

A KNOWLEDGE BASED PRODUCT LINE FOR SEMANTIC MODELING OF
WEB SERVICE FAMILIES

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

BY

UMUT ORHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2009

Approval of the thesis

**“A KNOWLEDGE BASED PRODUCT LINE FOR SEMANTIC
MODELING OF WEB SERVICE FAMILIES”**

submitted by **Umut Orhan** in partial fulfillment of the requirements for the degree
of **Master of Science in Computer Engineering, Middle East Technical
University** by,

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

Prof. Dr. Müslim Bozyigit _____
Head of Department, **Computer Engineering**

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

Examining Committee Members:

Assoc. Prof. Dr. Ferda Nur Alpaslan _____
Department of Computer Engineering, METU

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

Assoc. Prof. Dr. Nihan Kesim Çiçekli _____
Department of Computer Engineering, METU

Asst. Prof. Dr. Pınar Şenkul _____
Department of Computer Engineering, METU

Yıldıray Kabak _____
SRDC Ltd.

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 : Umut Orhan

Signature :

ABSTRACT

A KNOWLEDGE BASED PRODUCT LINE FOR SEMANTIC MODELING OF WEB SERVICE FAMILIES

Orhan, Umut

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali H. Doğru

January 2009, 118 pages

Some mechanisms to enable an effective transition from domain models to web service descriptions are developed. The introduced domain modeling support provides verification and correction on the customization part. An automated mapping mechanism from the domain model to web service ontologies is also developed. The proposed approach is based on Feature-Oriented Domain Analysis (FODA), Semantic Web technologies and ebXML Business Process Specification Schema (ebBP).

Major contributions of this work are the conceptualizations of a feature model for web services and a novel approach for knowledge-based elicitation of domain-specific outcomes in order to allow designing and deploying services better aligned with dynamically changing business goals, stakeholders' concerns and end-users' viewpoints. The main idea behind enabling a knowledge-based approach is to pursue automation and intelligence on reflecting business requirements into service descriptions via model transformations and automated reasoning. The proposed reference variability model encloses the domain-specific knowledge and is formalized by using Web Ontology Language (OWL). Adding formal semantics to feature models allows us to perform automated analysis over them such as the verification of model customizations through exploiting rule-based automated reasoners.

This research was motivated due to the needs for achieving productivity gains, maintainability and better alignment of business requirements with technical capabilities in engineer-

ing service-oriented applications and systems.

Keywords: Feature-Oriented Domain Analysis, Service-Oriented Computing, Software Product Lines, Semantic Web, ebBP, OWL, OWL-S, SWRL, JESS

ÖZ

BİLGİ TABANLI ANLAMSAL AĞ SERVİS AİLESİ MODELİ ÜRETİM BANDI

Orhan, Umut

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ali H. Doğru

Ocak 2009, 118 sayfa

Alan modellerinden, ağ servis tanımlarına geçiş sağlayan etkin mekanizmalar geliştirilmiştir. Tanıtılan alan modelleme desteğinin özelleştirme kısmında, doğrulama ve düzeltme yetenekleri bulunmaktadır. Ayrıca, alan modelinden ağ servis ontolojilerine eşleme sağlayan otomatikleştirilmiş bir mekanizma geliştirilmiştir. Önerilen yaklaşım özellik yönelimli alan analizi (FODA), anlamsal ağ ve ebXML iş süreç belirleme şeması (ebBP) teknolojilerini temel almaktadır.

