks found
 [explored 772 states, re-explored 0].
```

There are several reasons why a deadlock occurs in choreographies; for instance a receive without a send action and the number of receives exceeding the available channel size. It can be thought that when a *send* interaction is specified, there is an implied *receive* associated with that *send* interaction. However, if only a *receive* interaction is defined, there is no way to make sure an associated *send* interaction exists. This situation results in a deadlock and whole choreography cannot process any further.

When there is no associated *send* interaction with the following *receive* statement as in "responsewindow receive fromcomparison message show()", SNIP tool outputs information about (i) where the deadlock occurs, (ii) which feature selection causes this deadlock and (iii) the program stack information provided as follows:

```
No never claim, checking only asserts and deadlocks..
Found deadlock [explored 42 states, re-explored 0].
 - Products by which it is violated (as feature expression):
   (Videorecord & Biometricdevice & Faketransaction &
!Fingerprint & Fingervein) | (Videorecord & Biometricdevice
& Faketransaction & Fingerprint & !Fingervein)

 - Stack trace:
   features                          = /
   globals.temp                      = 0
   globals.notification              = 0
   ...
```

```
   ...
 -- Final state repeated in full:
   features                          = (Videorecord &
Biometricdevice & Faketransaction & !Fingerprint & Fingervein) |
(Videorecord & Biometricdevice & Faketransaction & Fingerprint
& !Fingervein)
   globals.temp                      = 0
   globals.notification              = 0
  ...
   globals.usernamepass              = 34
   globals.fakeinterface             = 1
   globals.temp_fakeinterface        = 1
   pid 00, encryption                @ end
   pid 01, credentials               @ end
   pid 02, verify                    @ NL129
    --
```

The deadlock can be resolved in two different ways.

1. Convert receive interaction *"responsewindow receive fromcomparison message show()"* to a send interaction *"comparison sendresponsewindow message show()"*.
2. Add a new send interaction before receive: *"comparison sendresponsewindow message show()"* preceeds *"responsewindow receive fromcomparison message show()"*.

Developers can also provide their own TVL feature models to be used in verification in SNIP. Transformation from XChor to fPromela does not add any assertion to the behavior model. Developers can insert additional assert statements to verify the system semantically. Assertion violations can result from wrong semantics of TVL feature model.

The TVL model of adaptable security system choreography, mode_online and mode_offline are optional variants of authentication_mode. In adaptable security system, *result* is defined for newly inserted assertion statement as *"int result;"*. In verify function, result indicates that "user" is verified when it equals to one or that "user" is rejected when it equals to zero. In the following setting of adaptive security system, the "comparison" service returns zero as the comparison result which is the case for mode_offline.

```
active proctype comparison() {
      chan_tempcomparison_compare!0;
}

active proctype verify() {
      ...
      gd
      ::f.Mode_online ->
      temp = temp+1;
      {
            chan_encryptionthirdparty_verify!1;
            chan_encryptionthirdparty_verify?result;
      };
      {
            gd
            ::f.Faketransaction ->
            chan_thirdpartycomparison_fakeanalysis!1;
            chan_thirdpartycomparison_fakeanalysis?fakeinterface;
            ::else -> skip;
            dg;
      };
      ::else -> skip;
      dg;

      gd
      ::f.Mode_offline ->
      temp = temp+1;
      {
            chan_encryptionstorage_gethasheddata!temp;
            chan_storagecomparison!temp;
      };
      {
            chan_tempcomparison_compare?result;
            assert(result == 1);
      };
      {
            gd
            ::f.Faketransaction ->
            chan_storagecomparison_fakeanalysis!comparisonresult;
            ::else -> skip;
            dg;
      };
      ::else -> skip;
      dg;
...
}
```

As the optional behavior of mode_online and mode_offline variants, there is a possible selection that two of them are set to be true simultaneously. Then, the variable parts guarded by f.Mode_online and f.Mode_offline conditions are executed sequentially. First the result coming from "thirdparty" service is set to one because f.Mode_online part is executed. Then with the result sent by "comparison" service, the value of result is set to zero. The assertion ensuring that whether the result came from "thirdparty" service or not is violated. SNIP output of assertion violation is depicted as follows:

```
No never claim, checking only asserts and deadlocks..
Assertion at line 139 violated [explored 21 states, re-explored 0].
 - Products by which it is violated (as feature expression):
   (Faketransaction & !Fingerprint & Fingervein & Mode_offline &
Mode_online) | (Faketransaction & Fingerprint & !Fingervein &
Mode_offline & Mode_online)
...
   globals.temp_fakeinterface          = 1
   pid 00, encryption                  @ end
   pid 01, comparison                  @ end
   pid 02, credentials                 @ end
   pid 03, verify                      @ NL139
   pid 04, enroll                      @ NL198
   pid 05, alert                       @ NL230
   pid 06, getcredentials              @ NL265
    --
```

The system generates two different results values due to an error of TVL semantics because two different comparisons are executed at the same time. Therefore, mode_online and mode_offline features should be mutually exclusive and the relation between them should be changed from optional to alternative in the TVL feature model.

4.5 Discussion

It might seem that constructs are easily converted to fPromela equivalent ones; however structural activities preserving sequential behavior need more effort due to native

concurrent behavior of fPromela. Transforming more than one choreography to a single fPromela specification is another challenging issue. Moreover, transforming variability associations and their logic are handled choreography-wide. Fault handling requires further research and will be also transformed into fPromela.

Table 4.12: Verification Results

Choreography				Verification		
Choreography	Function	Interact	Variation Point	TT (ms)	EX (ms)	ES
1	1	10	1	12	447	41
1	1	20	1	20	483	316
1	2	20	1	111	437	2356
1	1	10	3	29	427	81
1	1	20	3	159	595	772
1	2	20	3	146	589	17408
1	1	10	5	74	462	337
1	1	20	5	81	496	1636
1	2	20	5	195	963	32100
2	2	15	5	439	758	10082
2	2	30	5	529	1200	37244
2	2	30	10	307	5734	114926
3	3	30	5	940	2109	69314
3	3	30	10	704	7671	109336
3	4	30	5	638	9352	323331
3	4	30	10	334	46704	509925

A set of experiments are conducted with one to three choreographies (Chor) including one to four functions with totally ten to thirty service interactions (Interact) and variation points (internal, external and configuration) ranging from one to ten. For one choreography experimentation travel itinerary choreography is utilized, while two and three choreography experiments adaptable security system is used. Transformation times (TT) based on our approach, the execution times (EX) of verification with SNIP in milliseconds and the number of explored states (ES) are represented in Table 4.12.

The results show that the execution time to transform and prepare FTS models in-

creases with the number of variations, choreographies and functions along with their interactions. Verification is getting slower with the increase in the variation, due to increase in the number of explored states. Because each choreography function is executed as a parallel active process in FTS, an increase in function number results in increase in execution time and explored states. Considering execution times in milliseconds, verification of variable choreographies takes reasonable effort even in case of increasing the numbers of variations, choreographies, functions and interactions. Our approach provides the verification basis for variable XChor choreographies which can then be enhanced by user defined assertion and property additions.

CHAPTER 5

TRANSFORMATION OF XCHOR MODELS TO EXISTING LANGUAGES

This chapter comprises a step by step approach for transforming XChor models to existing languages, namely BPEL4Chor, VxBPEL and BPEL. Existing choreography or orchestration languages can not fulfill the requirements of variability specification in choreography and variability association with interacting orchestrations. Although XChor Language is a richer model, VxBPEL and BPEL4Chor are closer specifications to XChor representing variable orchestrations and choreography respectively. For VxBPEL, the lack of information about internalization or externalization of variation points restricts our approach. Whole COVAMOF model should be provided along with VxBPEL specification. Moreover, XChor variability attachment specifications have a complex structure to directly transform to VxBPEL variability specification easily. In order to overcome this situation, an additional mechanism is needed. For BPEL4Chor, there is no support for variability specification which requires a variability handling mechanism other than choreography specification. Transforming to BPEL4Chor model after resolving all variability is a solution to this situation. On the other hand, BPEL4Chor model can be transformed to XChor models without variability specification. Therefore, XChor models are not directly transformed fully to existing languages and transformations are processed along with assumptions and further requirements.

5.1 Transformation to BPEL4Chor, VxBPEL and BPEL

The target models can not satisfy representation of variable XChor models wholly. Therefore, first differences and similarities between models are discussed, then assumptions and requirements for the transformation are stated in the following subsections.

5.1.1 Differences and Similarities Between Models

5.1.1.1 BPEL4Chor and XChor Models

Forming Choreographies Creating and generating choreographies using BPEL4Chor Language can be top-down and bottom-up. In the top-down approach, a choreography including topology and grounding is specified in BPEL4Chor language and all interacting orchestration skeletons as abstract business processes are created. In bottom-up approach, a choreography specification in BPEL4Chor can be generated from a set of already abstract process definitions of BPEL orchestrations. Message links and technical information in groundings are gathered from WSDL interfaces of orchestrations. Like BPEL4Chor, choreographies can be formed and gathered either top-down or bottom-up in XChor. However, XChor does not include all technical information required for choreography grounding, therefore the skeleton of choreography grounding is generated with blanks to filled later by developers. The developer should fill the blanks with the appropriate function names in WSDL. In the presence of the abstract process definition in a configuration interface, the BPEL orchestration skeleton is constituted using this definition and parts of choreography grounding is filled through this information.

Multiple Choreography Specification BPEL4Chor allows developers to define only one choreography, whereas XChor facilitates to define and reuse multiple choreographies. Therefore, for each choreography in XChor, there should be one BPEL4Chor choreography topology and grounding models. To interrelate all choreographies in XChor, each choreography specification in BPEL4Chor reveals its interface as a service to enroll in the ultimate choreography in BPEL4

Chor.

Executability Netiher XChor nor BPEL4Chor models are executable. BPEL4Chor is transformed to BPEL abstract process definitions which requires further specification for executable processes. Likewise, XChor models are not relying on any executable structure.

Service Interactions BPEL4Chor has *bindSenderTo* to send a link from source to destination. Besides, it is used for the realization of *refersTo(<a set of services>) withnotification* in XChor. With *partifipantRefs*, when service A sends a message to service B, if service C is defined in *partifipantRefs*, then service B can directly interacting with service C and service A is out of concern.

Event Handling BPEL4Chor is based on BPEL specification which includes event handling mechanism. Therefore the whole choreography with orchestration specifications can specify required events, whereas XChor has no support of event handling.

Fault Handling BPEL4Chor is based on BPEL specification which enables the definition and handling of faults, while XChor has a basic fault specification mechanism which can be mapped to that of BPEL4Chor.

5.1.1.2 VxBPEL and XChor Models

Abstract Process Definition Configuration files of services comprise abstract process definitions if specified which can be seen as BPEL abstract processes with variability. VxBPEL does not specify abstract business processes. However, from abstract process definitions of services, a skeleton of VxBPEL orchestrations can be generated with variability information specified in COVAMOF.

Variability Representation Some variability specifications in abstract process definitions can not directly be transformed to VxBPEL equivalent ones due to different characteristics of variability reference mechanism in service compositions, namely the variability attachment structure.

Interface Variability VxBPEL does not address interface variability, while XChor has the ability to alter service and choreography functions and parameters via

their configuration interfaces. VxBPEL can only be used for variability in composition in XChor context.

5.1.1.3 BPEL and XChor Models

Service Interactions Main goal of BPEL specifications is to represent composition among collaborating services at orchestration level. Therefore, BPEL is capable of designating all service interactions specified in XChor choreography specifications and interactions stated in *abstract business process* part of the services.

Variability Representation Only the parts without variation indicating service interactions can be transformed to BPEL specifications due to lack of variability support in BPEL.

5.1.2 Assumptions and Requirements for Model Transformation

5.1.2.1 XChor Models to BPEL4Chor Transformation

- All interactions have sender and receiver specification.
- No unbound variation point exists in XChor choreography specification, because BPEL4Chor does not have any support for variability management mechanism. All variation points of XChor choreography specification should be resolved and bound before transforming to BEPL4Chor.
- All choreographies should have a service interface in which choreography functionalities are declared. This is a must for multiple choreography interaction.
- WSDL files of interacting services and choreographies are created for web services from technical point of view which resides in grounding part. But these creations in transformation is not a must.
- BPEL4Chor can interrelate VxBPEL variable orchestrations by means of orchestrations' provided interfaces without variability.

- If any choreography has variability in choreography specification, an error is generated and transformation of XChor models to BPEL4Chor is suspended.

- BPEL4Chor enables the specification of different types explicitly via *participantTypes*, whereas XChor assumes that every service has its own type. In other words, every service represents a different type in XChor.

- All interacting services have *abstract process definition* parts in their configuration interfaces. In BPEL4Chor, topology model includes only message links and all structured interaction logic reside in the abstract business process definitions of services. However, in XChor all interactions are modeled within choreography specification if service configuration interfaces do not include *abstract process definition* parts. Therefore, the behavioral logic of each interacting service should be specified and be consistent with choreography specification.

5.1.2.2 BPEL4Chor Models to XChor Transformation

- BPEL4Chor follows single choreography approach, so there is only one choreography specification defined.

- As BPEL4Chor does not have any support for variability management mechanism, no variability in configuration interface for service and choreography is specified.

- WSDL files should be provided which specifies service functions and their parameters. The service functions are used to construct service interfaces.

- XChor supports a a basic fault handling mechanism and does not support event handling. Therefore, the parts specifying event handling mechanism in BPEL4Chor are not transformed to XChor.

- The global view indicating control flow is not provided by BPEL4Chor models. Therefore, to help constructing choreography specifications, messageLink specifications are specified sequentially.

5.1.2.3 XChor Model to VxBPEL Transformation

- XChor choreographies are transformed to a set of variable VxBPEL orchestrations.
- XChor service orchestrations, transformed to VxBPEL equivalent, are these that have no variability mapping specification across a choreography.
- Due to structural differences in variability specification, an additional configuration file which includes variability attachment logic should be generated after transformation.
- All transformed VxBPEL orchestrations have a service interface; it can be a WSDL file.

Table 5.1: Mapping of Variability Modeling of XChor and VxBPEL

XChor	**COVAMOF**
Variation Point	Variation Point
External Variation Point	None
Internal Variation Point	Variation Point
Configuration Variation Point	Configurable Variation Point
Variant	Variant
Constraint	Dependency

5.2 The Transformation Approach to BPEL4Chor, VxBPEL and BPEL

After binding variability of choreographies, two approaches can be applied:

1. The choreographies are transformed to BPEL4Chor choreography models and interacting services are transformed either to BPEL or VxBPEL depending on whether configuration interfaces exist or not.
2. The choreographies are transformed to VxBPEL variable orchestrations and interacting services are transformed either to BPEL or VxBPEL depending on whether configuration interfaces exist or not.

5.2.1 Transformation to BPEL4Chor

Each XChor choreography specification and its interacting services are transformed to BPEL4 Chor models; a topology, a grounding and a set of abstract process definitions due to lack of support for multi choreography concept in BPEL4Chor. Besides, XChor supports multi functionality in choreography specification, so there should be one BPEL4Chor topology for each defined function in the choreography specification. Topology model, comprising participants and message links, is constructed from choreography specification. Grounding model and wsdl definitions are formed with the help of interacting service interfaces. Participant Behavior Description model is created from the *abstract process definition* part of each interacting services. However, XChor does not force to define all send activities corresponding to receive ones. Therefore, all related corresponding send and receive activities are created during transformation. If any interacting service has unbound variability, then the service is transformed to VxBPEL specification which is described in Section 5.2.3. Transformation rules are listed in Table 5.2 and Table 5.4 for the Topology model, Table 5.5 for the Grounding model, and Table 5.7 for the Abstract Business Process Definition model. Table 5.7 displays the case if a service starts with its interaction with a *receive* interaction, then *createInstance = "yes"* is added within receive tag.

In *Topology*, message links describe the connection between two services along with their activities; send and receive. The technical part, the binding of activities to real wsdl functions is done in *Grounding*. Therefore, there is always a corresponding entry in grounding related with a message link.

5.2.2 Transformation from BPEL4Chor

BPEL4Chor models defining a choreography; namely topology, grounding and participant behavior description model are transformed to XChor models; choreography specification, service interfaces, configuration interfaces of services respectively. As BPEL4Chor models define only one choreography through topology model, only one choreography specification in XChor is created. The topology model specifies only message links between service, so the control flow semantics are scattered over

Table 5.2: Rules for Transformation to BPEL4Chor

XChor Choreography Model	*BPEL4Chor Topology Model*
Choreography name	**Topology name**
choreography <chorname> ... *Function* <functionname>: ...	<topology name=”<chorname_functionname>” targetNamespace="http://example.com/configuredchoreography/"<chorname>" xmlns="urn:HPI_IAAS:choreography:schemas:choreography:topology:2006/12" xmlns:chordef="http://example.com/configuredchoreography/"<chorname>" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:HPI_IAAS:choreography:schemas:choreography: topology:2006/12 http://www.iaas.uni-stuttgart.de/schemas/bpel4chor/topology.xsd"> ... </topology>

Table 5.3: Rules for Transformation to BPEL4Chor-cont'd

Interacting services and choreographies	**Participants**
use choreography <chorname1> *import service* <service1>	<participants> <participant name="<chorname1" type="<chorname1>_type"/> <participant name="<service1>" type="<service1>_type"/> </participants>
Types of interacting services and choreographies	**participantTypes**
	<participantTypes> <participantType name="< chorname1>_type" participantBehaviorDescription="< chorname1>:< chorname1>"/> <participantType name="<service1>_type" participantBehaviorDescription="<service1>:<service1>"/> </participantTypes>

Table 5.4: Rules for Transformation to BPEL4Chor-cont'd

XChor Choreography Model	*BPEL4Chor Topology Model*
Choreography specification	**Participant Types**
Partner type definitions for multiple send	<participantType name="< function>_<sender>_type" participantBehaviorDescription="< function>_<sender>:< function>_<sender>"/>
Partner Set definition	<participantSet name="< function>_<sender>" type="< function>_<sender>_type"> <participant name="<service2>" scope="<service2>:<function>_FE" /> for each receiver service2,service3,...servicen </participantSet>
Partner type definitions for multiple receive	<participantType name="< function>_<sender>_type" participantBehaviorDescription="< function>_<sender>:< function>_<sender>"/>
Partner type definitions for multiple receive	<participantType name="< function>_<receiver>_type" participantBehaviorDescription="< function>_<receiver>:< function>_<receiver>"/>

Table 5.5: Rules for Transformation to BPEL4Chor-cont'd

XChor Choreography Model	*BPEL4Chor Topology Model*
Interacting services and choreographies	**Message Links**
<service1> send $\{< service2 >\}$ message <function>(<parameters>)	<messageLink name="<function>Link" sender="<service1>" sendActivity="get<function>" receiver="<service2>" receiveActivity="<function>" messageName="<function>"/>
<service1> receive from $\{< service2 >\}$ message <function>(<parameters>)	<messageLink name="<function>Link" sender="<service1>" sendActivity="get<function>"

Table 5.6: Rules for Transformation to BPEL4Chor-cont'd

XChor Choreography Model	*BPEL4Chor Grounding Model*
For every messagelink created in topology a corresponding groundign specification	<messageLink name="<receiverfunction>Link" porttype = "ns:receiver_pt" operation = <receiverfunction/>
<service1> send $\{< service2 >\}$ message <function>(<parameters>)	PBD of service1 <"invoke" wsu:id="<function>"/>
<service1> send send $\{< service2 >, < service3 >, .. < servicen >\}$ message <function>(<parameters>)	PBD of service1 < forEach wsu:id="<function>FE" paralel = "yes"> <scope wsu:id="<function>Scope"> <sequence> <invoke wsu:id="<function>"> </sequence> </scope> <forEach>

Table 5.7: Rules for Transformation to BPEL4Chor-cont'd

XChor Choreography Model	*BPEL4Chor Participant Behavior Description (PBD)*
<service1> receive from $\{< service2 >\}$ message <function>(<parameters>)	PBD of service2 <receive wsu:id="<function>" createInstance = "yes"/> or <receive wsu:id="<function>" />
<service1> receive from $\{< service2 >, < service3 >, .. < servicen >\}$ message <function>(<parameters>)	PBD of service1 < forEach wsu:id="<function>FE" paralel = "yes"> <scope wsu:id="<function>Scope"> <sequence> <receive wsu:id="<function>" createInstance = "yes"/> </sequence> </scope> <forEach>

Table 5.8: Rules for Transformation to BPEL4Chor-cont'd

XChor Choreography Model	*BPEL4Chor Participant Behavior Description (PBD)*
repeat expr (P1 P2)	<while> <condition opaque = "yes"> BPEL4Chor-equivalent P1 BPEL4Chor-equivalent P2 </while>
sequence (P1 P2)	<sequence> BPEL4Chor-equivalent P1 BPEL4Chor-equivalent P2 </sequence>
paralel expr (P1 P2)	<flow> BPEL4Chor-equivalent P1 BPEL4Chor-equivalent P2 </flow>
guard(expr) P1	<if> <condition opaque = "yes"> BPEL4Chor-equivalent P1 </if>

participant behavior descriptions. The sequence of the participantSet specifications and scoping mechanism are used to reveal which participant behavior descriptions are interacting with each other. Type definitions and partnerSet specifications are not covered by XChor, so these are not transformed. While service interactions are formed by message links in topology model, concrete service functions are taken from grounding model and control flow semantics come from participant behavior description in order to create choreography specification. However, still developer intervention is needed to preserve control flow semantics, because although message links are provided in topology, the control flow of the global interaction is missing. Service interfaces are constructed based on the grounding model. For each participant behavior description model, one configuration interface without variability specification is constructed. Abstract process definition part of service configuration interface is filled with participant behavior description model content. Transformation rules are listed in Table 5.9 for Choreography Specification, Table 5.10 for the Configuration Interface, and Table 5.13 for the the Service Interfaces.

As XChor enables to define more than one function for a choreography, a <functionname> should be provided by developer describing the whole service interaction in BPEL4Chor. MessageLinks in Topology model cannot specify service composition on its own, Participant Behavior Descriptions are used for revealing control flow semantics. The process of forming choreography specification is as follows:

1. Follow the *messageLink* specification sequence in Topology model.
2. Start from the first *messageLink* specification within *messageLinks*.
3. Process Participant Behavior Descriptions of *sender* and *receiver* and find *sendActivity* and *receiveActivity* in BPEL specification.
4. If there is a structured activity above this service interaction, then use this structured activity in choreography specification as is.
5. Check the consistency of structured activity including these activities in Participant Behavior Descriptions of *sender* and *receiver*. If they are not consistent with each other, add this new structured activity of callee service to the choreography specification.

Table 5.9: Rules for Transformation from BPEL4Chor

BPEL4Chor Topology Model	*XChor Choreography Model*
Choreography name	**Topology name**
<topology name=”<chorname>”	*choreography* <chorname>
targetNamespace= "http://example.com/	...
configuredchoreography/"<chorname>"	...
</topology>	...
Participants	**Interacting services and choreographies**
<participants>	
<participant name="<service1>" type="<servicetype>_type"/>	*import service* <service1>
</participants>	
participantTypes	
<participantTypes>	
<participantType name="<servicetype>" participantBehaviorDescription="<service1>:<service1>"/>	
</participantTypes>	
	Choreography Function
	Function <functionname>:
	where <functionname> is provided by developer

6. Process the next messageLink and go to step 3.

5.2.3 Transformation to VxBPEL and BPEL

There are two cases; transforming from XChor choreography and transforming from abstract process definition for any XChor service. In the XChor choreography transformation case, for each function of the choreography specification there is one equivalent VxBPEL variable orchestration with its interface represented in WSDL. In abstract process definition of service case, only one VxBPEL variable orchestration is created as an abstract process in BPEL with variability attachments. Interactions in functions of choreography and abstract business processes are transformed to VxBPEL or BPEL-equivalent ones with the help of transformation rules listed in Table 5.14, Table 5.16 and Table 5.17. If any unbound variability exist in service's configuration file, then abstract process definition is converted to VxBPEL-equivalent, otherwise to BPEL-equivalent. Variation point specifications in XChor configuration interfaces and variability attachments indicating variable parts of choreography specification are transformed to VxBPEL ones as displayed in Table 5.18 and Table 5.20.

VxBPEL configures service composition based on specified configurable variation points which are realized by variation points defined within an orchestration. Configuration variation points in configuration interfaces are directly converted to Configurable variation points, however the relation between variants can not be specified, such as mandatory, optional or alternative. In XChor the variable parts are labeled with variability attachments which can include more than one variation point and related variant selection. For instance "#vp i_auth_type ifOneSelected(fingerprint fingervein iris face)" and "i_auth_mode ifSelected (mode_online)#". However in VxbPEL inline variability enables to define simple variation point and variant specifications such as one variation point along with multiple variants. We can not easily settle complex variability logic like in "#vp i_auth_type ifOneSelected(fingerprint fingervein iris face) and i_auth_mode ifSelected(mode_online)#". The complex logic can be converted to VxBPEL one, but the VxBPEL code gets complicated. Therefore, variability attachment transformation is changed with regard to its complexity.

Table 5.10: Rules for Transformation from BPEL4Chor- cont'd

BPEL4Chor Participant Behavior Description (PBD)	*XChor Configuration Interface*
For each PBD	
<process ...	Configuration interface vconf_<servicename> of service <servicename>
name=<servicename> ... >	
PBD of service1	<service1> receive from <service2>
<receive wsu:id="<function>"	message <function>(<parameters>)
	<parameters> information comes from WSDL interfaces
	and <service2> information comes from messageLinks
PBD of service1	<service1> send <service2>
<reply wsu:id="<function>"	message <function>(<parameters>)
	<parameters> information comes from WSDL interfaces
	and <service2> information comes from messageLinks

Table 5.11: Rules for Transformation from BPEL4Chor- cont'd

BPEL4Chor Participant Behavior Description (PBD)	*XChor Configuration Interface*
PBD of service1	
< forEach wsu:id="<id>" paralel = "yes">	<service1> receive from
<scope wsu:id="<scopeid>">	$\{< service2 >, < service3 >, ...\}$
<sequence>	message <function>(<parameters>)
<receive wsu:id="<function>"	where <service2>, <service3>,...
createInstance = "yes"/	information comes from participantSet specification>
</sequence>	related with <scopeid>
</scope>	and <parameters> information
<forEach>	comes from WSDL interfaces
<while>	**repeat** expr (P1 P2)
<condition opaque = "yes">	
XChor-equivalent P1	
XChor-equivalent P2	
</while>	

Table 5.12: Rules for Transformation from BPEL4Chor- cont'd

BPEL4Chor Participant Behavior Description (PBD)	*XChor Configuration Interface*
<sequence> XChor-equivalent P1 XChor-equivalent P2 </sequence>	**sequence** (P1 P2)
<flow> XChor-equivalent P1 XChor-equivalent P2 </flow>	**paralel** expr (P1 P2)
if <condition opaque = "yes"> BPEL4Chor-equivalent P1 </if>	**guard**(<exp>) P1 where <exp> is provided by developer

Table 5.13: Rules for Transformation from BPEL4Chor- cont.

WSDL Files	*XChor Service Interface Model*
<definitions name="<service1>"	Service interface <service1>
For each message	
<message name="<functionname>">	function <functionname>
<part name="<paramname1>"/>	input (<paramname1>,<paramname2>,...)
<part name="<paramname2>	
</message>	
...	...
<service name="<servicename>">	portName <servicename> binding <bindinglocation>
<port binding ="tns:functionaname_Binding" name="<service1>">	
<soap:address location="<bindinglocation>">	
</port>	

Table 5.14: Rules for Transformation to VxBPEL and BPEL

XChor Choreography Model	*BPEL Model*
Choreography specification	**...**
<service1> send $\{< service2 >\}$ message <function>(<parameters>)	<bpel:invoke aei:id="<id>" name="<service2>" operation="<function>" partnerlink="<function>" inputVariable="<parameters>"/>
<service1> send $\{< service2 >, < service3 >, .. < servicen >\}$ message <function>(<parameters>)	<bpel:flow> <bpel:invoke aei:id="<id>" name="<service2>" operation="<function>" partnerlink="<function>" inputVariable="<parameters>"/> <bpel:invoke aei:id="<id>" name="<service3>" ... <bpel:invoke aei:id="<id>" name="<servicen>" ... </bpel:flow>
<service1> send <service2> message <function>(<parameters>) %<context element> = <service2>.<function>%	<bpel:invoke aei:id="<id>" name="<service2>" operation="<function>" partnerlink="<function>" outputVariable="<context element>" inputVariable="<parameters>">

Table 5.15: Rules for Transformation to VxBPEL and BPEL

XChor Choreography Model	*BPEL Model*
<service1> receive from $\{< service2 >\}$ message <function>(<parameters>)	<bpel:receive aei:id="<id>" name="<service1>" operation="<function>" partnerlink="<function>" variable="<parameters>"/>
<service1> receive from $\{< service2 >, < service3 >, .. < servicen >\}$ message <function>(<parameters>)	<bpel:flow> <bpel:receive aei:id="<id>" name="<service1>" operation="<function>" partnerlink="<function>" variable="<parameters>"/> <bpel:receive aei:id="<id>" name="<service1>" ... <bpel:receive aei:id="<id>" name="<service1>" ... </bpel:flow>
<service1> send $\{< service2 >\}$ message <function>(<parameters>) withNotification	<bpel:invoke aei:id="<id>" name="<service2>" operation="<function>" partnerlink="<function>" inputVariable="<parameters>" outputVariable="<notification>"/>

Table 5.16: Rules for Transformation to VxBPEL and BPEL- cont'd

XChor Choreography Model	*BPEL Model*
<service1> send $\{< service2 >\}$ message <function>(<parameters>) referedDestinations $\{< service3 >, .. < servicen >\}$	<bpel:invoke aei:id="<id>" name="<service2> operation="<function>" partnerlink="<function> inputVariable="<parameters>"/> <bpel:flow> <bpel:invoke aei:id="<id>" name="<service3> ... <bpel:invoke aei:id="<id>" name="<servicen> ... </bpel:flow>
<service1> send $\{< service2 >, < service3 >, .. < servicen >\}$ message <function>(<parameters>) in-sequence	<bpel:sequence> <bpel:invoke aei:id="<id>" name="<service2> operation="<function>" partnerlink="<function> inputVariable="<parameters>"/> <bpel:invoke aei:id="<id>" name="<service3> ... <bpel:invoke aei:id="<id>" name="<servicen> ... </bpel:sequence>

Table 5.17: Rules for Transformation to VxBPEL and BPEL- cont'd

XChor Choreography Model	*BPEL Model*
parallel(Atomic or Composite Interaction)	<bpel:flow aei:id="<id>"> BPEL-equivalent Interaction-code </bpel:flow>
sequence(Atomic or Composite Interaction)	<bpel:sequence aei:id="<id>"> BPEL-equivalent Interaction-code </bpel:sequence >
repeat <expr> (Atomic or Composite Interaction)	<bpel:while aei:id="<id>"> <bpel:condition>"<BPEL-equivalent expr>"</bpel:condition> BPEL-equivalent Interaction-code </bpel:while>
guard <expr> Atomic or Composite Interaction	<bpel:if aei:id="<id>"> <bpel:condition>"<BPEL-equivalent expr>"</bpel:condition> BPEL-equivalent Interaction-code </bpel:if>

Table 5.18: Rules for Transformation to VxBPEL and BPEL- cont'd

XChor Choreography Model	*VxBPEL Model*
<Simple Variability Attachment>	
(i) #vp <VP1>	<vxbpel:VariationPoint name = VP1>
ifSelected(<V1>)	<vxbpel:Variants>
... # Atomic or Composite Interaction	<vxbpel:Variant name = V1>
(ii)#vp <VP1>	<vxbpel:VPBpelCode>
ifOneSelected(<V1>)	BPEL-equivalent
... # Atomic or Composite Interaction	interaction-code
(iii) #vp <VP1>	</vxbpel:VPBpelCode>
ifAllSelected(<V1>)	</vxbpel:Variant>
... # Atomic or Composite Interaction	</vxbpel:Variants>
	</vxbpel:VariationPoint>

Table 5.19: Rules for Transformation to VxBPEL and BPEL- cont'd

XChor Choreography Model	*VxBPEL Model*
	<vxbpel:VariationPoint name = VPCounter>
	<vxbpel:Variants>
	<vxbpel:Variant name = VCounter>
#vp	<vxbpel:VPBpelCode>
<Complex Variability Attachments>	BPEL-equivalent
... # Atomic or Composite Interaction	interaction-code
	</vxbpel:VPBpelCode>
	</vxbpel:Variant>
	</vxbpel:Variants>
	</vxbpel:VariationPoint>

Table 5.20: Rules for Transformation to VxBPEL and BPEL- cont'd

XChor Choreography Model	*VxBPEL Model*
	<vxbpel:ConfigurableVariationPoint id = "<id>" default-Variant = "<defaultVariant>"> <vxbpel:Name>"<explanation>"</vxbpel:Name> <vxbpel:Rationale>...</vxbpel:Rationale> <vxbpel:Variants> <vxbpel:Variant name = "<name>"> <vxbpel:VariantInfo>...</vxbpel:VariantInfo> <vxbpel:RequiredConfiguration> <vxbpel:VPChoices> <vxbpel:VPChoice vpname = "<vpname>" variant = "<variant>"/> ... </vxbpel:VPChoices> </vxbpel:RequiredConfiguration> </vxbpel:Variant> </vxbpel:Variants> </vxbpel:ConfigurableVariationPoint>

For simple cases the variability is converted to VxBPEL equivalent one, for complex cases a new variation point is created automatically and the logic of the complex variability attachment is stored in a newly created configuration file. For instance if there is a variation attachment residing in choreography specification, then vp_1 and v_1 are created and an assignment is added to the configuration file.

For configuration of VxBPEL variable orchestrations, an additional configuration mechnasim needed to analyze both VxBPEL variability specification in configurable variation points and assignments in configuration file. Contents of the file is as follows:

```
vp_1.v_1 = #vp i_auth_type ifOneSelected( fingerprint fingervein iris face) #
```

Transformation to BPEL4Chor, VxBPEL and BPEL are applied to travel itinerary system case study and related models and the following files are generated and can be found in Appendix E:

- Generated BPEL4Chor files: topology and grounding for travelitinerary choreography and participant behavior description for travel agency,
- Generated VxBPEL file for travelitinerary choreography and its BPEL model after selection of "airline" variant of "booking" and "activities" variant of "facilities" variation point.

CHAPTER 6

VARIABILITY MANAGEMENT IN SOFTWARE PRODUCT LINES WITH XCHOR

This chapter represents the usage of XChor models in order to provide a solution to achieve management of variability in Software Product Lines. Software product lines (SPL) is a set of methods, tools and techniques for developing similar software systems from a common set of software assets via systematic reuse of commonalities and management of variability. The coarse building blocks of the system, software assets define composition rules between a collection of artifacts to achieve a common goal plus a set of variability. The key issue for effective creation of products is reusing SPL architecture, both in asset and artifact levels by extending and configuring through variation points. In the asset level, specification of variability, its effects on composition, its relation with other asset and artifact variability need to be addressed. In artifact level, variation should be exposed to be configured by assets. Due to the complex nature of variability, providing a consistent configuration of assets and artifacts is a challenging issue. To address this challenge, XChor is applied to Software Product Lines approach as a way to cope with variation specifications and integration of these in both asset and artifact levels.

6.1 Software Product Lines

A software product line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed

way[52]. An increasing number of organizations are building their products within product line settings in order to achieve large scale productivity gains, improve time to market, maintain a market presence, compensate for an inability to hire, leverage existing resources, and achieve mass customization.

In January 1997, the Carnegie Mellon Software Engineering Institute (SEI) launched the Product Line Practice Initiative to help facilitate and accelerate the transition to sound software engineering practices using a product line approach. The goal of this initiative is to provide organizations with an integrated business and technical approach to systematic reuse, so they can produce and maintain similar systems of predictable quality more efficiently and at a lower cost.

A key strategy for achieving this goal has been the creation of a conceptual framework for product line practice. The SEI Framework for Software Product Line Practice describes the foundational product line concepts and identifies the essential activities and practices that an organization must master before it can expect to successfully field a product line of software or software intensive systems. The framework is a living document that is evolving as experience with product line practice grows.

The framework's contents are based on information gathering workshops, extensive work with collaboration partners, surveys and investigations, and continued research. The SEI has also incorporated practices reported at its international Software Product Line Conferences and collected from the community.

In March 1998, the SEI hosted its first Department of Defense (DoD) product line practice workshop, "Product Lines: Bridging the Gap—Commercial Success to DoD Practice". Topics discussed and documented included DoD barriers and mitigation strategies, and similarities and differences between DoD product line practice and commercial product line practices. Subsequent workshops were held in successive years[75].

Software product lines capitalize on the commonalities and bounded variabilities among similar products, can address problems such as [54] dissatisfaction with current project/product performance need to reduce cost and schedule complexity of managing and maintaining too many product variants, and need to quickly respond

to customer and marketplace demands. A key component enabling the effective resolution of these problems is the use of a product line architecture that allows an organization to identify and reuse software artifacts for the efficient creation of products sharing some commonality, but varying in known and managed ways.

Software product families have achieved a broad recognition in the software industry. Many organizations either have adopted or are considering adopting the technology. The key artifacts in software product families are the development, evolution and use of a product family architecture and a set of shared components. Being able to develop a software artifact once and use it in several products or systems is, obviously, one of the main benefits to be achieved.[41]

The software product line strategy for producing software intensive products has produced very promising results for early adopters of the approach. Hewlett-Packard, for example, experienced a twenty fivefold decrease in defects using a product line approach. Cummins, Inc., the world's largest manufacturer of large diesel engines, reduced the effort needed to produce the software for a new engine from 250 person months to three person-months or less.

The product line strategy is widely used in hard goods manufacturing but has only recently been a major influence on software product development processes. A product line approach seeks to achieve gains in productivity and time to market by designing a set of products to have many parts in common. So this is, in a sense, yet another software reuse scheme, but it is one that has proven effective in actual industrial experience. The product line approach also seeks to identify and manage the variations among the products.

The success of the software product line strategy is due, at least partially, to its comprehensive nature. The software product line strategy defines specific tasks for the organizational management, technical management, and software engineering aspects of product production. However, its comprehensive nature also means that the effort to initiate a software product line can be more than that required to adopt a new programming language or change the design method being used.

The comprehensive nature of the product line strategy makes it an umbrella under

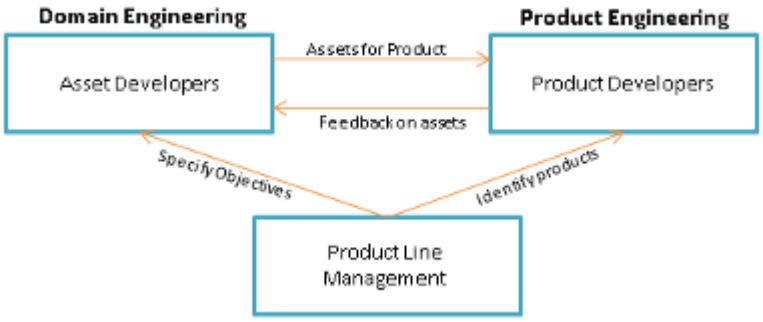

Figure 6.1: The Roles and Interactions[52].

which a range of techniques and methods can be assembled. Agile development methods, model driven architectures, and generative programming can all be part of a successful product line organization.

The product line definition identifies the main roles in a product line organization. Core asset developers provide a range of assets, such as architectures, specifications, and implementations, to product developers for their use in producing products. Product line managers coordinate and facilitate the work of these two groups as illustrated in Figure 6.1. Executives in the organization set strategic goals such as producing more products more quickly and allocate responsibility for achieving those goals.

The organization adopting the product line approach develops a business case that defines objectives, such as increasing productivity, for the product line. The organization identifies the set of products to be included in the product line using scoping techniques that determine the areas of commonality among the products and the points at which the products vary from one another. The products to be produced in the product line are selected so that the objectives of the product line are achieved. If the goal is improved productivity, products might be chosen so that variations among the products are minimized and reuse of components is maximized.

Using the information from the scoping activity and considering the objectives defined in the business case, the organization develops a product line architecture. This architecture incorporates sufficient variation to encompass all of the products in the product line. The architecture serves as the basic guide for specifying and acquiring the other resources that will be used to create the products.

The core asset developers provide the resources needed to produce the selected prod-

ucts. This includes the architecture, the system components that populate the architecture, plans such as production plans and test plans, and templates for process definitions. At points of variation among the products, multiple assets are designed and implemented to cover the possible product permutations.

The core assets of a product line can be more completely specified than traditional reusable components. This is possible because they are designed to work for the specific products in the product line. The assets can be produced for less cost than a similar asset intended for general use in an unspecified environment.

The product developers select the appropriate assets and use these to produce the products identified during product line scoping. Products are assembled quickly and efficiently due to all of the planning and design done by the core asset developers. The product developers may add product specific features that are not shared by other products and hence are not created using core assets. Product line organizations have used a variety of techniques ranging from standard component integration techniques to program generators to produce products from the assets[90].

6.1.1 Variability Notion in Software Product Lines

The success of the software product line strategy is due to its comprehensive nature as well as effective variability management[42]. Variability can be modeled in all phases of product family development addressing traceability and automation issues ranging from requirements to implementation. Different modeling techniques focus different parts of development processes, for instance expressing requirement variability in terms of features; feature modeling with commonality and variability of product lines/families. Moreover, variability modeling and traceability supports evolution in which several evolution categories are presented[127, 128] :

- New product line
- Introduction of new product
- Adding new feature
- Extend standards support

- New version of infrastructure
- Improvement of quality attribute

For effective management issues and reusability of variability models, some models support the separation of concerns idea in which variability is separated from domain knowledge, defined and related with domain artifacts. As stated in [129] in the context of identifying, constraining, implementing and managing of variability, main parts and issues are features, types of features, variability points, types of variability points, variants, realization techniques, software entities, components and frameworks.

Variability points can be introduced in various levels of abstraction in development of product lines; architecture description, detailed design documentation, source code, compiled code, linked code, and running code[98]. Each variability point can be in one of the following states at each variability level stated in [72]; implicit, designed and bound. When a variability point is introduced to a feature model, it is denoted as implicit. When its design is decided in the architectural design phase, it becomes designed. After the variability point is finally bound to a particular variant, it is bound. Binding, when a variability point is bound to a variant, can occur at product architecture, derivation time, compilation time, linking time, start-up time and runtime. A variability point can be either open or closed. If new variants can be added to a variation point, then it is open. On the other hand, if there is no way to add new variants, then it is closed.

6.2 Software Product Lines and Variability of SOA

A software product line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way[52]. Software product lines capitalize on the commonalities and bounded variabilities among similar products. Therefore, variability management is the key issue enabling product line architecture incorporate sufficient variation to encompass all of the products in the product line. In the level of assets, variation points, variants and relation with each other are defined and managed. In the level of artifacts, supported

variabilities are provided to be configured by assets. Asset and related artifact variability should be mapped and compatible with each other for consistency. By this way, artifacts can provide desired behavior stated in the asset. Therefore, variability consistency check can be done at the asset level.

One way to specify variation is expressing requirement variability in terms of features; feature modeling with commonality and variability of product lines. Besides, this mechanism can be used to support evolution as stated in [127, 128]. Along with features specified in the problem domain, a variability model indicating solution domain variability points, constraints and dependencies between them, related variants, variant bindings and realizations are defined. Variability points can be introduced in various levels of abstraction in development of SPL: Architecture description, detailed design documentation, source code, compiled code, linked code, and running code. Here, we focused on architecture description level of abstraction revealing behavioral model where feature model, variability model and architectural views are depicted in Figure 6.2.

6.2.1 Choreography/Orchestration Relation with Asset/Artifacts

Assets and artifacts are reusable parts shaped with the analysis of domain feature model and reference architecture. Assets are not usually executable, but descriptive abstractions including a collection of artifacts plus variabilities. They are used in order to:

- Provide conceptual modularization for understandability,
- Manage variability in one place,
- Define composibility rules between artifacts, and
- Define and encapsulate context information which comes from reference architecture.

Choreography in SOA space deals with other choreographies, orchestrations and atomic services. Likewise, asset in SPL gathers interacting artifacts together while

managing their variability as in this global view. The choreography model in asset describes a collaboration/composition between a collection of artifacts in order to achieve a common goal. Assets and their choreography definitions form the required behavior of product observed from a global viewpoint. There is no restriction on how artifacts are composed to achieve a common goal. Therefore, orchestration and choreography concepts can be used to realize artifact composition. In our scope, asset is not a process which is strongly related with orchestration in the SOA context. Instead, assets deal with composition in choreography point of view as depicted in Figure 6.2.

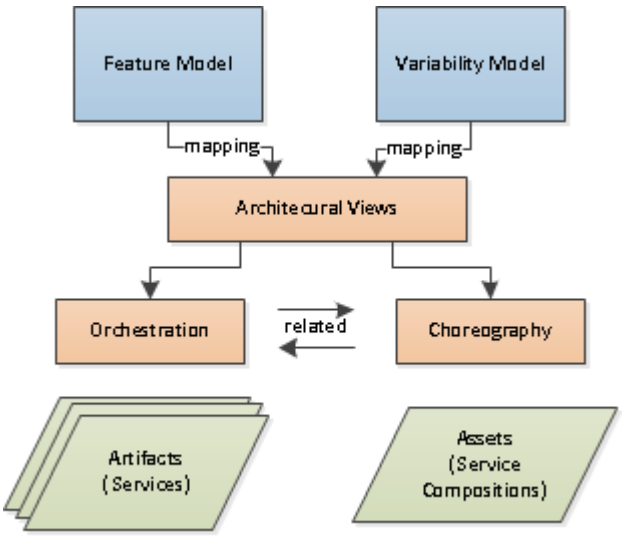

Figure 6.2: SPL and SOA Concept Relations.

Artifacts can be seen orchestrations or atomic services, whereas assets can be realized with choreography revealing service compositions. Composition context, dependency on usage of other artifacts, and composition rules over artifact types are strongly related with choreography definition of an asset. Asset Model includes:

- Artifacts and their dependencies
- Variability points and their dependencies
- Public artifacts (provided)
- External artifacts (required)
- Context

- Constraints
- Choreography

Choreography defines the behavioral part of the asset by specifying artifact composition. At first sight, choreography can be seen as a flow of artifacts or can be modeled by orchestration concepts. In fact, specifications of orchestration and choreography resemble each other but they are different in their composition approaches; first one requires a central orchestrator, whereas the second one is a description of how participants work together without a control mechanism. In this case, an asset includes more than one process in it without a central mechanism, meaning choreography. In this setting, to manage variability at the asset level, there needs to be a choreography model and a language with variability support.

In Reusable Asset Specification (RAS)[19], assets are categorized as executable and non-executable. For executable assets, usage part describes how to compose which can be mapped to choreography in our case. The approach uses MDA approach without including feature mapping. Moreover, RAS asset includes activity for artifact composition, as a simple workflow. Moreover, RAS asset includes activity in order to compose artifacts, however activity comprises a simple workflow.

6.2.2 Component and Service Interfaces

Within component-oriented and service-oriented approaches, at least three kinds of interface categories supported by a component can be identified:

- Provided interfaces
- Required interfaces
- Configuration interfaces

Services provided by a component are exposed through provided interfaces. Required interfaces instead represent services that the component expects from other components. Components are connected through connectors that wire each compatible pair

of provided and required interfaces together. Configuration interface is specific to product lines. It provides a point of access for a product developer to configure the component instance according to product specific requirements. A component configuration interface is associated with each variability point. So basically, the configuration interfaces make variability of the component explicit for the developer.[40] At the architectural level we have three entities that can be made to vary: components, their relationships and connectors as variants. In this context, component identification and related interfaces can be mapped to assets. An asset can have a configuration interface enabling variability management which can be seen by the developers. Here, mainly asset variability is taken into consideration, however connector variability can also be handled.

6.3 Managing Variability with XChor in Software Product Lines

This section elaborates how XChor models can be used in SPL in order to manage variability after analyzing XChor and SPL concepts and depicting XChor under Software Product Line Engineering Framework introduced in [107].

6.3.1 Relation of Software Product Line and XChor Concepts

Software Product Line (SPL) engineering comprises two interrelated processes: domain and application engineering in which requirements engineering, design, implementation and testing phases are applied for each. Within this context, XChor approach targets domain and application design, a part of implementation and testing phases for service-orientation. XChor covers a part of implementation, yet choreography specifications are not executable artifacts. In fact, services can be realized with use of intended technologies by using provided XChor service interfaces. In domain engineering process, to represent domain design and a part of implementation, XChor models with their variability are provided. XChor models are verified according to specified variability via Featured Transition Systems explained in Chapter 4. In application engineering process, application design and a part of implementation are revealed with the analysis and configuration of XChor models via variability binding.

Application testing can be achieved by Promela and SNIP model checker.

In order to cover and satisfy all phases in SPL:

- Domain requirements engineering phase should be related with XChor variability model. For instance if a feature model is provided for variability representation, then proper mappings between feature model and variability model should be defined.
- Domain implementation phase should include a consistently associated runtime environment for XChor service models. A transformation should be defined from XChor models to runtime environment data model.

XChor choreography models, including interface, configuration interface and specification, satisfy the asset model in SPL and XChor services are handled as artifacts which can be an orchestration or an atomic service. An abstract derivation process is depicted in Figure 6.3.

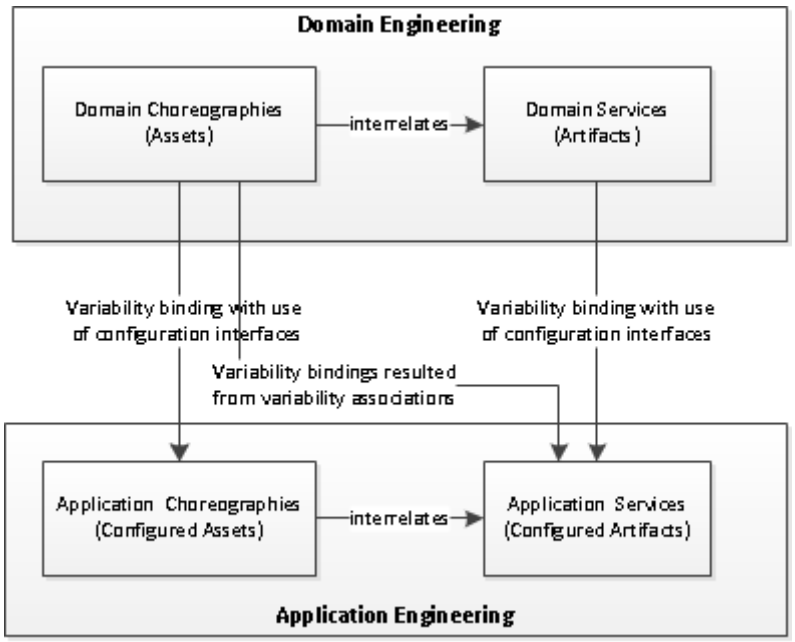

Figure 6.3: SPL and SOA Concept Relations.

In order to derive possible choreographies from domain design, XChor choreography includes variability mechanism. Variability of choreography can be categorized in two; (i) Variability resulting from SPL requirements, and feature model and (ii)

Variability resulting from inner variability structure of the architecture.

Variability resulting from SPL requirements: Functional requirements represent domain capabilities which can be realized by domain artifacts and assets. Variability of these functional requirements can be represented by different models, one of which is feature modeling, a prominent one. While a feature model is related with the XChor model variability, functional requirements are associated with a set of choreography functionalities.

Variability resulting from inner variability structure of the architecture: Abstraction of inner variability details of variation point bindings requires high level variation point descriptions, in our case configuration variation points. For configuration purposes a configurable variation point specifies proper bindings to a set of variation points which are referenced from choreography specifications. Apart from that, choreography can establish variability associations between services in order to form a consistent interaction.

6.3.2 XChor in Software Product Line Framework

The Software Product Line Engineering Framework introduced in [107] has two main processes; domain and application engineering. Domain engineering process is composed of five key sub-processes dealing with domain artifacts; product management, domain requirements engineering, domain design, domain realization, and domain testing. Application engineering comprises four sub-processes dealing with application artifacts; namely application requirements engineering, application design, application realization, and application testing.

XChor targets variability in space while focusing on service interactions as choreographies and services within the SOA context. Within this framework, XChor facilitates following abilities:

- to define domain artifacts via choreography specification and interfaces of service and choreography,
- to represent domain variability model via configuration interfaces,

- to reveal domain architecture via service interactions within choreography specifications,
- to form domain realization artifacts from detailed design view via all XChor models.

6.4 Application of our approach to Axiomatic Design for Component Orientation

Within component oriented software development approaches, Axiomatic Design for Component Orientation (ADCO)[132] is a way to design systems based on divide-and-conquer and find and integrate techniques.

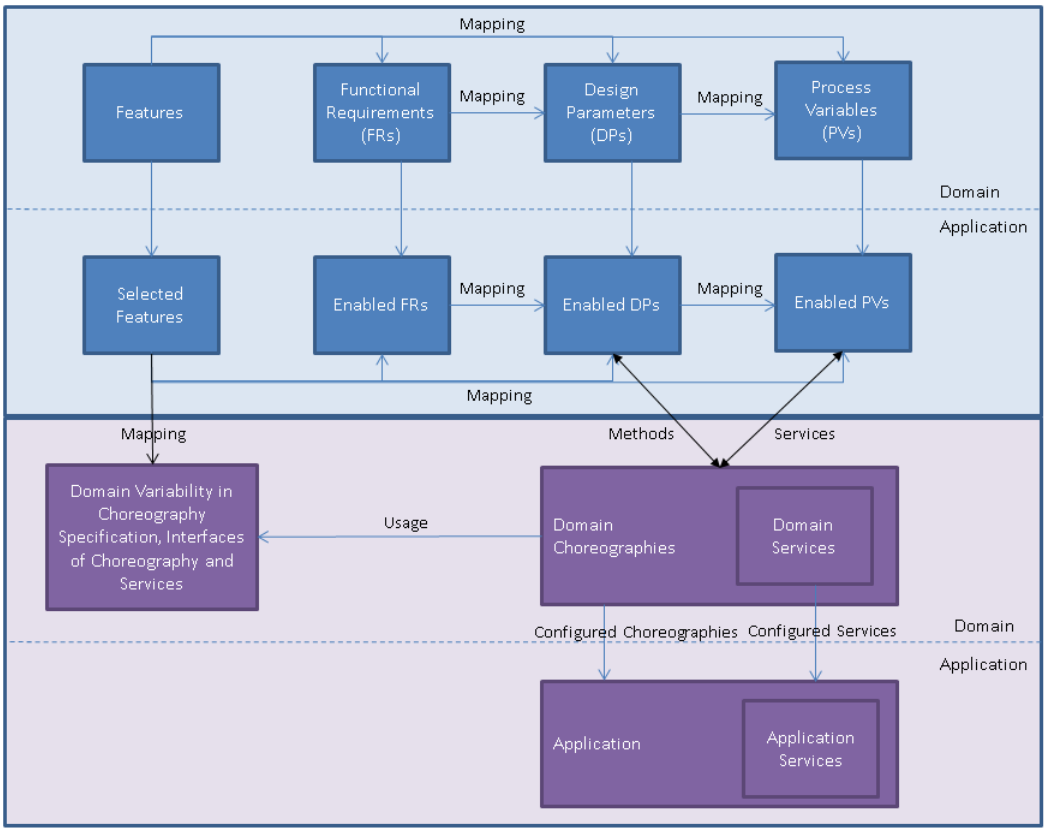

Figure 6.4: Axiomatic Design for Component Orientation (ADCO) Approach with XChor[123].

ADCO brings together Axiomatic Design (AD) and Component Orientation (CO) which supports service-oriented development. The approach utilizes Feature Model[76]

and Axiomatic Design Theory[121] to identify requirements and components that satisfies requirements in mature domains including all functional requirements (FRs), design parameters (DPs) and process variables (PVs). The alteration process of mature domains includes creation of new services or alteration of existing ones in which service maintenance appears as a challenge. In case of variability in service composition specifying and managing variability explicitly cannot be achieved easily with collaboration diagrams or even if a feature model is integrated with FRs.

An application of the usage of XChor within SPL is explained in [123] in which a step by step method is proposed to enrich ADCO approach with XChor Language to fulfill the need of developing reusable service-oriented systems. The integration of ADCO and XChor is represented in Figure 6.4 by connecting into two phases, respectively feature model and DP-PV mapping. Two main approaches of handling ADCO with XChor in domain engineering are defined; fully automatic and evolving. In fully automatic approach, all mappings have been completed beforehand in order to reveal intended applications after feature selection without developer intervention. In evolving approach, a step by step explanation is provided to establish a mature domain with the help of ADCO and XChor. Mapping of feature model to XChor variability model has preconditions: (i) Feature model should be syntactically correct, (ii) Feature model must represent at least one product, (iii) Feature model has at least one variability in that this variability differentiates the derivated products.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This chapter summarizes this thesis study, reveals contributions, and faced challenges and states future work.

7.1 Summary

Several organizations develop business processes for the use within and across organization boundaries. Being agile and flexible requires easily changeable processes where reusable service interactions play a key role. The challenge comes from reuse in composition and coordination of interacting services. For this purpose, many approaches have been proposed to tackle complexity and change via variability management mechanisms, middleware and reconfiguration solutions, dynamic adaptations, and rule-based approaches. Within these approaches, by using variability management mechanisms, reusing existing services and service architecture in an efficient and systematic way is a difficult task.

This study proposes a metamodel and its realization XChor Language. They both are based on reusing existing architecture via explicit variability definition and management in SOA, systematically modeling commonalities and variations across a similar set of service-oriented applications provided by an organization. In this direction, a comparison framework is introduced with three main components; (i)variability modeling, (ii) composition and configuration of models and (iii) tool support. Existing variability models, orchestration and choreography languages are compared based on this comparison framework. Based on this comparison results, shortcomings of ex-

isting approaches, needs and requirements of systematically managing variability in choreography level in SOA are revealed. The challenges of designation of variability needs within service-oriented context lie in determination of

- the types of variation points and variants,
- associations between variation points and variants,
- the parts where and how variability associations are stated,
- the effect of variation points on shared elements of choreography,
- the parts where and how choreography variability in composition is stated or referenced from outside,
- the relationships of service and choreography capabilities with variability and where they are specified.

This study follows choreography approach in order to integrate variable orchestration and atomic services consistently from the global point of view. Taking into account challenges of variability scattered throughout the architecture in order to make variable service-oriented system development feasible, the metamodel of XChor comprises variability model, choreography model and their mapping.

Variability model of XChor enables to specify external and internal variation points with mandatory, optional and alternative variants. Configuration Variation Points are the structures where high level variation points can be mapped to low level variation points in order to increase understandability and decrease complexity. Logical and numerical constraints are defined among variation points and variants.

Choreography model of XChor facilitates to define service interactions as choreography specifications with variability support in composition. Following separation of concerns paradigm, variability of choreography and services is specified outside their capability definition. In other words, all possible capabilities are specified in choreography and service interfaces without variability. With the use of XChor variability model, choreography, orchestration and atomic services can define their own interface variability in their configuration interfaces where existing function and related

parameters can be altered. By this way, choreography can associate its own variability bindings to that of interacting service's either specifying a one-to-one mapping or altering service interface functions or parameters by enabling and disabling. In this model, type, role and participant type concepts, with which role based specifications and multiple service behavior of the same time are specified, are not covered.

Mapping of Choreography and Variability Models creates an alteration mechanism for service and choreography interfaces via variation points. The values and existence of shared variables utilized in choreography specifications and system variables are determined by variation point bindings with use of Parameter Settings part in configuration interfaces of choreography. The variability referencing mechanism from choreography specification enables to define the variable parts of service interactions.

7.2 Contributions

Main contribution lies on the ability of metamodel and XChor Language to specify variability in interface (service and choreography), variability in composition and variability associations. Variability constructs are treated as first class entities and can be defined in all levels of SOA, namely choreography, orchestration, and atomic services. This explicit variation specification and establishment of their associations enable to develop consistent service-oriented compositions. By this way, our study addressed consistent variability bindings and configurations scattered over interacting orchestrations and atomic services.

Specification. The metamodel and XChor Language facilitates to define variation points, variants, and constraints the explicitly treating them as first class entities. The metamodel differentiates between external and internal variation points, and enriches variation point concept by providing a high level configuration facility, namely configuration variation point specification. By the variation point mechanism, services and choreographies can alter their own interfaces internally or externally. The important part of the metamodel addresses variation point associations, mappings in choreography specifications which facilitate to interrelate variation point bindings of interacting services.

However, misuse of variation point associations can lead to inconsistent configurations, because of intervention to the same part of the service interface from configuration interfaces of choreography and the service interface. In any case, inconsistencies are captured during analysis of XChor models. In choreography and service level, optional and alternative variation point types are not supported by XChor. In other words, all specified variation points are treated as mandatory. For creation and specification of XChor models, Eclipse environment with Xtext Domain Specific Language Framework is used. In order to create XChor models, developers should download Eclipse Environment for Xtext from [22], create a new project under a directory, include *XChor.xtext* metamodel specification in Xtext. A developer manual is provided indicating usage details of the metamodel and XChor Language.

Analysis. Analysis of XChor models, such as consistency checking can be done after their specification. To this end, XChorS tool is implemented for analyzing specified variabilities and revealing missing variations, inconsistencies where all warnings and errors are shown to developers.