Bu çalışma ile ortaya konulan temel kazanımlar; ağ servisleri için hazırlanan bir özellik modeli ile alana özgü ürünlerden bilgi temelli çıkarımlar yapılmasını ve böylece dinamik iş hedeflerine, pay sahipleri ve son kullanıcı görüşlerine daha iyi uyum gösteren servislerin tasarlanması ve konuşlandırılmasını sağlayan bir yöntemdir. Bilgi temelli yaklaşımın arkasında yatan temel düşünce model dönüşümleri ve çıkarımlar yaparak iş gereksinimlerinin servis tanımlamalarına yansıtılması işlemine otomasyon kazandırmaktır. İleri sürülen referans değişkenlik modeli alana özgü bilgiyi içermekte olup ağ ontoloji dili (OWL) ile biçimlendirilmiştir. Özellik modellerine kazandırılan bu anlamsal yapılar sayesinde model uyarlamalarının doğrulanması gibi otomatikleşmiş analizlerin kural tabanlı çıkarım motorları ile gerçekleştirilmesi sağlanmıştır.

Bu araştırma çalışmasının çıkış noktalarını servis yönelimli uygulama ve sistemlerin mühendisliğinde ihtiyaç duyulan üretkenlik kazanımları ve devamlılığın sağlanması ile iş

gereksinimleri ve teknik yeterlilikler arasındaki uyumun iyileştirilmesi konuları oluşturmaktadır.

Anahtar Kelimeler: Özellik Yönelimli Alan Analizi, Servis Yönelimli Hesaplama, Yazılım Üretim Bandları, Anlamsal Ağ, ebBP, OWL, OWL-S, SWRL, JESS

ACKNOWLEDGMENTS

First of all, I am honored to express my sincere gratitude and appreciation to Assoc. Prof. Dr. Ali H. Doğru for his encouragement and support throughout this study.

I would also like to convey thanks to jury members and Assoc. Prof. Dr. Halit Oğuztüzün for their valuable comments on this thesis.

I would like to express my thanks to my friends Gökdeniz Karadağ, Mehmet Olduz, Mustafa Yüksel and Özgür Gülderen for their help and support during my graduate study.

I am deeply grateful to my family for their love and support. Without them, this work could not have been completed.

Finally, I am also grateful to my dear friend Simge Çetintoprak for her love, continued motivating support and cheerful presence.

To my family...

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGMENTS	viii
DEDICATON	ix
TABLE OF CONTENTS	x
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
LIST OF ABBREVIATIONS	xvii
CHAPTER	

1 INTRODUCTION	1
2 BACKGROUND ON ENABLING TECHNOLOGIES AND STANDARDS	6
2.1 Product Line Engineering and Software Product Lines	6
2.1.1 Software Product Line Process	9
2.1.2 Domain Engineering	10
2.1.3 Application Engineering	13
2.1.4 Software Factory Automation (SFA)	14
2.2 Service-Oriented Architecture (SOA)	17
2.2.1 Web Services	20
2.2.2 Service Composition	21
2.2.3 ebXML Business Process Specification Schema	23
2.2.4 Semantic Interoperability	25
2.2.5 Historical Origins and Motivating Needs Behind SOA	26
2.3 Semantic Web Technologies	27
2.3.1 Resource Description Framework (RDF)	29

2.3.2	Web Ontology Language (OWL)	31
2.3.3	Description Logics	33
2.3.4	Protégé Ontology Editor and Knowledge Acquisition System	34
2.3.5	OWL-S: Semantic Markup for Web Services	35
2.3.6	Semantic Web Rule Language (SWRL) and JESS	37
3	ENHANCING DOMAIN KNOWLEDGE BASE WITH BUSINESS PROCESS DEFINITIONS	40
3.1	Core Components of the ebXML Business Process Specification Schema	41
3.1.1	Business Collaborations and Choreography	41
3.1.2	Business Transactions, Transaction Activities and Business Docu- ment Flow	43
3.1.3	Business Signals and Exceptions	46
3.2	The ebBP Editor	46
3.2.1	Introduction	46
3.2.2	Overview of the ebBP Editor Components	48
3.3	Mapping Business Collaborations to Web Service Process Models	53
3.3.1	Motivation Behind the Transformation Method	53
3.3.2	ebBP to OWL-S Mapping	55
3.3.3	Limitations of the Transformation Method	62
4	ADDING FORMAL SEMANTICS AND REASONING SUPPORT TO FEA- TURE MODELS	63
4.1	Feature Model Ontology	64
4.2	Feature Model Editor and Reasoner	68
4.2.1	System Design	69
4.2.2	User Guide	70
4.2.3	Verification and Correction	75
4.2.4	Performance Evaluation	78
5	EXPLOITING SEMANTICALLY ENRICHED FEATURE MODELS FOR SER- VICE ONTOLOGY DEVELOPMENT	79
5.1	A Variability Modeling Approach for Web Service Semantics	80
5.2	An Example Walkthrough with the GENoDL	84
6	RELATED WORK	87