Configuration. XChor model configuration enables to bind proper variation point bindings, to set parameters, to configure service and choreography interfaces and to form choreography specifications with regard to variation point bindings. XChor model configuration can be available only for development time.

Verification XChor models are transformed to Featured Transition models in order to be formally verified. FTS enables to check system behavior regardless of interface variability. Basic transformed models enables to be checked whether the system has deadlocks or not. For further verification, additional assertions can be inserted by developers to analyze and verify system behavior.

Transformation. Transformation from XChor models to BPEL4Chor, VxBPEL and BPEL represents the applicability of XChor models in existing environments, even if XChor models can not fully be mapped because of lack of variability support in BPEL4Chor and difference between VxBPEL and XChor variability models. Moreover, after transformation, generated models can be verified with different mechanisms; BPEL4Chor language models can be verified via

BPEL2oWFN[87] approach and VxBPEL variable orchestrations via the approach introduced in [113].

Executability. XChor is a non-executable modeling approach, therefore there is no executable environment developed. The models can be mapped to BPEL orchestrations for executability with a limited coverage. Because in case of variability, there should be a management mechanism to handle linking time, start time and runtime variability.

As a summary, main contribution of this thesis stands for defining, modifying, and managing variability scattered over atomic service, orchestration and choreography, as well as configuring all XChor models in a consistent way. As a result of our contributions, we improve development of variable service-oriented systems reducing its complexity level while providing consistent configuration and behavior among choreography, orchestration and atomic services with regard to variability binding.

7.3 Evaluation

The metamodel and XChor language brings a new approach of specification and management of variability in all granularity levels with a single model which provides a consistent collaboration among namely choreography, orchestration and atomic service. The approach reveals the variability needs of interacting services and how they can be configured consistently. The metamodel and language become prominent with specification of relations between choreography and variability models. Among them, variability association mechanism is introduced with the use of explicit behavior of variability. By this way, choreography variability and specification of required proper bindings of interacting services are managed at an abstract, choreography level which eases understanding of choreography goals. Revealing variability logic explicitly through specification of variation points and variability associations establishes a structure that is easy to manage and understand by non-technical and technical business process developers. Hidden variation logic is removed from service interfaces, compositions and configuration logic. While interface and composition variability are supported, connector variability is left as a future work.

The metamodel and language supports interaction model by defining choreography specifications and interconnection model by enabling services to define their abstract business processes. Because the main focus is managing variability integrated with all granularity levels, it is not claimed that fully mapping from current languages is achieved. For instance, BPEL statements in Participant Behavior Descriptions model of BPEL4Chor can not be fully transformed to XChor because XChor does not support event handling and a full fault handling mechanism. All required tools are provided to developers including (i) specification and constructing a variable service compositions, (ii) analysis and configuration of variable XChor models, (iii) verification of variable XChor models, and (iv) transformation to closer specifications; namely BPEL4Chor and VxBPEL.

7.4 Future Work

While the study represents a significant improvement in variability management in choreography model for developing consistent service-oriented systems and providing reuse, it also opens a number of further long term or short term research areas. Metamodel and its realization XChor Language have been studied, constructs, analysis, configuration, verification tools have been provided in this study. In order to form a complete approach from head to toe, the following long term research areas can be addressed:

Runtime environment There are different approaches to handle the runtime variability management ranging from agent-oriented decision models, rule based systems to dynamic linking. A runtime environment should be provided so as to fully support variability management in all binding times.

Configuration A mechanism should be provided in order to support derivation time, compile time, linking, startup and runtime configurations so as to fully support processing in all variation binding times.

Variability in Connectors The metamodel and XChor language tools are available as open source. The metamodel and accordingly language should be extended

to fulfill variability in connectors, for instance via mediations, artificial intelligence structures, decision models.

Transformation from XChor to FTS model for verification In XChor functions and function parameters of service and choreography interfaces can be changed by configuration interfaces and variability associations defined within XChor choreographies. These interface changes lead to existence or non-existence of functions and parameters which are referenced within choreography specifications. In other words, due to service interface changes, one of the service function can not be provided by a service. Therefore, the service interaction including this function can not be achieved in choreography behavior. These interface consistency checking can not be directly addressed by FTS. Therefore, an additional mechanism is applied to fulfill this need.

Transformation from BPEL4Chor and VxBPEL Although BPEL4Chor does not support variability specification, transformation from a set of BPEL4Chor specifications to XChor models eases choreography specification process. Then developers can create configuration interfaces with a set of variability and manually fill choreography specifications with variability associations, context elements and variability attachments, the structures where a set of variation points are referenced. Due to non-existence of multiple choreography support (more than one interacting choreographies) in BPEL4Chor, an additional mechanism is needed to differentiate and gather possible choreographies. Participant behavior descriptions is partially created from *abstract process definition* parts of services which needs human intervention in some case. Therefore, the transformation does not cover all semantics which are left as a future work. In transformation from VxBPEL case, the differences between variability models requires additional information of variability types specified by developers. Moreover, one choreography specification can be generated from one VxBPEL variable orchestration. In case of multiple VxBPEL orchestration specification, a new high level choreography specification is created indicating all VXBPEL orchestration interactions.

Short term issues that can be addressed are as follows:

- Developer and user manuals indicating installation and usage guides with an example project should be provided.
- XChorS GUI which eases development and configuration process, guides developers and users ,increases understanding of XChor models with XChor application developer and user view options should be provided. For instance, whereas developer can see external and internal variation points and internal structure of the choreography specification, the model user can only observe external variability and behavior of the models.

REFERENCES

[1] Organization for the advancement of structured information standards, oasis. http://www.oasis-open.org/home/index.php, last visited on November 2013.

[2] Standard generalized markup language,sgml. (ISO 8879:1986, http://www.iso.org/iso/catalogue_detail?csnumber=16387, 1986, last visited on November 2013.

[3] World wide web. http://en.wikipedia.org/wiki/Www, 1989, last visited on November 2013.

[4] W3c. http://www.w3.org/, 1994, last visited on November 2013.

[5] extensible markup language specification, xml. http://www.w3.org/TR/WD-xml-961114, http://www.w3.org/XML/hist2002, 1996, last visited on November 2013.

[6] Indigo project. http://msdn.microsoft.com/en-us/magazine/cc164026.aspx, 1999, last visited on November 2013.

[7] Web service concept. http://msdn.microsoft.com/en-us/library/ms954826.aspx, 1999, last visited on November 2013.

[8] Electronic business using extensible markup language, ebxml. http://www.ebxml.org/, 2000, last visited on November 2013.

[9] Universal description, descovery and integration specification, uddi. http://uddi.xml.org/, 2000, last visited on November 2013.

[10] Web service description language specification, wsdl. http://www.w3.org/TR/wsdl, 2000, last visited on November 2013.

[11] Web services flow language specification, wsfl. http://xml.coverpages.org/wsfl.html, 2000, last visited on November 2013.

[12] Working draft of wsdl pointing need for choreography. http://www.w3.org/2005/12/wscwg-charter.html, 2000, last visited on November 2013.

[13] Xlang specification. http://www.ebpml.org/xlang.htm, 2000, last visited on November 2013.

[14] Reo coordination language, reo. http://reo.project.cwi.nl, 2001, last visited on November 2013.

[15] Web service conversation language specification, wscl. http://www.w3.org/TR/wscl10/, 2002, last visited on November 2013.

[16] Orc coordination lanugage, orc. http://orc.csres.utexas.edu/index.shtml, 2004, last visited on November 2013.

[17] Service oriented architecturei soa. http://msdn.microsoft.com/en-us/magazine/cc164026.aspx, 2004, last visited on November 2013.

[18] W3c web services glossary. http://www.w3.org/TR/ws-gloss/, 2004, last visited on November 2013.

[19] Ras ,reusable asset specification 2.2. http://www.omg.org/spec/RAS/, 2005, last visited on November 2013.

[20] Jolie language specification. http://www.jolie-lang.org/, 2006, last visited on November 2013.

[21] Web services coordination specification, ws-coordination. http://docs.oasis-open.org/ws-tx/wscoor/2006/06, 2006, last visited on November 2013.

[22] Xtext 2.3.1. http://www.eclipse.org/Xtext/, 2012, last visited on November 2013.

[23] J. van Gurp A. G. J. Jansen, R. Smedinga and J. Bosch. First class feature abstractions for product derivation. *Software, IEE Proceedings*, 151(4):187–197, January 20048.

[24] Wil M. Aalst, Arjan J. Mooij, Christian Stahl, and Karsten Wolf. Formal methods for web services. chapter Service Interaction: Patterns, Formalization, and Analysis, pages 42–88. Springer-Verlag, Berlin, Heidelberg, 2009.

[25] M. Abu-Matar. Toward a service-oriented analysis and design methodology for software product lines. http://www.ibm.com/developerworks/webservices/library/ar-soaspl/index.html, 2007.

[26] Phillipa Oaks Alistair Barros, Marlon Dumas. A critical overview of the web services choreography description language (ws-cdl). BPTrends Newsletter, www.bptrends.com, 2005.

[27] Timo Asikainen, Tomi Männistö, and Timo Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Adv. Eng. Inform.*, 21(1):23–40, January 2007.

[28] Timo Asikainen, Timo Soininen, and Tomi Männistö. A koala-based approach for modelling and deploying configurable software product families. In FrankJ. Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 225–249. Springer Berlin Heidelberg, 2004.

[29] Darren C. Atkinson, Daniel C. Weeks, and John Noll. The design of evolutionary process modeling languages. In *APSEC*, pages 73–82. IEEE Computer Society, 2004.

[30] Felix Bachmann, Michael Goedicke, Julio Leite, Robert Nord, Klaus Pohl, Balasubramaniam Ramesh, and Alexander Vilbig. A meta-model for representing variability in product family development. In FrankJ. Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin Heidelberg, 2004.

[31] M.R. Barbacci, N.A. Habermann, and M. Shaw. *The Software Engineering Institute: Bridging Practice and Potential*. 1985.

[32] Adam Barker, Christopher D. Walton, and David Robertson. Choreographing web services. *IEEE Trans. Serv. Comput.*, 2(2):152–166, April 2009.

[33] Alistair Barros, Marlon Dumas, and ArthurH.M. Hofstede. Service interaction patterns. In WilM.P. Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 302–318. Springer Berlin Heidelberg, 2005.

[34] George Baryannis, Olha Danylevych, Dimka Karastoyanova, Kyriakos Kritikos, Philipp Leitner, Florian Rosenberg, and Branimir Wetzstein. Service research challenges and solutions for the future internet. pages 55–84, Berlin, Heidelberg, 2010. Springer-Verlag.

[35] Bernhard Bauer and JörgP. Müller. Mda applied: From sequence diagrams to web service choreography. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *Web Engineering*, volume 3140 of *Lecture Notes in Computer Science*, pages 132–136. Springer Berlin Heidelberg, 2004.

[36] Martin Becker. Towards a general model of variability in product families. In *Proceedings of the First Workshop on Software Variability Management*, 2003.

[37] A. K. Bhattacharjee and R. K. Shyamasundar. Scriptorc: A specification language for web service choreography. In *APSCC*, pages 1089–1096. IEEE, 2008.

[38] Barry Boehm. Managing software productivity and reuse. volume 32, pages 111–113, Los Alamitos, CA, USA, September 1999. IEEE Computer Society Press.

[39] Nicola Boffoli, Marta Cimitile, Maria Maggi Fabrizio, and Giuseppe Visaggio. Managing soa system variation through business process lines and process oriented development. In *Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL)*, pages 61–68. Springer Berlin Heidelberg, 2009.

[40] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[41] Jan Bosch. Staged adoption of software product families. *Software Process: Improvement and Practice*, 10(2):125–142, 2005.

[42] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 13–21, London, UK, UK, 2002. Springer-Verlag.

[43] Gary Brown. Pi calculus for soa. http://sourceforge.net/projects/pi4soa/, last visited on November 2013.

[44] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Towards a formal framework for choreography. In *WETICE*, pages 107–112. IEEE Computer Society, 2005.

[45] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in eflow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, CAiSE '00, pages 13–31, London, UK, UK, 2000. Springer-Verlag.

[46] Fabio Casati, Ski Ilnicki, Li-Jie Jin, and Ming-Chien Shan. An open, flexible, and configurable system for service composition. In *Proceedings of the Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, WECWIS '00, pages 125–, Washington, DC, USA, 2000. IEEE Computer Society.

[47] Soo Ho Chang and Soo Dong Kim. A variability modeling method for adaptable services in service-oriented computing. In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 261–268, Washington, DC, USA, 2007. IEEE Computer Society.

[48] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 81–90, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[49] Henry William Chesbrough. *Open Innovation: The new imperative for creating and profiting from technology*. Harvard Business Review Press, 2003.

[50] Andreas Classen. *Modelling and Model Checking Variability-Intensive Systems*. PhD thesis, PReCISE Research Centre, Faculty of Computer Science, University of Namur (FUNDP), 5000 Namur, Belgium, October 2011.

[51] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Sci. Comput. Program.*, 76(12):1130–1143, December 2011.

[52] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2002.

[53] Thomas Cottenier and Tzilla Elrad. Engineering distributed service compositions. In *Proceedings of the First International Workshop on Engineering Service Compositions*, WECS'05, pages 51–58, Amsterdam, The Netherlands, 2005. IEEE Computer Society.

[54] Gary Chastek Dave Zubrow. Measures for software product lines, software engineering measurement and analysis initiative. Technical Report TN-031, CMU/SEI, 2005. Technical Note.

[55] Gero Decker, Margarit Kirov, Johannes Maria Zaha, and Marlon Dumas. Maestro for let's dance: An environment for modeling service interactions. In *Demonstration Session of the 4th International Conference on Business Process Management (BPM)*, 2006.

[56] Gero Decker, Oliver Kopp, Frank Leymann, Kerstin Pfitzner, and Mathias Weske. Modeling service choreographies using bpmn and bpel4chor. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 79–93, Berlin, Heidelberg, 2008. Springer-Verlag.

[57] Arie van Deursen, Merijn de Jonge, and Tobias Kuipers. Feature-based product line instantiation using source-level packages. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 217–234, London, UK, UK, 2002. Springer-Verlag.

[58] Business Modeling & Integration (BMI) Domain Task Force (DTF). Business process markup language specification, bpml. http://www.ebpml.org/bpml.htm, 2002, last visited on November 2013.

[59] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30, August 2005.

[60] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Printice Hall, 2005.

[61] Paul C. Clements Felix Bachmann. Variability in software product lines. Technical Report TR-012, ESC-TR-2005-012, CMU/SEI, 2005.

[62] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[63] Roy Thomas Fielding. *Representational State Transfer, REST*. Phd thesis, University of California, Irvine, http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm, 2000.

[64] Dario Fischbein, Sebastian Uchitel, and Victor Braberman. A foundation for behavioural conformance in software product line architectures. In *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, ROSATEA '06, pages 39–48, New York, NY, USA, 2006. ACM.

[65] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan. Service-oriented adaptation in ubiquitous computing environments. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 02*, CSE '09, pages 458–463, Washington, DC, USA, 2009. IEEE Computer Society.

[66] Kurt Geihs, Roland Reichle, Michael Wagner, and MohammadUllah Khan. Modeling of context-aware self-adaptive applications in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 146–163. Springer Berlin Heidelberg, 2009.

[67] David Gelernter and Arthur J. Bernstein. Distributed communication via global buffer. pages 10–18, 1982.

[68] Roberto Gorrieri, Claudio Guidi, and Roberto Lucchi. Reasoning about interaction patterns in choreography. In *Proceedings of the 2005 international conference on European Performance Engineering, and Web Services and Formal Methods, international conference on Formal Techniques for Computer Systems and Business Processes*, EPEW'05/WS-FM'05, pages 333–348, Berlin, Heidelberg, 2005. Springer-Verlag.

[69] Short Keith with Cook Steve Greenfield Jack and Kent Stuart. *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*. Wiley Publishing, 2004.

[70] Lombard Hill Group. What is software reuse? http://www.lombardhill.com/, last visited on November 2013.

[71] Alexander Gruler, Martin Leucker, and Kathrin Scheidemann. Modeling and model checking software product lines. In *Proceedings of the 10th IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS '08, pages 113–131, Berlin, Heidelberg, 2008. Springer-Verlag.

[72] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 45–, Washington, DC, USA, 2001. IEEE Computer Society.

[73] IAAS. Open source bpel4chor tools. https://github.com/IAAS, last visited on November 2013.

[74] Thales Tat Consultancy Service IBM, Franhoufer FOKUS. Common variability language (cvl). http://www.omgwiki.org/variability/doku.php, last visited on November 2013.

[75] Patrick Donohoe Lawrence G. Jones John K. Bergey, Sholom Cohen. Software product lines: Experiences from the eighth dod software product line workshop. Technical Report TR-023 ESC-TR-2005-023, CMU/SEI, 2005.

[76] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[77] Lars Åke Fredlund. Implementing ws-cdl. In *Proceedings of JSWEB 2006 (II Jornadas Científico-Técnicas en Servicios Web)*, BPM'05, Santiago de Compostela, Spain, 2006.

[78] Sedigheh Khoshnevis. An approach to variability management in service-oriented product lines. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1483–1486, Piscataway, NJ, USA, 2012. IEEE Press.

[79] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[80] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Inf. Softw. Technol.*, 51(2):258–269, February 2009.

[81] Oliver Kopp, Lasse Engler, Tammo Lessen, Frank Leymann, and Jörg Nitzsche. Interaction choreography models in bpel: Choreographies on the enterprise service bus. In Albert Fleischmann, Werner Schmidt, Robert Singer, and Detlef Seese, editors, *Subject-Oriented Business Process Management*, volume 138 of *Communications in Computer and Information Science*, pages 36–53. Springer Berlin Heidelberg, 2011.

[82] Charles W. Krueger. Variation management for software production lines. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 37–48, London, UK, UK, 2002. Springer-Verlag.

[83] Peter Kim Krzysztof Czarnecki, Chang Hwan. Cardinality-based feature modeling and constraints: A progress report. OOPSLA'05, 2005.

[84] Ulrich W. Eisenecker Krzysztof Czarnecki, Simon Helsen. Staged configuration using feature models. In *SPLC*, pages 266–283, 2004.

[85] Ivan Lanese, Antonio Bucchiarone, and Fabrizio Montesi. A framework for rule-based dynamic adaptation. In *Proceedings of the 5th international conference on Trustworthly global computing*, TGC'10, pages 284–300, Berlin, Heidelberg, 2010. Springer-Verlag.

[86] Gang Liu, Shengqi Lu, and Ronghua Chen. The role-oriented process modeling language. In *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*, pages 1–5, 2013.

[87] Niels Lohmann, Oliver Kopp, Frank Leymann, and Wolfgang Reisig. Analyzing BPEL4Chor: Verification and Participant Synthesis. In *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia*, pages 46–60. Springer-Verlag, September 2007.

[88] Ayesha Manzer. *Formalization of Core-competency Process for Integration of Value-add Chains*. Phd thesis, Middle East technical University, Ankara, Turkey, 2002.

[89] Deasy Kevin M. Martin Anne C. The effect of software support needs on the department of defense software acquisition policy: Part 1 a framework for analyzing legal issues. Technical Report 87-TR-2, CMU/SEI, 1987.

[90] John D. McGregor. Software product lines. *Journal of Object Technology*, 3(3):65–74, 2004.

[91] David William Mennie. An architecture to support dynamic composition of service components and its applicability to internet security. In *5 th International Workshop on Component-Oriented Programming – WCOP 2000 at the 14th European Conference on Object-Oriented Programming - ECOOP 2000*, 2000.

[92] Microsoft. Component object model / distributed component object model, com/ dcom. http://www.microsoft.com/com/default.mspx, 1996, last visited on November 2013.

[93] Bardia Mohabbati, Marek Hatala, Dragan Gašević, Mohsen Asadi, and Marko Bošković. Development and configuration of service-oriented systems families. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1606–1613, New York, NY, USA, 2011. ACM.

[94] Fabrizio Montesi, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Jolie: a java orchestration language interpreter engine. *Electron. Notes Theor. Comput. Sci.*, 181:19–33, June 2007.

[95] Thomas Motal, Marco Zapletal, and Hannes Werthner. The business choreography language (bcl) - a domain-specific language for global choreographies. In *SERVICES II*, pages 150–159. IEEE Computer Society, 2009.

[96] Nataliya A. Mulyar. *Patterns for Process-Aware Information Systems: An Approach Based on Colored Petri Nets*. Phd thesis, Technische Universiteit Eindhoven, 2009.

[97] Dirk Muthig and Colin Atkinson. Model-driven product line architectures. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 110–129, London, UK, UK, 2002. Springer-Verlag.

[98] Tommi Myllymäki. Variability management in software product-lines. Technical Report 30, Institute of Software Systems, Tampere University of Technology, 2002.

[99] Nanjangud C. Narendra and Karthikeyan Ponnalagu. Towards a variability model for soa-based solutions. *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 0:562–569, 2010.

[100] Tuan Nguyen, Alan Colman, Muhammad Adeel Talib, and Jun Han. Managing service variability: state of the art and open issues. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 165–173, New York, NY, USA, 2011. ACM.

[101] Alex Norta. A choreography language for ebusiness collaboration. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 468–469, New York, NY, USA, 2011. ACM.

[102] OASIS. Web services business process execution language specification, ws-bpel. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel, 2007, last visited on November 2013.

[103] OMG. Common object request broker architecture specification, corba. http://www.corba.org/, 1991, last visited on November 2013.

[104] OMG. Business process model and notation-bpmn 2.0 specification. http://www.omg.org/spec/BPMN/2.0/, 2011, last visited on November 2013.

[105] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, November 2007.

[106] Joonseok Park, Mikyeong Moon, and Keunhyuk Yeom. Variability modeling to develop flexible service-oriented applications. *Journal of Systems Science and Systems Engineering*, 20(2):193–216, 2011.

[107] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[108] Maryam Razavian and Ramtin Khosravi. Modeling variability in the component and connector view of architecture using uml. In *Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '08, pages 801–809, Washington, DC, USA, 2008. IEEE Computer Society.

[109] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Appl. Ontol.*, 1(1):77–106, January 2005.

[110] Davide Rossi and Elisa Turrini. Epml: an executable process modeling language for process-aware applications. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 132–133, New York, NY, USA, 2008. ACM.

[111] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 164–182. Springer Berlin Heidelberg, 2009.

[112] Serge Salicki and Nicolas Farcet. Expression and usage of the variability in the software product lines. In Frank Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 304–318. Springer Berlin Heidelberg, 2002.

[113] Mustafa Yucefaydalı Selma Süloğlu, Riza Aktunc. Verification of variable service orchestrations using model checking. In *Proceedings of 3rd International Symposium on Business Modeling and Software Design*. ACM, 2013.

[114] Antonio Ruiz-Cortes Pablo Trinidad Sergio Segura, David Benavides. An approach to variability management in service-oriented product lines. In *In first Workshop on Service-oriented Architectures and Product Lines*. SEI, 2007.

[115] Robert Wolfe-Steve Olding Shahin Samadi, Nadine Alameh and David Isaac. Strategies for enabling software reuse within the earth science community. In *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, volume 3. IEEE International, 2004.

[116] B.G. Silverman. Software cost and productivity improvements: An analogical view. *Computer*, 18(5):86–96, 1985.

[117] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Inf. Softw. Technol.*, 49(7):717–739, July 2007.

[118] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In RobertL. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer Berlin Heidelberg, 2004.

[119] Ian Sommerville. *Software Engineering*. Addison-Wesley, 2010.

[120] Zoran Stojanovic and Ajantha Dahanayake. *Service-oriented Software System Engineering Challenges And Practices*. IGI Publishing, Hershey, PA, USA, 2005.

[121] Nam Pyo Suh. *Axiomatic Design: Advances and Applications*. Oxford University Press, USA, 2001.

[122] Selma Suloglu. Xchor language representation in xtext. http://www.xchor.com/XChorLanguage-xtext.pdf, 2012, last visited on November 2013.

[123] Selma Suloglu, Cengiz Togay, and Ali H. Dogru. Managing variability in service composition with axiomatic design. In *Proceedings of 18th International Conference on Society for Design and Process Science, SDPS 2013*. SDPS, 2013.

[124] BEA Sun, SAP and Intalio. Web services choreography interface specification, wsci. http://www.w3.org/TR/wsci/, 2002, last visited on November 2013.

[125] Chang-ai Sun, Tieheng Xue, and Marco Aiello. Valysec: A variability analysis tool for service compositions using vxbpel. In *Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference*, APSCC '10, pages 307–314, Washington, DC, USA, 2010. IEEE Computer Society.

[126] Hongyu Sun, Robyn R. Lutz, and Samik Basu. Product-line-based requirements customization for web service compositions. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 141–150, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[127] Mikael Svahnberg and Jan Bosch. Characterizing evolution in product line architectures. In *Proceedings of the 3rd annual IASTED, International Conference on Software Engineering and Applications*, pages 92–97.

[128] Mikael Svahnberg and Jan Bosch. Evolution in software product lines: Two cases. *Journal of Software Maintenance*, 11(6):391–422, November 1999.

[129] Mikael Svahnberg and Bosch Jan van Gurp, Jilles. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, July 2005.

[130] Rajesh K. Thiagarajan, Amit K. Srivastava, Ashis K. Pujari, and Visweswar K. Bulusu. Bpml: A process modeling language for dynamic business models. In *WECWIS*, pages 239–241.

[131] Steffen Thiel and Andreas Hein. Systematic integration of variability into product line architecture design. In GaryJ. Chastek, editor, *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 130–153. Springer Berlin Heidelberg, 2002.

[132] Cengiz Togay, Ali H. Dogru, and John Urcun Tanik. Systematic component-oriented development with axiomatic design. *Journal of Systems and Software*, 81(11):1803–1815, 2008.

[133] N.Yasemin Topaloglu and Rafael Capilla. Modeling the variability of web services from a pattern point of view. In Liang-Jie(LJ) Zhang and Mario Jeckle, editors, *Web Services*, volume 3250 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin Heidelberg, 2004.

[134] André van der Hoek. Design-time product line architectures for any-time variability. *Sci. Comput. Program.*, 53(3):285–304, December 2004.

[135] W3C. Simple object access protocol, soap. http://www.w3.org/TR/soap/, 1998, last visited on November 2013.

[136] W3C. Text markup language specification, html. http://www.w3.org/html/, 2004, last visited on November 2013.

[137] World Wide Web W3C. Web services choreography description language specification, ws-cdl. http://www.w3.org/2005/12/wscwg-charter.html, 2005, last visited on November 2013.

[138] Yong Wang, Xiang Yi, Kai Li, and Meilin Liu. An actor-based language to unifying web service orchestration and web service choreography. In *Computer Science and Information Processing (CSIP), 2012 International Conference on*, pages 1055–1060, 2012.

[139] Hanpin Wang Yu Huang. A petri net semantics for web service choreography. In *SAC*, pages 1689–1690, 2007.

[140] J. M. Zaha, M. Dumas, A. H.M. ter Hofstede, A. Barros, and G. Decker. Bridging global and local models of service-oriented systems. *Trans. Sys. Man Cyber Part C*, 38(3):302–318, May 2008.

[141] Johannes Maria Zaha, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Let's dance: a language for service behavior modeling. In *Proceedings of the 2006 Confederated international conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE - Volume Part*

I, ODBASE'06/OTM'06, pages 145–162, Berlin, Heidelberg, 2006. Springer-Verlag.

[142] Yongwang Zhao, Dianfu Ma, Min Liu, and Chunyang Hu. Coordination behavioral structure: A web services coordination model in dynamic environment. In *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, ICIS '08, pages 611–617, Washington, DC, USA, 2008. IEEE Computer Society.

APPENDIX A

XCHOR METAMODEL REALIZATION IN XTEXT

```
1  grammar oorg.xtext.doctorate.xchor.Xchor with org.eclipse.xtext.
       common.Terminals
2
3  generate xchor "http://www.xtext.oorg/doctorate/xchor/Xchor"
4
5
6  /**
7    * Variable Choreography Model is a set of Abstract Elements
         specified to define
8    * (i) choreography specifications with variabilities,
9    * (ii) service and choreography interfaces and
10   * (iii) their configuration interfaces.
11   * Root of the grammar
12   *
13   */
14 VarChorModel:
15    (elements += AbstractElement)*
16 ;
17 /**
18  * An ID assigned to variation points whether they reside in
        composition or
19  * take part in configuring service interfaces via mapping or
20  *
21  * Reside in composition: @composition
22  * Take part in configuring service interfaces: @vconfservice
23  * Take part in configuration variation point realization:
        @vconfrealization
```

```
24   */
25  Tag:
26    "@" (name = "composition" | name ="vconfservice" | name = "
          vconfrealization")
27  ;
28
29  /**
30   * Assignment of the return value of a service (Interface) function
          (Function) to a
31   * ContextElement which is a shared variable of choreography
32   *
33   */
34  ChorComputation:
35    "%comp" name = [ContextElement] "=" s = [Interface] "." f = [
          Function] "%"
36  ;
37
38  /**
39   * Abstract Element definition which can be either
40   * a choreography definition (Choreography),
41   * configuration of a service/choreography (VarConfigurationModel)
        or
42   * interface of service/choreography (Interface).
43   * Here service concept covers atomic and orchestrated services
44   */
45  AbstractElement:
46    Choreography | VarConfigurationModel | Interface
47  ;
48
49  /**
50   * Choreography definition which
51   * (i) imports its configuration interface (via VConfModelImport),
52   * interacting choreographies (via ChorImport) and services (via
          ServiceImport),
53   * (ii) defines shared variables (via Context Elements),
54   * (iii) maps choreography variability with other service s and
          choreography s
```

```
55   * variation point and variant specifications (via VMMapping)
56   * (iv) defines the choreography composition for each function with
         inline variation
57   * attachments as a guard to execute the piece of choreography (via
         Composition)
58   *
59   */
60  Choreography:
61    "choreography" name=ID
62      (vconfmodelimport = VConfModelImport)?
63      (cimports  += ChorImport)*
64      (simports += ServiceImport)+
65      ("Context Elements" (contexts += ContextElement)*)?
66      ("Choreography Variability Mapping" (mappings += VMMapping)*)?
67      ("Function" func += [Function] ":" comp += Composition)+
68  ;
69
70
71  /**
72   * Import mechanism to include choreography s configuration
         Interface
73   * There can be more than one configuration Interface of the same
         choreography.
74   *
75   */
76  VConfModelImport:
77    "import configuration" importedNamespace = [
         VarConfigurationModel4Chor]
78  ;
79
80  /**
81   * Import mechanism to include services utilized (ServiceInterface)
         in choreography
82   * composition with specified service configuration interface (
         VarConfigurationModel4Service)
83   * if required.
```

```
84   * There can be more than one configuration Interface of the same
         service.
85   *
86   */
87  ServiceImport:
88    "import service" s = [ServiceInterface] ( "with configuration"
         importedNamespace = [VarConfigurationModel4Service])?
89  ;
90
91  /**
92   * Import mechanism to include other choreographies (ChorInterface)
         interacted with the current choreography
93   *
94   */
95  ChorImport:
96    "use choreography" name = [ChorInterface]
97  ;
98
99  /**
100  * An abstraction of two types of interface: Choreography and
         Service
101  *
102  */
103 Interface:
104   ChorInterface | ServiceInterface
105 ;
106
107 /**
108  * A definition of interface for a choreography, the opened face to
         other choreographies
109  * incuding invariants (Invariant), externalized functions (
         Function), port description (Port) and
110  * required interfaces from other choreographies (RequiredInterface
         ).
111  *
112  */
113 ChorInterface:
```

```
114   "Choreography interface" name = QualifiedName "of " chorname = ID
115   (invariants += Invariant)*
116   ((functions += Function)+)
117   port += Port
118   ("required interfaces"
119   (reqints += RequiredInterface )*)?
120 ;
121
122 /**
123  * A definition of demanded functions from other choreographies
        seperated by ;
124  *
125  */
126 RequiredInterface:
127   "from" name = QualifiedName "function" "{" ( f += [Function]) ("," 
        (f += [Function]))*  "}"
128 ;
129
130 /**
131  * A definition of a service interface including invariants (
        Invariant),
132  * functions (Function) and port specification (Port).
133  *
134  */
135 ServiceInterface:
136   "Service interface" name = ID
137   ((invariants += Invariant)*)
138   ((functions += Function)+)
139   port += Port
140 ;
141
142 /**
143  * Definition of a function including its name, pre and post
        conditions (ConditionSet),
144  * input parameters (Params) and
145  * its output (Param).
146  *
```

```
147  */
148  Function:
149    "function" name = ID
150    ("precondition" precond += ConditionSet)?
151    ("postcondition" postcond += ConditionSet)?
152    ("input" ipars = Params)?
153    ("output" opar = Param)?
154  ;
155
156  /**
157   * A set of conditions (Condition) composed via "or" and "and"
          logical relationships.
158   *
159   */
160  ConditionSet:
161    "(" c1 += Condition (("or" | "and") c2 += Condition )* ")"
162  ;
163
164  /**
165   * A definition of the value of an object is equal or not equal to
          a booelan value
166   * (true or false).
167   *
168   */
169  Condition:
170    name = ID ("==" | "!=") BOOLEAN
171  ;
172
173  /**
174   * A variable definition in choreography/service interface assigned
          to
175   * a boolean value which is valid throughout the choreography/
          service composition
176   */
177  Invariant:
178    "invariant" name = ID "==" BOOLEAN
179  ;
```

```
180
181 /**
182  * An abstraction of three types of variation point:
183  * internal, external and configuration variation point.
184  *
185  */
186 VarPoint:
187   ConfigurationVarPoint | InternalVarPoint | ExternalVarPoint
188 ;
189
190 /**
191  * A variation point definition which is invisible to outer context
        so as to
192  * describe a variability with a set of variants (VariantSet) and
       specified binding time (BINDING).
193  *
194  */
195 InternalVarPoint:
196   vt = "internalVP" name=ID  :  variants = VariantSet  "bindingTime
       " btime =BINDING
197 ;
198
199 /**
200  * A variation point definition which is visible to outer context
       to be configured by other
201  * services/choreographies. It specifies its variability with a set
       of variants (VariantSet) and
202  * specified binding time (BINDING).
203  * Note that "externalVP" should be used while defining a variation
       point for choreograph in
204  * configuration interface and "vp" should be used for service
       variation in configuration interface
205  *
206  */
207 ExternalVarPoint:
208   ( vt = "externalVP" | vt2 = "vp")  name=ID  :  variants =
       VariantSet "bindingTime" btime =BINDING
```

```
209 ;
210
211
212 /**
213  * An abstract high level variation point definition that maps its
        variants to
214  * a set of internal variation points with their variant selections
        , specifying each realization.
215  * It can be either internal or external which is specified by "
        vartype" keyword.
216  * It defines a set of variants (VariantSet) and their realization
        (ConfVariantWithChoices),
217  * default variant (Variant) selection and binding time (BINDING).
218  *
219  */
220 ConfigurationVarPoint returns ConfigurationVarPoint:
221   "configuration" ({InternalVarPoint} name=QualifiedName  :  "
        varType"  vt = "internalVP" | {ExternalVarPoint} name=
        QualifiedName  :  "varType"  vt = "externalVP" )
222   (variants = VariantSet)
223   ("realization" rea = STRING) ((confvariants +=
        ConfVariantWithChoices)+ )
224   ("defaultVariant" defaultVariant = [Variant]) ("type" type=
        CONFTYPE "bindingTime" btime = BINDING )
225 ;
226
227 /**
228  * A variant definition of a configuration variation point
        including  a set of choices.
229  *
230  */
231 ConfVariantWithChoices:
232   "confvariant" name = ID "mapping"
233   (choices += Choice)+
234 ;
235
236 /**
```

```
237  * Selection defition of a variation point among defined ones and
         related selected variants
238  * as well as minimum and/or maximum number of variant selections
         as optional
239  *
240  */
241 Choice:
242   "VPName" vp = [VarPoint] "selectedVariants(" (vars += [Variant])+
          ("; min:" INT)? (", max:" INT)? ")"
243 ;
244
245 /**
246  * A set of variants (Variant) grouped by as mandatory, optional
         and alternative.
247  * Alternative variants are specified with minimum and maximum
         number of selections.
248  *
249  */
250 VariantSet:
251   {VariantSet} ("mandatory" (variants += Variant)* )?
252   ("optional " (variants += Variant)*)?
253   ("alternative " (variants += Variant)* "(min:" INT ",max:"INT ")
          ")?
254 ;
255
256 /**
257  * A variable definition of a variation point which activates
         functions of services (MethodsWithoutDefinedServices)
258  * or its functions stated in the interface (
         MethodsWithoutDefinedServices) and/or
259  * sets a parameter (Function) to a function (Function) stated in
         service interface if required.
260  *
261  */
262 Variant returns Variant :
263   "variant" name = ID  ( (":activateMethods(" ( m1 =
          MethodsWithDefinedServices | m2 = MethodsWithoutDefinedServices
```

```
        ) ")")?
264     (":setParameter(toFunct:" f = [Function] ",parameter:" pars =
            Param (";toFunct:" func += [Function] ",parameter:" fpars +=
             Param)*")" )?
265   )
266 ;
267
268 /**
269  * A set of functions (Function) with related services seperated
        by comma.
270  *
271  */
272 MethodsWithDefinedServices:
273   "service:" s = [ServiceInterface] ",funct:" funct = [Function]
        ("," functs += [Function])* ("; service:" s2 += [
        ServiceInterface] ",funct:" funct2 += [Function] ("," functs2
        += [Function])*)*
274 ;
275
276 /**
277  * A set of functions of its own seperated by comma.
278  *
279  */
280 MethodsWithoutDefinedServices:
281   funct = [Function] ("," functs += [Function])*
282 ;
283
284 /**
285  * Binding definition of other services to current service/
        choreography with a defined host.
286  *
287  */
288 Port:
289   "portName" name = ID "binding" host = TEXT
290 ;
291
292 /**
```

```
293  * An abstraction of two types of configuration model
294  * VarConfigurationModel4Service for service and
295  * VarConfigurationModel4Chor for choreography
296  *
297  */
298 VarConfigurationModel:
299   VarConfigurationModel4Service | VarConfigurationModel4Chor
300 ;
301
302 /**
303  * A definition of a configuration interface for a service
        including
304  * (i) a set of external variation points (ExternalVarPoint) with a
        tag (Tag)
305  * defining the role of it if required,
306  * (ii) constraints (Constraint) among external variation points
        and
307  * (iii) its abstract process definition (Composition) which
        specifies external behavior
308  * of the service with other services.
309  *
310  */
311 VarConfigurationModel4Service:
312   "Configuration interface" name = ID "of service" servicename = [
        ServiceInterface]
313
314   ((tag += Tag)? vars += ExternalVarPoint)*
315
316   ("Constraints"
317   (constraints += Constraint )*)?
318
319   ("abstract process definition"
320     processdef = Composition
321
322   )?
323 ;
324
```

```
/**
 * A definition of a configuration interface for a choreography
    including
 * (i) a set of internal, external and configuration variation
    points (VarPoint)
 * with a tag (Tag) defining the role of them if required,
 * (ii) constraints (Constraint) among variation points and
 * (iii) parameter settings (ParameterSetting) which includes a set
     of defined
 * parameters used in choreography.
 *
 */
VarConfigurationModel4Chor:
  "Configuration interface" name = ID  "of choreography" chorname =
      QualifiedName

  ((tag += Tag)?  vars += VarPoint)*

  ("Constraints"
  (constraints += Constraint )*)?

  ("Parameter Settings"
  (parametersetting += ParameterSetting )*)?
;

/**
 * An assignment of a value to ContextElements resided in
    choreography specification.
 *
 */
ParameterSetting:
  "parameter" name = [ContextElement] ("= #ofVariantsSelected{" (
      vars += [Variant])+ "} Of " vp = [VarPoint] |
  "= value(" var += [Variant] ("," vars += [Variant])* ")" | "
      existswhenselected{" vp=[VarPoint]"."v =[Variant]
  ("," vp2+=[VarPoint]"."v2 +=[Variant])* "}")
;
```

```
355
356
357 /**
358  * An abstraction of two types of constraints:
359  * LogicalConstraint and NumericalConstraint
360  *
361  */
362 Constraint:
363   LogicalConstraint | NumericalConstraint
364 ;
365
366 /**
367  * A definition depicting a constraining relationship in which a
        variation
368  * point and/or related variants decide another variation points
        and/or its selected variants
369  * status either excluded, implied, required or negated.
370  *
371  */
372 LogicalConstraint:
373   ( p1 = [VarPoint] (p2 = [Variant])? ) c =CONST  p3 = [VarPoint]
        ("selectedVariants("(vars += [Variant])+ (", min:" INT)? (",
        max:" INT)? ")")?
374 ;
375
376 /**
377  * A definition depicting a constraining relationship in which a
        variation
378  * point and related variant result in an assignment of a value to
        another variation point
379  * and related variant or to a property with expressions (greater
        than, less than, greater than or equal,
380  * less than or equal, equal, not equal)
381  *
382  */
383 NumericalConstraint:
384   vp1 = [VarPoint] v1 = [Variant] nconst = NUMCONST
```

```
385     rhs = RHS exp = EXPR
386     (STRING | pro2 = Property | "valueOf{" (vars += [Variant])* "}")
387   ;
388
389   RHS:
390     pro1=Property | (vp2 = [VarPoint] v2 = [Variant])
391   ;
392   /**
393    * A specification of a system property with its name
394    *
395    */
396   Property:
397     name = ID
398   ;
399
400   /**
401    * A shared element definition used in choreography composition
402    *
403    */
404   ContextElement:
405     name = QualifiedName (defaultvalue = INT | STRING | ID | BOOLEAN)
406   ;
407
408   /**
409    * An abstraction of two types of varaibility mapping
410    * VMServiceMapping for service and
411    * VMChorMapping for choreography
412    */
413   VMMapping:
414       VMServiceMapping | VMChorMapping
415   ;
416
417   /**
418    * A structural mapping from choreography variation to service
          variation.
419    * First variation points are mapped and then each variant of
          related choreography
```

```
420  * variation point is mapped to that of service variation point.
421  *
422  */
423 VMServiceMapping:
424   "VP" vp = [VarPoint] "maps service" service = [ServiceInterface]
          "VP" svp = [VarPoint]
425   ( "Variant"  vars += [Variant] "maps Variant" (mvars += [Variant]
          )+)+
426 ;
427
428 /**
429  * A structural mapping from choreography variation to utilized
        choreography variation.
430  * First variation points are mapped and then each variant of
        realted choreography variation
431  * point is mapped to that of utilized choreography variation point
        .
432  *
433  */
434 VMChorMapping:
435   "VP" vp = [VarPoint] "maps choreography" chor = [Choreography] "
          VP" cvp = [VarPoint]
436   ( "Variant"  vars += [Variant] "maps Variant" (mvars += [Variant]
          )+)+
437 ;
438 /**
439  * A definition of an attachment to choreography composition in
        order to define the conditions of
440  * variation point and variant selections.
441  * Relationships between variation point and variants used are:
442  * "ifOneSelected" if one of the variants in a variant set is
        selected
443  * "ifAllSelected" if all of the variants in a variant set is
        selected
444  * "ifSelected" if some of the variants in a variant set is
        selected
445  * "excl:"is used when a set of variants needed not to be selected.
```

```
446  * The composition segment tagged with VariabilityAttachment is
         added to the composition if
447  * the selections are realized.
448  */
449 VariabilityAttachment:
450   "#vp" vp += [VarPoint] ("ifOneSelected(" | "ifAllSelected(" | "
         ifSelected(") (vs += [Variant])+ (";excl:" (vsexc += [Variant
         ])+)? ")"
451         (("and" | "or") vp2 += [VarPoint] ("ifOneSelected(" | "
             ifAllSelected("| "ifSelected(") (vs2 += [Variant])+ (";
             excl:" (vsexc2 += [Variant])+)? ")" )* "#"
452 ;
453
454 /**
455  * A definition of a set of interactions in order to realize a
         common goal via one or more atomic
456  * (AtomicInteractions) and/or composite (CompositeInteraction)
         interactions tangled with each other.
457  *
458  */
459 Composition:
460   (interactions += (AtomicInteraction | CompositeInteraction) (WS
         interactions += ( AtomicInteraction | CompositeInteraction))* )
         +
461 ;
462
463 /**
464  * A definition of an interaction between services with/without a
         guard (IntCondition) including
465  * a set of selection of an interaction among others (SelectInt),
466  * repeating a set of interactions (RepeatInt),
467  * paralellization of a set of interactions (ParalelInt)
468  * and flowing down in a sequence (SequenceInt).
469  *
470  */
471 CompositeInteraction:
```

```
472   ("guard(" guard = IntConditionSet ")" )? ("precedent")? (
          interaction = SelectInt | interaction = RepeatInt | interaction
           = ParalelInt | interaction = SequenceInt)
473   ("timeout" INT)?
474 ;
475
476 /**
477  * A definition of a selection between a set of interactions among
          services which can be atomic (AtomicInteraction) or
478  * composite (CompositeInteraction) with/without variability
          attachment (VariabilityAttachment).
479  * SelectInt is written in such a way that the block is started
          with "select", interactions
480  * are surrounded with paranthesis.
481  *
482  */
483 SelectInt:
484   (va = VariabilityAttachment)? "select" (cond = IntConditionSet)?
          " (" interactions += (AtomicInteraction | CompositeInteraction)
          + ")"
485 ;
486
487 /**
488  * A definition of a repeatition of a set of interactions between
          services which can be atomic (AtomicInteraction) or
489  * composite (CompositeInteraction) with an exit condition and with
          /without variability attachment (VariabilityAttachment).
490  * RepeatInt is written in such a way that the block is started
          with "repeat" following a condition and
491  * a set of interactions are surrounded with paranthesis.
492  *
493  */
494 RepeatInt:
495   (va = VariabilityAttachment)? "repeat" cond = IntConditionSet
          "("(interactions += (AtomicInteraction | CompositeInteraction))
          + ")"
496 ;
```

```
497
498 /**
499  * A definition of a paralelization of a set of interactions
        between services which can be atomic (AtomicInteraction) or
500  * composite (CompositeInteraction) with/without variability
        attachment (VariabilityAttachment).
501  * ParalelInt is written in such a way that the block is started
        with "paralel", interactions
502  * are surrounded with paranthesis.
503  *
504  */
505 ParalelInt:
506   (va = VariabilityAttachment)? "parallel (" (interactions += (
        AtomicInteraction | CompositeInteraction))+ ")"
507 ;
508
509 /**
510  * A definition of a sequence of a set of interactions between
        services which can be atomic (AtomicInteraction) or
511  * composite (CompositeInteraction) with/without variability
        attachment (VariabilityAttachment).
512  * SequenceInt is written in such a way that the block is started
        with "sequence", interactions
513  * are surrounded with paranthesis.
514  *
515  */
516 SequenceInt:
517   (va = VariabilityAttachment)? "sequence (" (interactions += (
        AtomicInteraction | CompositeInteraction))+ ")"
518 ;
519
520 /**
521  * A specification of a basic interaction between two services with
        /without variability attachment (VariabilityAttachment).
522  * AtomicInteraction is written
523  * with/without a guard condition (IntCondition),
```

```
524  * depiction of source and destination services (Interface) with "
         send" or "receive" actions,
525  * a message (Message),
526  * with/without a computation effect to a ContextElement (
         ChorComputation) and
527  * with other additional constructs.
528  * If the action is "receive" from a set of services and one should
          be selected then "pickOne" is added. (for racing incoming
         messages pattern)
529  * If more than one receive is accomplished from a source to a
         destination, then "multiple times" should be added.
530  * If the "send" action requires notification from destination,
         then "withNotification" is added.
531  * If an atomic action is limited with a duration, then "wait"
         keyword with a time specification should be provided.
532  * When an atomic action wants to explicitly depict a fault when a
         problem is occurred, a "fault" should be defined.
533  * If interaction is "send" willing to get a request from one of
         avaliable destionations with a limited duration, then
534  * "callingSequence" (for contingent requests pattern) is defined
         with a sequence of destinations.
535  * If the interaction causes an interaction; sending the value of
         the computation to another service/services,
536  * then "referedDestinations" is defined.
537  * If the AtomicInteraction causes one or more changes in
         ContextElement s values, then a set of ChorComputation is
         defined.
538  *
539  */
540 AtomicInteraction:
541   (va = VariabilityAttachment)?
542   ("guard(" guard = IntConditionSet")")?
543   (source = [Interface] type = "send"  "{" (destionation += [
          Interface])+ "} " ("in sequence")? ("atomic")? ("viewer")? |
544    destination = [Interface] type = "receive"  ("from{" (rsource +=
          [Interface])* "} ")? ("multiple times")? ("pickOne")? )
545   (message += Message)
```

```
546   ("stopmessage from" stopservice = [Interface] )?
547   ("wait" (t = Time)? ("until" INT "messagescame")?)?
548   ("inactivity interval" inact = Time)?
549   ("referedDestinations(" refpart += [Interface] ((",," refpart += [
          Interface])*)? ")" )?
550   ("withNotification" ("(min:" min = INT ",max:" max = INT ")" )? )
          ?
551   (f += Faults ("toreferrals")?)?
552   (comp +=ChorComputation)*
553 ;
554
555 /**
556  * A definition of message including set of parameters (Param),
          semantical description,
557  * refering service (Interface) and its function (Function).
558  *
559  */
560 Message:
561   "message" name = [Function] ("(" (par += [Param] ("," par += [
          Param] )*)? ")")
562   ("refers"  (service += [Interface] "." funct += [Function])* )?
          //?     *
563   ("semantic (" s = STRING ")")?
564 ;
565
566 /**
567  * A set of Interaction Conditions (IntCondition).
568  *
569  */
570 IntConditionSet:
571   icond += IntCondition (("or" | "and") icond += IntCondition )*
572 ;
573
574 /**
575  * A specification of a condition used to guard a part of an
          interaction.
```

```
576  * IntCondition can be either a definition of a condition with
         expression and numerical/non numerical values or
577  * a specification of number.
578  *
579  */
580 IntCondition:
581   p1 = GUARDTEXT ((exp = EXPR (STRING | INT | ID | BOOLEAN) ) | "times
         ")?
582 ;
583
584 /**
585  * A set of parameters separated by comma and surrounded with
         parenthesis
586  *
587  */
588 Params:
589   pars = "(" p1 = Param ("," p2 += Param)* ")"
590 ;
591
592 /**
593  * A parameter definition with its name
594  *
595  */
596 Param:
597   name = ID
598 ;
599
600 /**
601  * A system failure with its name and explanation and sends fault
         notification to corresponding senders
602  *
603  */
604 Faults:
605   "fault(" fname1 = FAULTTYPES ("," fname2 += FAULTTYPES)* (",
         terminateIf" number = INT "fails")? ")"
606 ;
607
```

```
608 /**
609  * A specification of how names of some elements in choreography
        should be defined.
610  *
611  */
612 QualifiedName:
613   ID ( _ ID)*;
614
615 /**
616  * The amount of duration with related units
617  *
618  */
619 Time:
620   name = INT ("second" |"seconds" | "hour" | "hours" | "day"| "days"| "
        month"| "months" )
621 ;
622 /**
623  * The rest of the grammar rules define constant values used in
        other rules.
624  *
625  */
626 GUARDTEXT:
627   ID | INT
628 ;
629
630 TEXT:
631   (ID | INT | ":" | "/")+
632 ;
633
634 CONST:
635   requires = requires | excludes = excludes | implies = implies
        | negates = negates
636 ;
637
638 enum NUMCONST:
639   const = "const"
640 ;
```

```
641 BOOLEAN:
642   "true" | "false"
643 ;
644
645 enum EXPR:
646   gt = " " |gte = " ="|lt = " "| lte = " =" | equ = "=="| neq = "!=" |
          eq = "="
647 ;
648
649 enum CONFTYPE:
650   subs = "substitution" | para ="parameterization" | add ="addition"
651 ;
652
653 BINDING:
654   devt = "devtime"| derv = "derivation" | comp = "compilation" | link
          = "linking" |strt= "start up" | runt ="runtime"
655 ;
656
657 FAULTTYPES:
658   "delivery" | "parameter" | "notready" | "waittimeout" | "
          insufficientmessage" | "notavaliable"
659 ;
```

APPENDIX B

TRAVEL ITINERARY SYSTEM IN XCHOR LANGUAGE

```
1  choreography travelitinerary
2
3    import configuration vconf_travelitinerary
4    import service airline
5    import service hotel
6    import service travelagency with configuration vconf_travelagency
7    import service traveler with configuration vconf_traveler
8    import service carrental
9    import service cruise
10   import service activityprovider
11
12   Context Elements
13     flighttticketconfirmation false
14     hotelbookingconfirmation false
15     cruiseconfirmation false
16     carrentalconfirmation false
17     activityconfirmation false
18         intelem 0
19         stringelem "str"
20
21   Choreography Variability Mapping
22     VP booking maps service travelagency VP plan
23       Variant airline maps Variant withairline
24       Variant hotel maps Variant withhotel
25     VP facilities maps service travelagency VP plan
26       Variant activities maps Variant withactivities
27       Variant carrental maps Variant withcarrental
```

```
28      Variant cruise maps Variant withcruise
29     VP facilities maps service traveler VP activityselection
30        Variant activities maps Variant withactivity
31
32  Function planitinerary:
33    sequence (
34      traveler send{travelagency} message querytrip(startdate,
            enddate,details)
35      parallel (
36        #vp booking ifSelected(airline)# travelagency send{airline}
            message requestprice(startdate,enddate)
37        #vp booking ifOneSelected(hotel )# travelagency send{hotel}
            message requestprice(startdate,enddate,details)
38        #vp facilities ifAllSelected(cruise)# travelagency send
            {cruise} message requestprice(packageid,date)
39        #vp facilities ifSelected(carrental)# travelagency send
            {carrental} message requestprice(carmodel,date)
40        #vp facilities ifSelected(activities)# sequence (
41          travelagency send{activityprovider} message listactivities
              (place,date) referedDestinations(traveler)
42          traveler send{travelagency} message setselectedactivities(
              selectedlist)
43          travelagency send{activityprovider} message requestprice(
              selectedlist,date)
44        )
45      )
46            travelagency send {traveler} message
                getavaliabletripoptions(travelerID)
47      traveler receive from{travelagency} message
          getavaliabletripoptions(travelerID)
48      #vp booking ifSelected(airline)# sequence (
49        traveler send{travelagency} message selectairline(airlineID)
50        %comp intelem=travelagency.selectairline%
51      )
52      #vp booking ifSelected(hotel)# sequence (
53        traveler send{travelagency} message selecthotel(hotelID)
54        %comp stringelem=travelagency.selecthotel%
```

```
55        )
56
57        #vp facilities ifSelected(cruise)# sequence (
58          traveler send{travelagency} message selectcruise(packageid,
              date)
59          %comp cruiseconfirmation=travelagency.selectcruise%
60        )
61
62        #vp facilities ifSelected(carrental)# sequence (
63          traveler send{travelagency} message selectcarrental(carmodel
              )
64          %comp carrentalconfirmation=travelagency.selectcarrental%
65        )
66
67        #vp facilities ifSelected(activities)# sequence (
68          traveler send{travelagency} message setselectedactivities(
              activitylist)
69          %comp activityconfirmation=travelagency.
              setselectedactivities%
70                travelagency send{traveler} message getconfirmedplan
                    ()
71        )
72
73        select (
74          guard(hotelbookingconfirmation == true and
              flightticketconfirmation == true) parallel (
75            #vp booking ifSelected(airline)# sequence (
76              travelagency send{airline} message bookflight(arrival,
                    departure) wait 3 seconds withNotification
77              travelagency send{airline} message processticket(
                    customerID) referedDestinations(traveler)
78            )
79            #vp booking ifSelected(hotel)# sequence (
80              travelagency send{hotel} message bookroom(arrival,
                    departure,details) wait 3 seconds withNotification
81              travelagency send{hotel} message processvoucher(
                    customerID) referedDestinations(traveler)
```

```
82            )
83
84            traveler receive from{travelagency} message
                  getconfirmedplan()
85          )
86          sequence (
87                      travelagency send{traveler} message
                            gettripplancancelation()
88            travelagency send{traveler} message gettripplancancelation
                  ()
89            traveler receive from{travelagency} message
                  gettripplancancelation()
90          )
91        )
92      )
```

```
1  Configuration interface vconf_travelitinerary of choreography
       travelitinerary
2
3    externalVP booking:
4      optional
5        variant hotel
6        variant airline
7      bindingTime devtime
8
9    externalVP facilities:
10     optional
11       variant cruise:activateMethods(planitinerary):setParameter(
             toFunct:planitinerary,parameter:planned)
12       variant carrental:activateMethods(service:carrental,funct:
             rent)
13       variant activities
14     bindingTime devtime
15
16   configuration itinerary:
17     varType externalVP
18     alternative
```

```
19        variant regular
20        variant vacationpackage
21        (min:1,max:1)
22      realization "it is realized by booking and facilities variation
            points"
23      confvariant regular mapping
24        VPName booking selectedVariants(hotel airline; min:1, max:2)
25        VPName facilities selectedVariants(cruise carrental
              activities; min:0, max:3)
26      confvariant vacationpackage mapping
27        VPName booking selectedVariants(hotel airline)
28        VPName facilities selectedVariants(cruise carrental
              activities; min:0, max:3)
29      defaultVariant regular
30      type parameterization
31      bindingTime devtime
32
33    Parameter Settings
34      parameter hotelbookingconfirmation existswhenselected{booking.
            hotel}
35      parameter flightticketconfirmation existswhenselected{booking.
            airline}
36      parameter cruiseconfirmation existswhenselected{facilities.
            cruise}
37      parameter carrentalconfirmation existswhenselected{facilities.
            carrental}
38      parameter activityconfirmation existswhenselected{facilities.
            activities}
```

```
1  Choreography interface travel of travelitinerary
2
3    function planitinerary
4      input(plan)
5      output tripplan
6
7    portName travel binding localhost:8081
```

```
1  Service interface travelagency
```

```
2
3   function querytrip
4     input(startdate ,enddate ,details)
5     output tripinfo
6
7   function selecthotel
8     input(hotelID)
9
10  function selectairline
11    input(airlineID)
12
13  function sendconfirmedplan
14    input(customerID)
15    output confirmedplan
16
17  function sendtripplancancelation
18    input(customerID)
19
20  function setselectedactivities
21    input(selectedlist)
22
23  function selectactivities
24    input(activitylist)
25
26  function selectcruise
27    input(packageid ,date)
28
29  function selectcarrental
30    input(carmodel)
31
32  portName agency binding localhost:8080
```

```
1 Configuration interface vconf_travelagency of service travelagency
2
3   externalVP plan:
4     optional
```

```
5       variant withhotel:activateMethods(selecthotel,querytrip,
            sendconfirmedplan,sendtripplancancelation)
6       variant withairline:activateMethods(selectairline,querytrip,
            sendconfirmedplan,sendtripplancancelation)
7       variant withcruise:activateMethods(selectcruise,querytrip,
            sendconfirmedplan,sendtripplancancelation)
8       variant withcarrental:activateMethods(selectcarrental,
            querytrip,sendconfirmedplan,sendtripplancancelation)
9       variant withactivities:activateMethods(setselectedactivities,
            selectactivities,querytrip,sendconfirmedplan,
            sendtripplancancelation)
10    bindingTime devtime
11
12  abstract process definition
13    sequence (
14    travelagency receive from{traveler} message querytrip(startdate,
        enddate,details)
15    parallel (
16      #vp plan ifSelected(withairline)# travelagency send{airline}
            message requestprice(startdate,enddate)
17      #vp plan ifSelected(withhotel)# travelagency send{hotel}
            message requestprice(startdate,enddate,details)
18      #vp plan ifSelected(withcruise)# travelagency send{cruise}
            message requestprice(packageid,date)
19      #vp plan ifSelected(withcarrental)# travelagency send
            {carrental} message requestprice(carmodel,date)
20      #vp plan ifSelected(withactivities)# sequence (
21        travelagency send{activityprovider} message listactivities(
              place,date) referedDestinations(traveler)
22        travelagency receive from{traveler} message
              setselectedactivities(selectedlist)
23        travelagency send{activityprovider} message requestprice(
              selectedlist,date)
24      )
25    )
26    travelagency send{traveler} message getavaliabletripoptions(
        travelerID)
```

```
27      #vp plan ifSelected(withairline)# sequence (
28        travelagency receive from{traveler} message selectairline(
              airlineID)
29        %comp flightticketconfirmation=travelagency.selectairline%
30      )
31      #vp plan ifSelected(withhotel)# sequence (
32        travelagency receive from{traveler} message selecthotel(
              hotelID)
33        %comp hotelbookingconfirmation=travelagency.selecthotel%
34      )
35
36      #vp plan ifSelected(withcruise)# sequence (
37        travelagency receive from{traveler} message selectcruise(
              packageid,date)
38        %comp cruiseconfirmation=travelagency.selectcruise%
39      )
40
41      #vp plan ifSelected(withcarrental)# sequence (
42        travelagency receive from{traveler} message selectcarrental(
              carmodel)
43        %comp carrentalconfirmation=travelagency.selectcarrental%
44      )
45
46      #vp plan ifSelected(withactivities)# sequence (
47        travelagency receive from{traveler} message selectactivities(
              activitylist)
48        %comp activityconfirmation=travelagency.setselectedactivities
              %
49      )
50
51      select (
52        guard(hotelbookingconfirmation == "true" and
              flightticketconfirmation == "true") parallel (
53          #vp plan ifSelected(withairline)# sequence (
54            travelagency send{airline} message bookflight(arrival,
                  departure) wait 3 seconds
```

```
55            travelagency send{airline} message processsticket(
                  customerID) referedDestinations(traveler)
56          )
57          #vp plan ifSelected(withhotel)# sequence (
58            travelagency send{hotel} message bookroom(arrival,
                  departure,details) wait 3 seconds
59            travelagency send{hotel} message processvoucher(customerID
                  ) referedDestinations(traveler)
60          )
61          travelagency send{traveler} message sendconfirmedplan(
                customerID)
62        )
63        travelagency send{traveler} message sendtripplancancelation(
            customerID)
64      )
65      )
```

```
1  Service interface traveler
2
3    function getavaliabletripoptions
4      input(travelerID)
5
6    function selectactivities
7      input(activitylist)
8      output selectedlist
9
10   function getconfirmedplan
11     input(confirmedplan)
12
13   function gettripplancancelation
14     input(cancellation)
15
16   portName traveler binding localhost:8082
```

```
1  Configuration interface vconf_traveler of service traveler
2
3    externalVP activityselection:
4      alternative
```

```
5       variant withactivity :activateMethods(getavaliabletripoptions
            ,selectactivities)
6       variant withoutactivity :activateMethods(
            getavaliabletripoptions)
7     (min:1,max:1)
8     bindingTime devtime
9
10  abstract process definition
11    sequence (
12      traveler send{travelagency} message querytrip(startdate,
            enddate,details)
13      traveler receive from{travelagency} message
            getavaliabletripoptions(travelerID)
14      #vp activityselection ifSelected(withactivity)#traveler send
            {travelagency} message selectactivities(activitylist)
15      traveler receive from{travelagency} message getconfirmedplan()
16      traveler receive from{travelagency} message
            gettripplancancelation()
17    )
```

```
1 Service interface hotel
2
3   function requestprice
4     input(startdate,enddate,additionalrequests)
5     output price
6
7   function makereservation
8     input(arrival,departure,additionalrequests)
9     output confirmation
10
11  function bookroom
12    input(arrival,departure,additionalrequests)
13    output confirmation
14
15  function processvoucher
16    input(customerID)
17    output voucher
```

```
18
19    portName hotel binding localhost:8087

1  Service interface airline
2
3     function requestprice
4       input(startdate,enddate)
5       output price
6
7     function bookflight
8       input(arrival,departure,additionalrequests)
9       output confirmation
10
11    function processticket
12      input(customerID)
13      output eticket
14
15    portName airline binding localhost:8084

1  Service interface cruise
2
3     function requestprice
4       input(packageid,date)
5       output price
6
7     function bookcruise
8       input(packageid,date,numbeofperson)
9       output confirmation
10
11    portName cruise binding localhost:8087

1  Service interface carrental
2
3     function requestprice
4       input(carmodel,date)
5       output price
6
7     function rent
```

```
8       input(dateinterval,carmodel)
9       output confirmation
10
11    portName carrental binding localhost:8085
```

```
1   Service interface activityprovider
2
3     function listactivities
4       input(place,date)
5       output activitylist
6
7     function requestprice
8       input(activitylist,date)
9       output price
10
11    function enrollactivity
12      input(activity,date,numbeofperson)
13      output confirmation
14
15    portName activityprovider binding localhost:8089
```

APPENDIX C

ADAPTABLE SECURITY SYSTEM IN XCHOR LANGUAGE

```
1
2  choreography adaptablesecuritysystem
3
4    import configuration vconf_adaptablesecuritysystem
5
6    use choreography chor_alert
7      use choreography chor_credentialmng
8
9    import service connection
10   import service encryption with configuration vconf_encryption
11   import service credentials
12   import service attemptcalc
13   import service comparison with configuration vconf_comparison
14   import service responsewindow
15   import service interfaceprep with configuration vm_interfaceprep
16   import service thirdparty with configuration vm_thirdparty
17   import service user
18   import service warning
19
20   //Shared variables
21   Context Elements
22     //user s wrong attempts
23     wrongattempts 0
24     //fake interface content enabling
25     fakeinterface false
26     //biometric selected authentication type variants specifies the
           number
```

```
27      noofbiometricauthtypeselected 0
28      //defualt parameters for encryption
29      defaultparams "username_passw"
30      //user entered credential data
31      usernamepass ""
32      //extracted features of user biometric data
33      processeddata ""
34
35    Choreography Variability Mapping
36        VP i_encryption_parameters maps service encryption VP
              encryption_params
37        Variant defaultparams maps Variant withdefaultparams
38        Variant setparams maps Variant withparams
39      VP i_transaction_type maps service comparison VP analysis
40        Variant faketransaction maps Variant fake
41        Variant realtransaction maps Variant real
42      VP i_auth_type maps service thirdparty VP user_device
43        Variant username_passw maps Variant ATM Mobile PDA PC
44        Variant onetimepassw maps Variant ATM Mobile PDA PC
45        Variant esign maps Variant ATM Mobile PDA PC
46        Variant fingerprint maps Variant PC
47        Variant fingervein maps Variant PC
48        Variant iris maps Variant PC
49        Variant face maps Variant PC
50      VP i_auth_type maps choreography credentialmng VP devicecon
51        Variant fingerprint maps Variant biometricdevice
52        Variant fingervein maps Variant biometricdevice
53        Variant iris maps Variant biometricdevice
54        Variant face maps Variant biometricdevice
55
56      Function verify:
57      sequence (
58        #vp i_auth_type ifOneSelected( fingerprint fingervein iris
              face) # repeat noofbiometricauthtypeselected times(
59          user send{chor_credentialmng} message getcredentials(
                deviceparameter)
60          %comp processeddata =chor_credentialmng.getcredentials%
```

```
61          chor_credentialmng send{encryption} message setparams(
                parameters)
62        )
63
64        #vp i_auth_mode ifSelected(mode_online)# sequence (
65          thirdparty receive message getconnection()
66          thirdparty send{encryption} message setparams(parameters)
67        )
68        user send{chor_credentialmng} message getcredentials(
              deviceparameter)
69        %comp usernamepass =chor_credentialmng.getcredentials%
70        chor_credentialmng send{encryption} message setparams(
              parameters)
71        encryption receive message encrypt(credentials)
72
73        #vp i_auth_mode ifSelected(mode_online)# sequence (
74            encryption send{thirdparty} message verify(data)
75            #vp i_transaction_type ifSelected(faketransaction)#
                  thirdparty send{comparison} message fakeanalysis(
                  comparisonresult)
76            %comp fakeinterface=comparison.fakeanalysis%
77        )
78        #vp i_auth_mode ifSelected(mode_offline)# sequence (
79            encryption send{storage} message gethasheddata()
                  referedDestinations(comparison)
80            #vp i_transaction_type ifSelected(faketransaction)#
                  storage send{comparison} message fakeanalysis()
81        )
82
83        guard(fakeinterface==false) sequence (
84          comparison send{attemptcalc} message calculate_wrong
                _attempts(result)
85          %comp wrongattempts=attemptcalc.calculate_wrong_attempts%
86          guard(wrongattempts == 3) parallel (
87            comparison send{responsewindow} message show()
88            attemptcalc send{connection} message closeconnection()
89          )
```

```
90          guard(wrongattempts    3) parallel (
91            comparison send{responsewindow} message show()
92            attemptcalc send{warning} message warn(response_warning)
93          )
94        )
95        guard(fakeinterface==true) #vp i_transaction_type ifSelected(
              faketransaction)#parallel (
96          sequence (
97            comparison send{interfaceprep} message prepareinterface()
98            interfaceprep send{responsewindow} message show()
99          )
100         comparison send{chor_alert} message alert()
101       )
102     )
103   Function enroll:
104     sequence (
105       user send{chor_credentialmng} message getcredentials(
              deviceparameter)
106       %comp usernamepass =chor_credentialmng.getcredentials%
107       chor_credentialmng send{encryption} message setparams(
              parameters)
108       #vp i_auth_type ifOneSelected( fingerprint fingervein iris
              face)# repeat noofbiometricauthtypeselected times(
109           user send {chor_credentialmng} message getcredentials(
                  deviceparameter)
110           %comp processeddata =chor_credentialmng.getcredentials%
111           chor_credentialmng send{encryption} message setparams(
                  parameters)
112       )
113       encryption receive message encrypt(credentials)
114       #vp i_auth_mode ifSelected(mode_online)# encryption send
              {thirdparty} message savehasheddata(hasheddata)
115       #vp i_auth_mode ifSelected(mode_offline)# encryption send
              {storage} message sethasheddata(hasheddata)
116       interfaceprep send{responsewindow} message show()
117     )
```

```
Choreography interface chor_adaptablesecuritysystem of
    adaptablesecuritysystem

  function verify
    precondition(authentication_mode_selected == true)
    postcondition  (verification_result_set == true)
    input(user_info)
    output response

  function enroll
    output enrollmentnotification

  portName verifyuser binding hostname:8082

  required interfaces
    from chor_credentialmng function { getcredentials }
    from chor_alert function { alert }
```

```
Configuration interface vconf_adaptablesecuritysystem of
    choreography adaptablesecuritysystem

  //determines number of different biometric authentication types
  @composition
  internalVP i_auth_type:
    mandatory
      variant username_passw
    optional
      variant onetimepassw
      variant esign
    alternative
      variant fingerprint
      variant fingervein
      variant iris
      variant face
      (min:1,max:2)
    bindingTime runtime

```

```
19  //determines authentication mode
20  @composition
21  internalVP i_auth_mode:
22    alternative
23      variant mode_online:activateMethods(service:thirdparty,funct:
            getconnection,savehasheddata,verify)
24      variant mode_offline:activateMethods(service:storage,funct:
            gethasheddata)
25    (min:1,max:1)
26    bindingTime devtime
27
28  //determines transaction type
29  @composition
30  internalVP i_transaction_type:
31    optional
32      variant realtransaction
33      variant faketransaction
34    bindingTime devtime
35
36  //determines the content of encryption parameters
37
38  @vconfrealization
39  internalVP i_encryption_parameters:
40    alternative
41      variant defaultparams
42      variant setparams
43      (min:1,max:1)
44    bindingTime runtime
45
46  configuration authentication_type:
47    varType externalVP
48    optional
49      variant userinfo
50      variant biometrics
51    realization "it is realized by i_encryption_parameters and i
        _auth_type variability points"
52    confvariant userinfo mapping
```

```
53        VPName i_encryption_parameters selectedVariants(defaultparams)
54      confvariant biometrics mapping
55        VPName i_auth_type selectedVariants(fingerprint fingervein
              iris face; min:1, max:1)
56        VPName i_encryption_parameters selectedVariants(setparams)
57      defaultVariant userinfo
58      type parameterization
59      bindingTime devtime
60
61    configuration authentication_mode:
62      varType externalVP
63      alternative
64        variant online
65        variant offline
66      (min:1,max:1)
67      realization "it is realized by i_auth_mode and i_encryption
            _parameters variability points, setting params for
            sessionkey"
68      confvariant online mapping
69        VPName i_auth_mode selectedVariants(mode_online)
70        VPName i_encryption_parameters selectedVariants(setparams)
71      confvariant offline mapping
72        VPName i_auth_mode selectedVariants(mode_offline)
73      defaultVariant offline
74      type parameterization
75      bindingTime devtime
76
77    configuration fake_transaction_enabling:
78      varType externalVP
79      optional
80        variant fake_trans
81        variant real_trans
82      realization "it is realized by i_transaction_type variability
            point"
83      confvariant fake_trans mapping
84        VPName i_transaction_type selectedVariants(faketransaction)
85      confvariant real_trans mapping
```

```
86        VPName i_transaction_type selectedVariants(realtransaction)
87      defaultVariant fake_trans
88      type addition
89      bindingTime devtime
90
91    Constraints
92      i_auth_type face requires i_auth_mode selectedVariants(mode
          _online)
93          i_auth_mode mode_online const protocol="https"
94          i_auth_type esign const i_encryption_parameters
              defaultparams=valueOf{username_passw esign}
95          i_auth_type esign const i_encryption_parameters Mobile =
              valueOf{Mobile PC}
96
97    Parameter Settings
98      parameter noofbiometricauthtypeselected = #ofVariantsSelected
          {fingerprint fingervein iris face} Of i_auth_type
99      parameter defaultparams = value(username_passw,onetimepassw,
          esign)
100     parameter fakeinterface existswhenselected{i_transaction_type.
          faketransaction}


1   choreography alert
2
3     import configuration vconf_alert
4
5     import service gpslocator
6     import service camera with configuration vconf_camera
7     import service alertsender
8
9     Function alert:
10      sequence (
11        gpslocator receive message getaddress()
12        #vp emergency_notification ifSelected(telephonecall)# camera
            send {alertsender} message call(destination)
13        #vp emergency_content ifSelected(picture)# sequence (
14          gpslocator send{camera} message takepicture()
```

```
15          camera send{alertsender} message setcontent(picture)
16        )
17        #vp emergency_content ifSelected(videorecord)# sequence (
18          gpslocator send{camera} message recordvideo(duration)
19          camera send{alertsender} message setcontent(video)
20        )
21        #vp emergency_notification ifSelected(mediasend)# camera send
              {alertsender} message sendmediacontent()
22      )
```

```
1  Choreography interface chor_alert of alert
2
3    function alert
4      precondition(session == false )
5      postcondition(thirdparty_alerted == true and alarm_mode == true)
6
7    portName alert binding hostname:5555
```

```
1  Configuration interface vconf_alert of choreography alert
2
3    externalVP destination:
4      mandatory
5        variant bank
6      optional
7        variant police
8      bindingTime devtime
9
10   externalVP emergency_notification:
11     optional
12       variant telephonecall:activateMethods(service:alertsender,
             funct:setcontent,call)
13       variant mediasend:activateMethods(service:alertsender,funct:
             setcontent,sendmediacontent)
14     bindingTime devtime
15
16   externalVP emergency_content:
17     mandatory
18       variant gpsdata
```

```
      variant datetime
    optional
      variant picture
      variant videorecord
    bindingTime devtime

choreography credentialmng

  import configuration vconf_credentialmng

  import service connection
  import service imageretrieval
  import service credentials

  Context Elements
    biometric_data ""
    processeddata  ""
    deviceparameter ""

  Function getcredentials:
    sequence (
      #vp devicecon ifSelected(biometricdevice) #sequence (
        connection receive message connectdevice(deviceid)
        %comp biometric_data =connection.connectdevice%
        connection send {imageretrieval} message extract_features(
            biometric_data)
        %comp processeddata =imageretrieval.extract_features%
      )
      credentials receive message getcredential()
    )

Choreography interface chor_credentialmng of  credentialmng

  function getcredentials
    input(deviceparameter)
    output processeddata

  portName credentialmng binding localhost:8050
```

```
8
9     required interfaces

1   Configuration interface vconf_credentialmng of choreography
        credentialmng
2
3     externalVP devicecon:
4       mandatory
5         variant usernamepassword
6       optional
7         variant biometricdevice
8       bindingTime devtime

1   Service interface alertsender
2
3     function setcontent
4       input(content)
5
6     function call
7       input(destination)
8
9     function sendmediacontent
10
11    portName alertsender binding localhost:8040

1   Service interface attemptcalc
2
3     function calculate_wrong_attempts
4       postcondition(connection_closed==true or user_warned == true)
5       input(session_id)
6       output no_wrong_attempts
7
8     portName attemptcalc binding hostname:5055

1   Service interface camera
2
3     function takepicture
4       output picture
5
```

```
  function recordvideo
    input(duration)
    output video

  portName camera binding localhost:8020

Configuration interface vconf_camera of service camera

  externalVP mode:
    optional
      variant picture :activateMethods(takepicture)
      variant video :activateMethods(recordvideo)
    bindingTime devtime

Service interface comparison

  function compare
    output result

  function fakeanalysis
    input(comparisonresult)
    output result

  portName comparison binding hostname:2011

Configuration interface vconf_comparison of service comparison
    @vconfservice
    vp analysis:
    optional
      variant fake :activateMethods(fakeanalysis,compare)
      variant real:activateMethods(compare)
      bindingTime devtime

  abstract process definition
    sequence (
      comparison receive from{storage} message compare(data)
      comparison send{attemptcalc} message calculate_wrong_attempts(
          result)
```

```
    )
    #vp analysis ifSelected(fake)# sequence (
      comparison receive from{thirdparty} message fakeanalysis(
          comparisonresult)
      comparison send{interfaceprep} message prepareinterface()
    )
    comparison send{responsewindow} message show()

Service interface connection

  function openconnection
    output sessionid

  function connectdevice
    input(deviceid)

  function closeconnection
    precondition(conn_opened == true)
    postcondition(conn_closed == true)
    input(session_id)

  portName connection binding hostname:4544

Service interface credentials

  function getcredential
    precondition(credentials_entered == true)
    postcondition(credentials_gathered == true)
    output credentials

  portName credentials_gathering binding hostname:8080

Service interface encryption

  function encrypt
    precondition(sessioncreated == true)
    postcondition(data_encrypted == true)
    input(credentials)
```

```
    output hasheddata

  function setparams
    precondition(params_required == true)
    postcondition(set_params == true)
    input(parameters)

  portName encryption binding hostname:8082
```

```
Configuration interface vconf_encryption of service encryption
  vp encryption_params:
    alternative
      variant withparams:activateMethods(encrypt,setparams):
          setParameter(toFunct:encrypt,parameter:params)
      variant withdefaultparams:activateMethods(encrypt)
      (min:1,max:1)
    bindingTime runtime
```

```
Service interface gpslocator

  function getaddress
    input(longtitudelattitude)
    output address

  portName gpslocator binding localhost:8050
```

```
Service interface imageretrieval
  function extract_features
    precondition(bio_data_gathered == true)
    postcondition(features_extracted == true)
    input(biometric_data)
    output extracted_template

  portName imageretrieval binding hostname:8080
```

```
Service interface interfaceprep

  function prepareinterface
    postcondition(interface_prepared == true)
```

```
    output interface_content

  portName interfacecontent binding hostname:2000

Configuration interface vm_interfaceprep of service interfaceprep
  vp content:
    optional
      variant fake
      variant real
    bindingTime devtime

Service interface responsewindow

  function show
    precondition(content_prepared == true)
    postcondition(result_showed == true)
    input(content)
    output response_window

  portName responsewindow binding hostname:4444

Service interface storage

  function gethasheddata
    precondition(data_stored == true)
    output storedhasheddata

  function sethasheddata
    input(hasheddata)

  portName storage binding hostname:2010

Service interface thirdparty

  invariant connection == true

  function getconnection
    precondition(driver_installation == true)
    postcondition(sessioncreated == true)
```

```
8       input(user_id)
9       output session_key
10
11    function savehasheddata
12      precondition(data_prepared == true)
13      postcondition(successfull_save == true or failed_save == true)
14      input(hasheddata)
15      output successful_save_ack
16
17    function verify
18      precondition(data_prepared == true)
19      postcondition(user_verified == true or user_denied == true)
20      input(data)
21      output response_warning
22
23    portName datasending binding hostname:8081

1   Configuration interface vm_thirdparty of service thirdparty
2     vp user_device:
3       optional
4         variant PC
5         variant Mobile
6         variant ATM
7         variant PDA
8       bindingTime runtime

1   Service interface user
2
3     function providecredentails
4       output usercredentials
5
6     portName user binding locahost:8045

1   Service interface warning
2
3     function warn
4       precondition(session == false)
5       postcondition(user_warned == true)
```

```
6       output warning_message
7
8     portName warning binding hostname:4555
```

APPENDIX D

GENERATED FTS FILES FOR VERIFICATION OF CASE STUDIES

D.1 TVL Feature Model File for Travel Itinerary System

```
root Application{
  group allOf{
    Adaptablesecuritysystem group allOf{
      Authentication_type group someOf{
        Biometrics group allOf{
          I_encryption_parameters group allOf{
            Setparams
          },
          I_auth_type group [1..1]{
            Fingerprint,
            Fingervein,
            Iris,
            Face
          }
        },
        Userinfo group allOf{
          I_encryption_parameters_1 group allOf{
            Defaultparams
          }
        }
      },
      Authentication_mode group oneOf{
        Offline group allOf{
```

```
24            I_auth_mode group allOf{
25              Mode_offline
26            }
27          },
28          Online group allOf{
29            I_auth_mode_2 group allOf{
30              Mode_online
31            },
32            I_encryption_parameters_3 group allOf{
33              Setparams_4
34            }
35          }
36        },
37        Fake_transaction_enabling group someOf{
38          Real_trans group allOf{
39            I_transaction_type group allOf{
40              Realtransaction
41            }
42          },
43          Fake_trans group allOf{
44            I_transaction_type_5 group allOf{
45              Faketransaction
46            }
47          }
48        }
49      },
50      Credentialmng group allOf{
51        Devicecon group someOf{
52          Biometricdevice
53        },
54        Devicecon_6 group allOf{
55          Usernamepassword
56        }
57      },
58      Alert group allOf{
59        Emergency_notification group someOf{
60          Telephonecall ,
```

```
61          Mediasend
62        },
63        Destination group someOf{
64          Police
65        },
66        Destination_7 group allOf{
67          Bank
68        },
69        Emergency_content group someOf{
70          Picture,
71          Videorecord
72        },
73        Emergency_content_8 group allOf{
74          Gpsdata,
75          Datetime
76        }
77      }
78    }
79    Setparams_4 -> Setparams;
80  }
```

D.2 fPromela File for Travel Itinerary System

```
1  chan chan_tempthirdparty_getconnection = [1] of {byte}
2  chan chan_comparisoninterfaceprep_prepareinterface = [1] of {byte}
3  chan chan_chor_credentialmngencryption_setparams = [4] of {byte}
4  chan chan_storagecomparison_compare = [1] of {byte}
5  chan chan_comparisonattemptcalc_calculate_wrong_attempts = [1] of
       {byte}
6  chan chan_tempencryption_encrypt = [1] of {byte}
7  chan chan_encryptionstorage_gethasheddata = [1] of {byte}
8  chan chan_storagecomparison = [1] of {byte}
9  chan chan_attemptcalcconnection_closeconnection = [1] of {byte}
10 chan chan_attemptcalcwarning_warn = [1] of {byte}
11 chan chan_encryptionthirdparty_verify = [1] of {byte}
12 chan chan_comparisonchor_alert_alert = [1] of {byte}
13 chan chan_encryptionthirdparty_savehasheddata = [1] of {byte}
```

```
chan chan_comparisonresponsewindow_show = [2] of {byte}
chan chan_thirdpartycomparison_fakeanalysis = [1] of {byte}
chan chan_thirdpartyencryption_setparams = [1] of {byte}
chan chan_userchor_credentialmng_getcredentials = [4] of {byte}
chan chan_interfaceprepresponsewindow_show = [2] of {byte}
chan chan_encryptionstorage_sethasheddata = [1] of {byte}
chan chan_cameraalertsender_sendmediacontent = [1] of {byte}
chan chan_tempgpslocator_getaddress = [1] of {byte}
chan chan_cameraalertsender_setcontent = [2] of {byte}
chan chan_cameraalertsender_call = [1] of {byte}
chan chan_gpslocatorcamera_takepicture = [1] of {byte}
chan chan_gpslocatorcamera_recordvideo = [1] of {byte}
chan chan_tempconnection_connectdevice = [1] of {byte}
chan chan_connectionimageretrieval_extract_features = [1] of {byte}
chan chan_tempcredentials_getcredential = [1] of {byte}
byte deviceid;
byte biometric_data;
byte temp;
byte destination;
byte picture;
byte duration;
byte video;
byte notification;
byte deviceparameter;
byte parameters;
byte credentials;
byte data;
byte comparisonresult;
byte result;
byte response_warning;
byte hasheddata;
int wrongattempts=0;
int noofbiometricauthtypeselected=0;
byte defaultparams=117;
byte temp_processeddata=98;
byte deviceparameter=34;
byte usernamepass=34;
```

```
51 byte processeddata=34;
52 byte biometric_data=34;
53 bool fakeinterface=0;
54 bool temp_fakeinterface=1;
55
56 typedef features {
57   bool Setparams;
58   bool Fingerprint;
59   bool Fingervein;
60   bool Iris;
61   bool Face;
62   bool Defaultparams;
63   bool Mode_offline;
64   bool Mode_online;
65   bool Realtransaction;
66   bool Faketransaction;
67   bool Biometricdevice;
68   bool Usernamepassword;
69   bool Telephonecall;
70   bool Mediasend;
71   bool Police;
72   bool Bank;
73   bool Picture;
74   bool Videorecord;
75   bool Gpsdata;
76   bool Datetime
77 };
78 features f;
79
80 active proctype encryption() {
81     chan_tempencryption_encrypt!credentials;
82     chan_tempencryption_encrypt!credentials;
83 }
84 active proctype thirdparty() {
85     chan_tempthirdparty_getconnection!temp;
86 }
87 active proctype credentials() {
```

```
88      chan_tempcredentials_getcredential!temp;
89  }
90  active proctype verify() {
91    {
92      gd
93      ::(( f.Fingerprint && !f.Fingervein && !f.Iris && !f.Face) || (
            f.Fingervein && !f.Fingerprint && !f.Iris && !f.Face) || (
          f.Iris && !f.Fingerprint && !f.Fingervein && !f.Face) || ( f
          .Face && !f.Fingerprint && !f.Fingervein && !f.Iris) ) ->
94        do
95        ::(noofbiometricauthtypeselected!= 0) ->
96        chan_userchor_credentialmng_getcredentials!34;
97        chan_userchor_credentialmng_getcredentials?processeddata;
98        chan_chor_credentialmngencryption_setparams!parameters;
99        noofbiometricauthtypeselected = noofbiometricauthtypeselected
              - 1;
100       ::else -> break;
101     od
102     ::else -> skip;
103         dg;
104   };
105   {
106     gd
107     ::f.Mode_online ->
108     temp = temp+1;
109       {
110       chan_tempthirdparty_getconnection?temp;
111       };
112       {
113       chan_thirdpartyencryption_setparams!parameters;
114       };
115     ::else -> skip;
116         dg;
117   };
118   {
119         chan_userchor_credentialmng_getcredentials!34;
120         chan_userchor_credentialmng_getcredentials?usernamepass;
```

```
121     };
122     {
123           chan_chor_credentialmngencryption_setparams!parameters;
124     };
125     {
126           chan_tempencryption_encrypt?credentials;
127     };
128     {
129       gd
130       ::f.Mode_online ->
131       temp = temp+1;
132         {
133         chan_encryptionthirdparty_verify!data;
134         };
135         {
136           gd
137           ::f.Faketransaction ->
138           chan_thirdpartycomparison_fakeanalysis!1;
139           chan_thirdpartycomparison_fakeanalysis?fakeinterface;
140           ::else -> skip;
141         dg;
142         };
143       ::else -> skip;
144           dg;
145     };
146     {
147       gd
148       ::f.Mode_offline ->
149       temp = temp+1;
150         {
151         chan_encryptionstorage_gethasheddata!temp;
152         chan_storagecomparison!temp;
153         };
154         {
155           gd
156           ::f.Faketransaction ->
157           chan_storagecomparison_compare!temp;
```

```
158           ::else -> skip;
159         dg;
160         };
161       ::else -> skip;
162           dg;
163     };
164     {
165       if
166       ::( fakeinterface==temp_fakeinterface) ->
167         {
168         chan_comparisonattemptcalc_calculate_wrong_attempts!0;
169         chan_comparisonattemptcalc_calculate_wrong_attempts?
              wrongattempts;
170         };
171         {
172           if
173           ::( wrongattempts == 3) ->
174           chan_comparisonresponsewindow_show!temp;
175           chan_attemptcalcconnection_closeconnection!temp;
176           ::else -> skip;
177         fi;
178         };
179         {
180           if
181           ::( wrongattempts < 3) ->
182           chan_comparisonresponsewindow_show!temp;
183           chan_attemptcalcwarning_warn!response_warning;
184           ::else -> skip;
185         fi;
186         };
187       ::else -> skip;
188           fi;
189     };
190     {
191       if
192       ::( fakeinterface==temp_fakeinterface && processeddata==temp
            _processeddata) ->
```

```
193         gd
194         ::f.Faketransaction ->
195         temp = temp+1;
196           {
197           chan_comparisoninterfaceprep_prepareinterface!temp;
198           };
199           {
200           chan_interfaceprepresponsewindow_show!temp;
201           };
202         chan_comparisonchor_alert_alert!temp;
203         ::else -> skip;
204               dg;
205       ::else -> skip;
206           fi;
207   };
208 }
209 active proctype enroll() {
210   {
211           chan_userchor_credentialmng_getcredentials!34;
212           chan_userchor_credentialmng_getcredentials?usernamepass;
213   };
214   {
215           chan_chor_credentialmngencryption_setparams!parameters;
216   };
217   {
218     gd
219     ::(( f.Fingerprint && !f.Fingervein && !f.Iris && !f.Face) || (
            f.Fingervein && !f.Fingerprint && !f.Iris && !f.Face) || (
          f.Iris && !f.Fingerprint && !f.Fingervein && !f.Face) || ( f
          .Face && !f.Fingerprint && !f.Fingervein && !f.Iris) ) ->
220       do
221       ::(noofbiometricauthtypeselected!= 0) ->
222       chan_userchor_credentialmng_getcredentials!34;
223       chan_userchor_credentialmng_getcredentials?processeddata;
224       chan_chor_credentialmngencryption_setparams!parameters;
225       noofbiometricauthtypeselected = noofbiometricauthtypeselected
            - 1;
```

```
226         ::else -> break;
227                 od
228       ::else -> skip;
229           dg;
230     };
231     {
232           chan_tempencryption_encrypt?credentials;
233     };
234     {
235       gd
236       ::f.Mode_online ->
237       chan_encryptionthirdparty_savehasheddata!hasheddata;
238       ::else -> skip;
239           dg;
240     };
241     {
242       gd
243       ::f.Mode_offline ->
244       chan_encryptionstorage_sethasheddata!hasheddata;
245       ::else -> skip;
246           dg;
247     };
248     {
249           chan_interfaceprepresponsewindow_show!temp;
250     };
251 }
252
253 active proctype alert() {
254     {
255           chan_tempgpslocator_getaddress!temp;
256     };
257     {
258           chan_tempgpslocator_getaddress?temp;
259     };
260     {
261       gd
262       ::f.Telephonecall ->
```

```
263       chan_cameraalertsender_call!destination;
264       ::else -> skip;
265           dg;
266     };
267     {
268       gd
269       ::f.Picture ->
270       temp = temp+1;
271         {
272                   chan_gpslocatorcamera_takepicture!temp;
273         };
274         {
275                   chan_cameraalertsender_setcontent!picture;
276         };
277       ::else -> skip;
278           dg;
279     };
280     {
281       gd
282       ::f.Videorecord ->
283       temp = temp+1;
284         {
285                   chan_gpslocatorcamera_recordvideo!duration;
286         };
287         {
288                   chan_cameraalertsender_setcontent!video;
289         };
290       ::else -> skip;
291           dg;
292     };
293     {
294       gd
295       ::f.Mediasend ->
296       chan_cameraalertsender_sendmediacontent!temp;
297       ::else -> skip;
298           dg;
299     };
```

```
300 }
301
302 active proctype getcredentials() {
303   {
304     gd
305     ::f.Biometricdevice ->
306     temp = temp+1;
307       {
308                 chan_tempconnection_connectdevice!deviceid;
309       };
310       {
311                 chan_tempconnection_connectdevice?biometric_data;
312       };
313       {
314                 chan_connectionimageretrieval_extract_features!34;
315                 chan_connectionimageretrieval_extract_features?
                        processeddata;
316       };
317     ::else -> skip;
318         dg;
319   };
320   {
321         chan_tempcredentials_getcredential?temp;
322   };
323 }
```

D.3 TVL Feature Model File for Adaptable Security System

```
1 root Application{
2   group allOf{
3     Travelitinerary group allOf{
4       Itinerary group oneOf{
5         Vacationpackage group allOf{
6           Facilities group [0..3]{
7             Cruise,
8             Carrental,
9             Activities
```

```
10            },
11            Booking group allOf{
12              Hotel,
13              Airline
14            }
15          },
16          Regular group allOf{
17            Facilities_1 group [0..3]{
18              Cruise_2,
19              Carrental_3,
20              Activities_4
21            },
22            Booking_5 group [1..2]{
23              Hotel_6,
24              Airline_7
25            }
26          }
27        }
28      }
29    }
30    Activities_4 -> Activities;
31    Hotel_6 -> Hotel;
32    Carrental_3 -> Carrental;
33    Airline_7 -> Airline;
34    Cruise_2 -> Cruise;
35  }
```

D.4 fPromela File for Adaptable Security System

```
1  chan chan_airlinetraveler = [1] of {byte}
2  chan chan_travelagencyairline_requestprice = [1] of {byte, byte}
3  chan chan_travelagencyactivityprovider_requestprice = [1] of {byte,
       byte}
4  chan chan_travelagencytraveler_getavaliabletripoptions = [1] of
       {byte}
5  chan chan_travelertravelagency_selectcruise = [1] of {byte}
6  chan chan_travelertravelagency_selectcarrental = [1] of {byte}
```

```
7  chan chan_travelertravelagency_querytrip = [1] of {byte, byte, byte}
8  chan chan_travelertravelagency_setselectedactivities = [2] of {byte}
9  chan chan_travelertravelagency_selectairline = [1] of {byte}
10 chan chan_travelertravelagency_selecthotel = [1] of {byte}
11 chan chan_travelagencyhotel_requestprice = [1] of {byte, byte, byte}
12 chan chan_hoteltraveler = [1] of {byte}
13 chan chan_travelagencycruise_requestprice = [1] of {byte, byte}
14 chan chan_travelagencyairline_bookflight = [1] of {byte, byte}
15 chan chan_travelagencyairline_processticket = [1] of {byte}
16 chan chan_travelagencytraveler_getconfirmedplan = [1] of {byte}
17 chan chan_travelagencyairline_bookflightnot = [1] of {byte}
18 chan chan_travelagencyactivityprovider_listactivities = [1] of {byte
       , byte}
19 chan chan_travelagencytraveler_gettripplancancelation = [2] of {byte
       }
20 chan chan_travelagencyhotel_bookroom = [1] of {byte, byte, byte}
21 chan chan_activityprovidertraveler = [1] of {byte}
22 chan chan_travelagencycarrental_requestprice = [1] of {byte, byte}
23 chan chan_travelagencyhotel_bookroomnot = [1] of {byte}
24 chan chan_travelagencyhotel_processvoucher = [1] of {byte}
25 byte temp;
26 byte notification;
27 byte startdate;
28 byte enddate;
29 byte details;
30 byte packageid;
31 byte date;
32 byte carmodel;
33 byte place;
34 byte selectedlist;
35 byte travelerID;
36 byte airlineID;
37 byte hotelID;
38 byte activitylist;
39 byte temp;
40 byte arrival;
41 byte departure;
```

```
42 byte customerID;
43 int intelem=0;
44 byte stringelem=115;
45 bool flighttticketconfirmation=0;
46 bool hotelbookingconfirmation=0;
47 bool carrentalconfirmation=0;
48 bool activityconfirmation=0;
49 bool cruiseconfirmation=0;
50 bool temp_flighttticketconfirmation =1;
51 bool temp_hotelbookingconfirmation =1;
52
53 typedef features {
54   bool Cruise;
55   bool Carrental;
56   bool Activities;
57   bool Hotel;
58   bool Airline
59 };
60 features f;
61
62 active proctype proc_chan_travelagencyairline_bookflightnot(){
63     chan_travelagencyairline_bookflightnot!notification
64 }
65 active proctype proc_chan_travelagencyhotel_bookroomnot(){
66     chan_travelagencyhotel_bookroomnot!notification
67 }
68 active proctype planitinerary() {
69   {
70   chan_travelertravelagency_querytrip!startdate,enddate,details;
71   };
72   {
73     gd
74     ::f.Airline ->
75     chan_travelagencyairline_requestprice!startdate,enddate;
76     ::else -> skip;
77         dg;
78     gd
```

```
79      ::f.Hotel ->
80      chan_travelagencyhotel_requestprice!startdate,enddate,details;
81      ::else -> skip;
82          dg;
83      gd
84      ::f.Cruise ->
85      chan_travelagencycruise_requestprice!packageid,date;
86      ::else -> skip;
87          dg;
88      gd
89      ::f.Carrental ->
90      chan_travelagencycarrental_requestprice!carmodel,date;
91      ::else -> skip;
92          dg;
93      gd
94      ::f.Activities ->
95      temp = temp+1;
96        {
97        chan_travelagencyactivityprovider_listactivities!place,date;
98        chan_activityprovidertraveler!temp;
99        };
100       {
101       chan_travelertravelagency_setselectedactivities!selectedlist;
102       };
103       {
104       chan_travelagencyactivityprovider_requestprice!selectedlist,
              date;
105       };
106     ::else -> skip;
107         dg;
108   };
109   {
110         chan_travelagencytraveler_getavaliabletripoptions!travelerID
              ;
111   };
112   {
```

```
113         chan_travelagencytraveler_getavaliabletripoptions?travelerID
                ;
114     };
115     {
116       gd
117       ::f.Airline ->
118       temp = temp+1;
119         {
120         chan_travelertravelagency_selectairline!0;
121         chan_travelertravelagency_selectairline?intelem;
122         };
123       ::else -> skip;
124           dg;
125     };
126     {
127       gd
128       ::f.Hotel ->
129       temp = temp+1;
130         {
131         chan_travelertravelagency_selecthotel!115;
132         chan_travelertravelagency_selecthotel?stringelem;
133         };
134       ::else -> skip;
135           dg;
136     };
137     {
138       gd
139       ::f.Cruise ->
140       temp = temp+1;
141         {
142                   chan_travelertravelagency_selectcruise!1;
143                   chan_travelertravelagency_selectcruise?
                          cruiseconfirmation;
144         };
145       ::else -> skip;
146           dg;
147     };
```

```
148     {
149       gd
150       ::f.Carrental ->
151       temp = temp+1;
152         {
153         chan_travelertravelagency_selectcarrental!1;
154         chan_travelertravelagency_selectcarrental?
                carrentalconfirmation;
155         };
156       ::else -> skip;
157           dg;
158     };
159     {
160       gd
161       ::f.Activities ->
162       temp = temp+1;
163         {
164         chan_travelertravelagency_setselectedactivities!1;
165         chan_travelertravelagency_setselectedactivities?
              activityconfirmation;
166         };
167         {
168         chan_travelagencytraveler_getconfirmedplan!temp;
169         };
170       ::else -> skip;
171           dg;
172     };
173     {
174       if
175       ::
176         if
177         ::if
178         ::( hotelbookingconfirmation == temp_hotelbookingconfirmation
                 && flightticketconfirmation == temp
               _flightticketconfirmation ) ->
179           gd
180           ::f.Airline ->
```

```
181         temp = temp+1;
182           {
183           chan_travelagencyairline_bookflight!arrival,departure;
184           chan_travelagencyairline_bookflightnot?notification;
185           };
186           {
187           chan_travelagencyairline_processticket!customerID;
188           chan_airlinetraveler!temp;
189           };
190         ::else -> skip;
191                 dg;
192         gd
193         ::f.Hotel ->
194         temp = temp+1;
195           {
196           chan_travelagencyhotel_bookroom!arrival,departure,details;
197           chan_travelagencyhotel_bookroomnot?notification;
198           };
199           {
200           chan_travelagencyhotel_processvoucher!customerID;
201           chan_hoteltraveler!temp;
202           };
203         ::else -> skip;
204                 dg;
205       chan_travelagencytraveler_getconfirmedplan?temp;
206       ::else -> skip;
207             fi;
208     ::{
209             chan_travelagencytraveler_gettripplancancelation!temp;
210     };
211     {
212             chan_travelagencytraveler_gettripplancancelation!temp;
213     };
214     {
215             chan_travelagencytraveler_gettripplancancelation?temp;
216     };
217         fi;
```

```
218         ::else -> skip;
219         fi;
220         };
221 }
```

APPENDIX E

GENERATED BPEL4CHOR, VXBPEL AND BPEL FILES

E.1 BPEL4Chor Files - Topology, Grounding and PBDs

```
1  <topology name="travelitinerary"
2    targetNamespace="urn:"travelitinerary"
3    xmlns="urn:HPI_IAAS:choreography:schemas:choreography:topology
         :2006/12"
4    xmlns:chordef="http://example.com/"travelitinerary"
5    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6    xsi:schemaLocation="urn:HPI_IAAS:choreography:schemas:
         choreography:topology:2006/12 http://www.iaas.uni-stuttgart.de
         /schemas/bpel4chor/topology.xsd">
7
8    <participantTypes>
9      <participantType name="querytrip_sender_type"
           participantBehaviorDescription="querytrip_sender:querytrip
           _sender"/>
10     <participantType name="cruise_type"
           participantBehaviorDescription="cruise:cruise"/>
11     <participantType name="carrental_type"
           participantBehaviorDescription="carrental:carrental"/>
12     <participantType name="traveler-tra_type"
           participantBehaviorDescription="traveler-tra:traveler-tra"/>
13   </participantTypes>
14
15   <participants>
16     <participant name="cruise" type="cruise_type"/>
17     <participant name="carrental" type="carrental_type"/>
```

```
18      <participant name="traveler-tra" type="traveler-tra_type"/>
19      <participantSet name="querytrip_sender" type="querytrip_sender
            _type" forEach="querytripFE/>
20        <participant name="travelagency-tra type="querytrip_sender
            _type" forEach="querytripFE/>
21        <participant name="airline type="querytrip_sender_type"
            forEach="querytripFE/>
22      </participantSet>
23    </participants>
24
25    <messageLinks>
26      <messageLink name="querytripLink" sender="traveler-tra"
            sendActivity="requestquerytrip" receiver="travelagency-tra"
            receiveActivity="querytrip" messageName="1" />
27      <messageLink name="requestpriceLink" sender="travelagency-tra"
            sendActivity="requestrequestprice" receiver="airline"
            receiveActivity="requestprice" messageName="2" />
28      <messageLink name="requestpriceLink" sender="travelagency-tra"
            sendActivity="requestrequestprice" receiver="cruise"
            receiveActivity="requestprice" messageName="3" />
29      <messageLink name="requestpriceLink" sender="travelagency-tra"
            sendActivity="requestrequestprice" receiver="carrental"
            receiveActivity="requestprice" messageName="4" />
30      <messageLink name="getavaliabletripoptionsLink" sender="
            travelagency-tra" sendActivity="
            requestgetavaliabletripoptions" receiver="traveler-tra"
            receiveActivity="getavaliabletripoptions" messageName="5" />
31      <messageLink name="selectairlineLink" sender="traveler-tra"
            sendActivity="requestselectairline" receiver="travelagency-
            tra" receiveActivity="selectairline" messageName="6" />
32      <messageLink name="selectcruiseLink" sender="traveler-tra"
            sendActivity="requestselectcruise" receiver="travelagency-
            tra" receiveActivity="selectcruise" messageName="7" />
33      <messageLink name="selectcarrentalLink" sender="traveler-tra"
            sendActivity="requestselectcarrental" receiver="travelagency
            -tra" receiveActivity="selectcarrental" messageName="8" />
```

```
34      <messageLink name="bookflightLink" sender="travelagency-tra"
            sendActivity="requestbookflight" receiver="airline"
            receiveActivity="bookflight" messageName="9" />
35      <messageLink name="processticketLink" sender="travelagency-tra"
             sendActivity="requestprocessticket" receiver="airline"
            receiveActivity="processticket" messageName="10" />
36      <messageLink name="sendconfirmedplanLink" sender="travelagency-
            tra" sendActivity="requestsendconfirmedplan" receiver="
            traveler-tra" receiveActivity="sendconfirmedplan"
            messageName="11" />
37      <messageLink name="getconfirmedplanLink" sender="travelagency-
            tra" sendActivity="requestgetconfirmedplan" receiver="
            traveler-tra" receiveActivity="getconfirmedplan" messageName
            ="12" />
38      <messageLink name="sendtripplancancelationLink" sender="
            travelagency-tra" sendActivity="
            requestsendtripplancancelation" receiver="traveler-tra"
            receiveActivity="sendtripplancancelation" messageName="13" /
            >
39      <messageLink name="gettripplancancelationLink" sender="
            travelagency-tra" sendActivity="
            requestgettripplancancelation" receiver="traveler-tra"
            receiveActivity="gettripplancancelation" messageName="14" />
40      <messageLink name="cancelorderLink" sender="traveler-tra"
            sendActivity="cancelorder" receiver="travelagency-tra"
            receiveActivity="getcancelorder" messageName="15" />
41    </messageLinks>
42  </topology>


1   <grounding topology="travelitinerary">
2   xmlns:top="urn:travelitinerary" xmlns:zab="urn:travelagency-tra"
        xmlns:rst="urn:traveler-tra" xmlns:fgh="urn:airline" xmlns:jkl="
        urn:carrental" xmlns:ghi="urn:cruise" xmlns="urn:HPI_IAAS:
        choreography:schemas:choreography:grounding:2006/12">
        <messageLinks>
3   <messageLink name="querytripLink" portType="zab:travelagency-tra_pt
        operation="querytrip"/>
```

```
4 <messageLink name="requestpriceLink" portType="fgh:airline_pt
    operation="requestprice"/>
5 <messageLink name="requestpriceLink" portType="ghi:cruise_pt
    operation="requestprice"/>
6 <messageLink name="requestpriceLink" portType="jkl:carrental_pt
    operation="requestprice"/>
7 <messageLink name="getavaliabletripoptionsLink" portType="rst:
    traveler-tra_pt operation="getavaliabletripoptions"/>
8 <messageLink name="selectairlineLink" portType="zab:travelagency-
    tra_pt operation="selectairline"/>
9 <messageLink name="selectcruiseLink" portType="zab:travelagency-tra
    _pt operation="selectcruise"/>
10 <messageLink name="selectcarrentalLink" portType="zab:travelagency-
    tra_pt operation="selectcarrental"/>
11 <messageLink name="bookflightLink" portType="fgh:airline_pt
    operation="bookflight"/>
12 <messageLink name="processticketLink" portType="fgh:airline_pt
    operation="processticket"/>
13 <messageLink name="sendconfirmedplanLink" portType="rst:traveler-
    tra_pt operation="sendconfirmedplan"/>
14 <messageLink name="getconfirmedplanLink" portType="rst:traveler-tra
    _pt operation="getconfirmedplan"/>
15 <messageLink name="sendtripplancancelationLink" portType="rst:
    traveler-tra_pt operation="sendtripplancancelation"/>
16 <messageLink name="gettripplancancelationLink" portType="rst:
    traveler-tra_pt operation="gettripplancancelation"/>
17 <messageLink name="cancelorderLink" portType="zab:travelagency-tra
    _pt operation="getcancelorder"/>


1 <process abstractProcessProfile="urn:HPI_IAAS:choreography:profile
    :2006/12" name="travelagency-tra" targetNamespace="urn:booking:
    agency" xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
    abstract" xmlns:wsu="urn:wsu">
2   <sequence>
3     <receive wsu:id="querytrip" createInstance="yes"/>
4     <flow>
5       <invoke wsu:id="requestprice"/>
```

```
6         <invoke wsu:id="requestprice"/>
7         <invoke wsu:id="requestprice"/>
8       </flow>
9       <invoke wsu:id="getavaliabletripoptions"/>
10      <receive wsu:id="selectairline"/>
11      <receive wsu:id="selectcruise"/>
12      <receive wsu:id="selectcarrental"/>
13      <pick>
14        <onMessage wsu:id="getcancelorder"
15          <if>
16            <condition opaque="yes"/>
17            <flow>
18              <sequence>
19                <invoke wsu:id="bookflight"/>
20                <invoke wsu:id="processticket"/>
21              </sequence>
22              <invoke wsu:id="sendconfirmedplan"/>
23            </flow>
24          </if>
25          <invoke wsu:id="sendtripplancancelation"/>
26        </onMessage>
27      </pick>
28    </sequence>
29  </process>
```

```
1  <process abstractProcessProfile="urn:HPI_IAAS:choreography:profile
       :2006/12" name="traveler-tra" targetNamespace="urn:booking:
       agency" xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
       abstract" xmlns:wsu="urn:wsu">
2    <sequence>
3      <forEach parallel="yes" wsu:id="querytripFE">
4        <scope wsu:id="querytripScope">
5          <sequence>
6            <invoke wsu:id="querytrip"/>
7          </sequence>
8        </scope>
9      </forEach>
```

```
10      <receive wsu:id="getavaliabletripoptions"/>
11      <receive wsu:id="getconfirmedplan"/>
12      <receive wsu:id="gettripplancancelation"/>
13    </sequence>
14  </process>
```

E.2 VxBPEL and BPEL Files

```
1  <bpel:process xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
       travelitinerary_planitinerary" ext1:linksAreTransitions="yes"
       ext:disableSelectionFailure="yes" xmlns:ext1="http://www.
       activebpel.org/2009/06/bpel/extension/links" suppressJoinFailure
       ="yes" xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
       ext:createTargetXPath="yes" xmlns:vxbpel="http://www.turkselma.
       com/vxbpel" xmlns:ext="http://www.activebpel.org/2006/09/bpel/
       extension/query_handling" xmlns:abx="http://www.activebpel.org/
       bpel/extension" targetNamespace="http://verificationprocess" aei
       :editStyle="BPMN" xmlns:bpel="http://docs.oasis-open.org/wsbpel
       /2.0/process/executable" xmlns:aei="http://www.activebpel.org
       /2009/02/bpel/extension/ignorable">
2    <bpel:extensions>
3      <bpel:extension mustUnderstand="no" namespace="http://www.
           activebpel.org/2009/02/bpel/extension/ignorable"/>
4      <bpel:extension mustUnderstand="no" namespace="http://www.omg.
           org/spec/BPMN/20100524/DI"/>
5      <bpel:extension mustUnderstand="yes" namespace="http://www.
           activebpel.org/2006/09/bpel/extension/query_handling"/>
6      <bpel:extension mustUnderstand="yes" namespace="http://www.
           activebpel.org/2009/06/bpel/extension/links"/>
7    </bpel:extensions>
8    <bpel:partnerLinks>
9      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
           name="querytrip" partnerRole="Provider" aei:interface="
           travelagencyProcess:TravelagencyProcess" aei:id
           ="4640414432"/>
10     <bpel:partnerLink partnerLinkType="ns:AirlineProcessPLT" name="
           requestprice" partnerRole="Provider" aei:interface="
```

```
        airlineProcess:AirlineProcess" aei:id="2431418264"/>
11      <bpel:partnerLink partnerLinkType="ns:HotelProcessPLT" name="
        requestprice" partnerRole="Provider" aei:interface="
        hotelProcess:HotelProcess" aei:id="1395626079"/>
12      <bpel:partnerLink partnerLinkType="ns:CruiseProcessPLT" name="
        requestprice" partnerRole="Provider" aei:interface="
        cruiseProcess:CruiseProcess" aei:id="5267441382"/>
13      <bpel:partnerLink partnerLinkType="ns:CarrentalProcessPLT" name
        ="requestprice" partnerRole="Provider" aei:interface="
        carrentalProcess:CarrentalProcess" aei:id="6160963606"/>
14      <bpel:partnerLink partnerLinkType="ns:
        ActivityproviderProcessPLT" name="listactivities"
        partnerRole="Provider" aei:interface="
        activityproviderProcess:ActivityproviderProcess" aei:id
        ="5105280568"/>
15      <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
        ="listactivities_temp" partnerRole="Provider" aei:interface
        ="travelerProcess:TravelerProcess" aei:id="9206277718"/>
16      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
        name="setselectedactivities" partnerRole="Provider" aei:
        interface="travelagencyProcess:TravelagencyProcess" aei:id
        ="7448427495"/>
17      <bpel:partnerLink partnerLinkType="ns:
        ActivityproviderProcessPLT" name="requestprice" partnerRole
        ="Provider" aei:interface="activityproviderProcess:
        ActivityproviderProcess" aei:id="8771548067"/>
18      <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
        ="getavaliabletripoptions" partnerRole="Provider" aei:
        interface="travelerProcess:TravelerProcess" aei:id
        ="7167471986"/>
19      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
        name="selectairline" partnerRole="Provider" aei:interface="
        travelagencyProcess:TravelagencyProcess" aei:id
        ="6908201778"/>
20      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
        name="selecthotel" partnerRole="Provider" aei:interface="
        travelagencyProcess:TravelagencyProcess" aei:id
```

```
        ="1721862691"/>
21      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
        name="selectcruise" partnerRole="Provider" aei:interface="
        travelagencyProcess:TravelagencyProcess" aei:id
        ="3579635831"/>
22      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
        name="selectcarrental" partnerRole="Provider" aei:interface
        ="travelagencyProcess:TravelagencyProcess" aei:id
        ="0826709246"/>
23      <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
        ="getconfirmedplan" partnerRole="Provider" aei:interface="
        travelerProcess:TravelerProcess" aei:id="6856152159"/>
24    </bpel:partnerLinks>
25    <bpel:variables>
26      <bpel:variable name="intelem" element="travelag:
        travelagencyProcessRequest"/>
27      <bpel:variable name="stringelem" element="travelag:
        travelagencyProcessRequest"/>
28      <bpel:variable name="cruiseconfirmation" element="travelag:
        travelagencyProcessRequest"/>
29      <bpel:variable name="carrentalconfirmation" element="travelag:
        travelagencyProcessRequest"/>
30      <bpel:variable name="activityconfirmation" element="travelag:
        travelagencyProcessRequest"/>
31    </bpel:variables>
32    <bpel:flow aei:id="7852398760">
33      <bpel:sequence aei:id="4753089694">
34        <bpel:invoke operation="querytrip" partnerlink="querytrip"
            name="travelagency" aei:id="1470070611" variable="
            startdate"/>
35        <bpel:flow aei:forkJoin="yes" aei:id="2091793067">
36          <vxbpel:VariationPoint name="booking">
37            <vxbpel:Variants>
38              <vxbpel:Variant name="airline">
39                <vxbpel:VPBpelCode>
40                  <bpel:invoke operation="requestprice" partnerlink="
                      requestprice" name="airline" aei:id="7704360219"
```

```
                    variable="startdate"/>
41              </vxbpel:VPBpelCode>
42            </vxbpel:Variant>
43            <vxbpel:Variant name="hotel">
44              <vxbpel:VPBpelCode>
45                <bpel:invoke operation="requestprice" partnerlink="
                      requestprice" name="hotel" aei:id="2512554302"
                      variable="startdate"/>
46              </vxbpel:VPBpelCode>
47            </vxbpel:Variant>
48          </vxbpel:Variants>
49        </vxbpel:VariationPoint>
50        <vxbpel:VariationPoint name="facilities">
51          <vxbpel:Variants>
52            <vxbpel:Variant name="cruise">
53              <vxbpel:VPBpelCode>
54                <bpel:invoke operation="requestprice" partnerlink="
                      requestprice" name="cruise" aei:id="8397945070"
                      variable="packageid"/>
55              </vxbpel:VPBpelCode>
56            </vxbpel:Variant>
57            <vxbpel:Variant name="carrental">
58              <vxbpel:VPBpelCode>
59                <bpel:invoke operation="requestprice" partnerlink="
                      requestprice" name="carrental" aei:id
                      ="9625798902" variable="carmodel"/>
60              </vxbpel:VPBpelCode>
61            </vxbpel:Variant>
62            <vxbpel:Variant name="activities">
63              <vxbpel:VPBpelCode>
64                <bpel:sequence aei:id="1704764723">
65                  <bpel:invoke operation="listactivities"
                        partnerlink="listactivities" name="
                        activityprovider" aei:id="6917304196" variable
                        ="place"/>
66                  <bpel:flow aei:forkJoin="yes" aei:id="4355232778"
                        >
```

```
67                    <bpel:invoke operation="listactivities_temp"
                          partnerlink="listactivities_temp" name="
                          traveler" aei:id="2257857948"/>
68                  </bpel:flow>
69                  <bpel:invoke operation="setselectedactivities"
                        partnerlink="setselectedactivities" name="
                        travelagency" aei:id="5558653459" variable="
                        selectedlist"/>
70                  <bpel:invoke operation="requestprice" partnerlink
                        ="requestprice" name="activityprovider" aei:id
                        ="0485344646" variable="selectedlist"/>
71                </bpel:sequence>
72              </vxbpel:VPBpelCode>
73            </vxbpel:Variant>
74          </vxbpel:Variants>
75        </vxbpel:VariationPoint>
76      </bpel:flow>
77      <bpel:invoke operation="getavaliabletripoptions" partnerlink
          ="getavaliabletripoptions" name="traveler" aei:id
          ="0509001798" variable="travelerID"/>
78      <bpel:receive operation="getavaliabletripoptions" partnerlink
          ="getavaliabletripoptions" name="traveler" aei:id
          ="6344151910" variable="travelerID"/>
79      <vxbpel:VariationPoint name="booking">
80        <vxbpel:Variants>
81          <vxbpel:Variant name="airline">
82            <vxbpel:VPBpelCode>
83              <bpel:sequence aei:id="2654629829">
84                <bpel:invoke outputVariable="intelem" operation="
                      selectairline" partnerlink="selectairline" name
                      ="travelagency" aei:id="6292378011" variable="
                      airlineID"/>
85              </bpel:sequence>
86            </vxbpel:VPBpelCode>
87          </vxbpel:Variant>
88          <vxbpel:Variant name="hotel">
89            <vxbpel:VPBpelCode>
```

```
90              <bpel:sequence aei:id="0623588291">
91                <bpel:invoke outputVariable="stringelem" operation
                      ="selecthotel" partnerlink="selecthotel" name="
                      travelagency" aei:id="5541067870" variable="
                      hotelID"/>
92              </bpel:sequence>
93            </vxbpel:VPBpelCode>
94          </vxbpel:Variant>
95        </vxbpel:Variants>
96      </vxbpel:VariationPoint>
97      <vxbpel:VariationPoint name="facilities">
98        <vxbpel:Variants>
99          <vxbpel:Variant name="cruise">
100           <vxbpel:VPBpelCode>
101             <bpel:sequence aei:id="2258916978">
102               <bpel:invoke outputVariable="cruiseconfirmation"
                      operation="selectcruise" partnerlink="
                      selectcruise" name="travelagency" aei:id
                      ="4593472620" variable="packageid"/>
103             </bpel:sequence>
104           </vxbpel:VPBpelCode>
105         </vxbpel:Variant>
106         <vxbpel:Variant name="carrental">
107           <vxbpel:VPBpelCode>
108             <bpel:sequence aei:id="0671685516">
109               <bpel:invoke outputVariable="carrentalconfirmation"
                       operation="selectcarrental" partnerlink="
                      selectcarrental" name="travelagency" aei:id
                      ="6741086100" variable="carmodel"/>
110             </bpel:sequence>
111           </vxbpel:VPBpelCode>
112         </vxbpel:Variant>
113         <vxbpel:Variant name="activities">
114           <vxbpel:VPBpelCode>
115             <bpel:sequence aei:id="5479513147">
116               <bpel:invoke outputVariable="activityconfirmation"
                      operation="setselectedactivities" partnerlink="
```

```
                    setselectedactivities" name="travelagency" aei:
                    id="2085641222" variable="activitylist"/>
117                 <bpel:invoke operation="getconfirmedplan"
                    partnerlink="getconfirmedplan" name="traveler"
                    aei:id="0403061426"/>
118               </bpel:sequence>
119             </vxbpel:VPBpelCode>
120           </vxbpel:Variant>
121         </vxbpel:Variants>
122       </vxbpel:VariationPoint>
123     </bpel:sequence>
124   </bpel:flow>
125   <vxbpel:ConfigurableVariationPoints>
126     <vxbpel:ConfigurableVariationPoint defaultVariant="regular" id
        ="itinerary">
127       <vxbpel:Name>itinerary</vxbpel:Name>
128       <vxbpel:Rationale>"it is realized by booking and facilities
          variation points"</vxbpel:Rationale>
129       <vxbpel:Variants>
130         <vxbpel:Variant name="vacationpackage">
131           <vxbpel:VariantInfo>...</vxbpel:VariantInfo>
132           <vxbpel:RequiredConfiguration>
133             <vxbpel:VPChoices>
134               <vxbpel:VPChoice vpname="facilities" variant="cruise
                  "/>
135               <vxbpel:VPChoice vpname="facilities" variant="
                  carrental"/>
136               <vxbpel:VPChoice vpname="facilities" variant="
                  activities"/>
137               <vxbpel:VPChoice vpname="booking" variant="hotel"/>
138               <vxbpel:VPChoice vpname="booking" variant="airline"/>
139             </vxbpel:VPChoices>
140           </vxbpel:RequiredConfiguration>
141         </vxbpel:Variant>
142         <vxbpel:Variant name="regular">
143           <vxbpel:VariantInfo>...</vxbpel:VariantInfo>
144           <vxbpel:RequiredConfiguration>
```

```
145                <vxbpel:VPChoices>
146                  <vxbpel:VPChoice vpname="facilities" variant="cruise
                         "/>
147                  <vxbpel:VPChoice vpname="facilities" variant="
                         carrental"/>
148                  <vxbpel:VPChoice vpname="facilities" variant="
                         activities"/>
149                  <vxbpel:VPChoice vpname="booking" variant="hotel"/>
150                  <vxbpel:VPChoice vpname="booking" variant="airline"/>
151                </vxbpel:VPChoices>
152              </vxbpel:RequiredConfiguration>
153            </vxbpel:Variant>
154          </vxbpel:Variants>
155        </vxbpel:ConfigurableVariationPoint>
156      </vxbpel:ConfigurableVariationPoints>
157  </bpel:process>
```

```
1  <bpel:process xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
       travelitinerary_planitinerary" ext1:linksAreTransitions="yes"
       ext:disableSelectionFailure="yes" xmlns:ext1="http://www.
       activebpel.org/2009/06/bpel/extension/links" suppressJoinFailure
       ="yes" xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
       ext:createTargetXPath="yes" xmlns:vxbpel="http://www.turkselma.
       com/vxbpel" xmlns:ext="http://www.activebpel.org/2006/09/bpel/
       extension/query_handling" xmlns:abx="http://www.activebpel.org/
       bpel/extension" targetNamespace="http://verificationprocess" aei
       :editStyle="BPMN" xmlns:bpel="http://docs.oasis-open.org/wsbpel
       /2.0/process/executable" xmlns:aei="http://www.activebpel.org
       /2009/02/bpel/extension/ignorable">
2    <bpel:extensions>
3      <bpel:extension mustUnderstand="no" namespace="http://www.
           activebpel.org/2009/02/bpel/extension/ignorable"/>
4      <bpel:extension mustUnderstand="no" namespace="http://www.omg.
           org/spec/BPMN/20100524/DI"/>
5      <bpel:extension mustUnderstand="yes" namespace="http://www.
           activebpel.org/2006/09/bpel/extension/query_handling"/>
```

```
6     <bpel:extension mustUnderstand="yes" namespace="http://www.
          activebpel.org/2009/06/bpel/extension/links"/>
7   </bpel:extensions>
8   <bpel:partnerLinks>
9     <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
          name="querytrip" partnerRole="Provider" aei:interface="
          travelagencyProcess:TravelagencyProcess" aei:id
          ="4640414432"/>
10    <bpel:partnerLink partnerLinkType="ns:AirlineProcessPLT" name="
          requestprice" partnerRole="Provider" aei:interface="
          airlineProcess:AirlineProcess" aei:id="2431418264"/>
11    <bpel:partnerLink partnerLinkType="ns:HotelProcessPLT" name="
          requestprice" partnerRole="Provider" aei:interface="
          hotelProcess:HotelProcess" aei:id="1395626079"/>
12    <bpel:partnerLink partnerLinkType="ns:CruiseProcessPLT" name="
          requestprice" partnerRole="Provider" aei:interface="
          cruiseProcess:CruiseProcess" aei:id="5267441382"/>
13    <bpel:partnerLink partnerLinkType="ns:CarrentalProcessPLT" name
          ="requestprice" partnerRole="Provider" aei:interface="
          carrentalProcess:CarrentalProcess" aei:id="6160963606"/>
14    <bpel:partnerLink partnerLinkType="ns:
          ActivityproviderProcessPLT" name="listactivities"
          partnerRole="Provider" aei:interface="
          activityproviderProcess:ActivityproviderProcess" aei:id
          ="5105280568"/>
15    <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
          ="listactivities_temp" partnerRole="Provider" aei:interface
          ="travelerProcess:TravelerProcess" aei:id="9206277718"/>
16    <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
          name="setselectedactivities" partnerRole="Provider" aei:
          interface="travelagencyProcess:TravelagencyProcess" aei:id
          ="7448427495"/>
17    <bpel:partnerLink partnerLinkType="ns:
          ActivityproviderProcessPLT" name="requestprice" partnerRole
          ="Provider" aei:interface="activityproviderProcess:
          ActivityproviderProcess" aei:id="8771548067"/>
```

```
18      <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
            ="getavaliabletripoptions" partnerRole="Provider" aei:
            interface="travelerProcess:TravelerProcess" aei:id
            ="7167471986"/>
19      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
            name="selectairline" partnerRole="Provider" aei:interface="
            travelagencyProcess:TravelagencyProcess" aei:id
            ="6908201778"/>
20      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
            name="selecthotel" partnerRole="Provider" aei:interface="
            travelagencyProcess:TravelagencyProcess" aei:id
            ="1721862691"/>
21      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
            name="selectcruise" partnerRole="Provider" aei:interface="
            travelagencyProcess:TravelagencyProcess" aei:id
            ="3579635831"/>
22      <bpel:partnerLink partnerLinkType="ns:TravelagencyProcessPLT"
            name="selectcarrental" partnerRole="Provider" aei:interface
            ="travelagencyProcess:TravelagencyProcess" aei:id
            ="0826709246"/>
23      <bpel:partnerLink partnerLinkType="ns:TravelerProcessPLT" name
            ="getconfirmedplan" partnerRole="Provider" aei:interface="
            travelerProcess:TravelerProcess" aei:id="6856152159"/>
24    </bpel:partnerLinks>
25    <bpel:variables>
26      <bpel:variable name="intelem" element="travelag:
            travelagencyProcessRequest"/>
27      <bpel:variable name="stringelem" element="travelag:
            travelagencyProcessRequest"/>
28      <bpel:variable name="cruiseconfirmation" element="travelag:
            travelagencyProcessRequest"/>
29      <bpel:variable name="carrentalconfirmation" element="travelag:
            travelagencyProcessRequest"/>
30      <bpel:variable name="activityconfirmation" element="travelag:
            travelagencyProcessRequest"/>
31    </bpel:variables>
32    <bpel:flow aei:id="7852398760">
```

```
33      <bpel:sequence aei:id="4753089694">
34        <bpel:invoke operation="querytrip" partnerlink="querytrip"
             name="travelagency" aei:id="1470070611" variable="
             startdate"/>
35        <bpel:flow aei:forkJoin="yes" aei:id="2091793067">
36          <bpel:invoke operation="requestprice" partnerlink="
               requestprice" name="airline" aei:id="7704360219"
               variable="startdate"/>
37          <bpel:sequence aei:id="1704764723">
38              <bpel:invoke operation="listactivities" partnerlink="
                   listactivities" name="activityprovider" aei:id
                   ="6917304196" variable="place"/>
39              <bpel:flow aei:forkJoin="yes" aei:id="4355232778">
40                  <bpel:invoke operation="listactivities_temp"
                       partnerlink="listactivities_temp" name="traveler
                       " aei:id="2257857948"/>
41              </bpel:flow>
42              <bpel:invoke operation="setselectedactivities"
                   partnerlink="setselectedactivities" name="
                   travelagency" aei:id="5558653459" variable="
                   selectedlist"/>
43              <bpel:invoke operation="requestprice" partnerlink="
                   requestprice" name="activityprovider" aei:id
                   ="0485344646" variable="selectedlist"/>
44          </bpel:sequence>
45        </bpel:flow>
46        <bpel:invoke operation="getavaliabletripoptions" partnerlink
             ="getavaliabletripoptions" name="traveler" aei:id
             ="0509001798" variable="travelerID"/>
47        <bpel:receive operation="getavaliabletripoptions" partnerlink
             ="getavaliabletripoptions" name="traveler" aei:id
             ="6344151910" variable="travelerID"/>
48        <bpel:sequence aei:id="2654629829">
49          <bpel:invoke outputVariable="intelem" operation="
               selectairline" partnerlink="selectairline" name="
               travelagency" aei:id="6292378011" variable="airlineID"/>
50        </bpel:sequence>
```

```
51        <bpel:sequence aei:id="5479513147">
52          <bpel:invoke outputVariable="activityconfirmation"
                operation="setselectedactivities" partnerlink="
                setselectedactivities" name="travelagency" aei:id
                ="2085641222" variable="activitylist"/>
53          <bpel:invoke operation="getconfirmedplan" partnerlink="
                getconfirmedplan" name="traveler" aei:id="0403061426"/>
54        </bpel:sequence>
55      </bpel:sequence>
56    </bpel:flow>
57  </bpel:process>
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Süloğlu, Selma
Nationality: Turkish (TC)
Date and Place of Birth: 30.09.1980, Eskişehir
Marital Status: Single
Phone: 0 312 297 61 57
Fax: 0 312 297 61 54

EDUCATION

Degree	Institution	Year of Graduation
M.S.	Natural and Applied Sciences	2006
B.S.	Social Sciences	2003
High School	Süleyman Çakır High School	1998
High School	Kılıçoğlu Anatolian High School	1997

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2010-	SOSOFT Information Technologies	Co-Founder - Project Manager
2007-2011	Middle East Technical University	Research and Teaching Assistant
2006-2007	Middle East Technical University	METU-ISTEC Project Personel
2005-2006	Datasel Information Systems	Software Engineer
2004	Mobilsoft, Meteksan System	Stajer Software Engineer

PUBLICATIONS

International Conference Publications

- Selma Suloglu, Cengiz Togay, Ali H. Dogru, "Managing Variability In Service Composition with Axiomatic Design", In 18th International Conference on Society for Design and Process Science (SDPS 2013), October 27-31, 2013, To be published.

- Selma Suloglu, Riza Aktunc, Mustafa Yucefaydalı, " Verification of Variable Service Orchestrations using Model Checking", In 2nd International Workshop on Quality Assurance for Service-Based Applications (QASBA), July 15, 2013, Lugano, Switzerland.

- Selma Suloglu, Bedir Tekinerdogan, Ali Dogru, " XChor: Chreography Language For Integration of Variable Orchestration Languages", In 3rd International Symposium on Business Modeling and Software Design, Noordwijkerhout, Netherlands, July 8-10, 2013.

- Eren Akbıyık, Selma Süloğlu, Cengiz Togay and Ali H. Doğru, "Service Oriented Systems Design Through Process Decomposition", The Eleventh World Conference on Integrated Design and Process Technology, pp: 332-338, Tauic hung, Taiwan, June 1-6, 2008.

- Semih Çetin, N. İlker Altıntaş, Halit Oğuztüzün, Ali H. Doğru, Özgür Tüfekçi and Selma Süloğlu, "A Mashup-Based Strategy for Migration to Service-Oriented Computing", IEEE International Conference on Pervasive Services, İstanbul, Turkey, pp: 169-172, July 15-20, 2007.

- Semih Çetin, N. İlker Altıntaş, Halit Oğuztüzün, Ali H. Doğru, Özgür Tüfekçi and Selma Süloğlu, "Legacy Migration to Service-Oriented Computing with Mashups" International Conference on Software Engineering Advances, August 2007.