7	CONCLUSION AND FUTURE WORK	89
7.1	Conducted Work	89
7.2	Concluding Remarks	90
7.3	Future Work	90
	REFERENCES	92
A	FEATURE MODEL ONTOLOGY	100
B	SCHEMA DEFINITIONS OF THE CORE EBBP COMPONENTS	111

LIST OF TABLES

TABLES

Table 2.1	Comparison between software engineering and domain engineering. Adapted from <i>Czarnecki, "Generative Programming", 1998</i> [16]	11
Table 2.2	Terminology of domain-specific kits	16
Table 3.1	Business Transaction to Simple Process	58
Table 3.2	DocumentEnvelope to Input (or Output)	58
Table 3.3	Business Transaction Activity to Atomic Process	59
Table 3.4	Complex Business Transaction Activity to Composite Process	60
Table 3.5	Business Collaboration to Service	60
Table 3.6	Choreography to Service Process Model	62
Table 5.1	Features to OWL-S parameters	83

LIST OF FIGURES

FIGURES

Figure 1.1	Product line approach for semantic modeling of web service families . . .	4
Figure 2.1	Cost analysis of a software project with or without SPL approach . . .	8
Figure 2.2	An overview of SPL's two-lifecycle process model. Adapted from <i>van der Linden et al "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering", 2007 [75]</i>	9
Figure 2.3	A common visual notation for some relationships among features . . .	12
Figure 2.4	A sample and abbreviated feature diagram of the concept car	13
Figure 2.5	A sample application engineering process described by SEI	14
Figure 2.6	Overview of the software factory automation approach. Adapted from <i>Altintas, "Feature Based Software Asset Modeling with Domain Specific Kits", 2007 [2]</i>	15
Figure 2.7	SOA actors in action	18
Figure 2.8	Web services technology stack	20
Figure 2.9	Service development life-cycle hierarchy	22
Figure 2.10	Representation of the "Drop Ship" multi-party collaboration with BPMN and the definition of a QueryResponse business transaction in ebBP format	25
Figure 2.11	An example RDF statements which means that John Doe is the creator of the resource http://www.metu.edu.tr/ John. Property "creator" refers to Dublin Core Definition Standard. RDF statement is given in two different representations; directed labeled graph and XML notation.	30
Figure 2.12	A screenshot from the OWL Classes view of the Protégé Ontology Editor	35
Figure 2.13	Upper ontology of services	36

Figure 3.1	A screenshot of the ebBP Editor	48
Figure 3.2	XmlStylist - Main Screen	49
Figure 3.3	XmlStylist - Select Root Dialog	50
Figure 3.4	XmlStylist warns the domain expert about an invalid Business Collaboration instance	52
Figure 3.5	Graphical Components of the ebBP Editor	53
Figure 3.6	Bridging the gap between domain and application engineering in developing service-oriented system	55
Figure 3.7	Overview of the mapping specification	56
Figure 4.1	Classes and properties of the feature model ontology	65
Figure 4.2	Classes of the feature model ontology with encapsulated SWRL ontology	67
Figure 4.3	Class diagram of the GENoDL	69
Figure 4.4	A screenshot from the GENoDL	70
Figure 4.5	Menu items of the GENoDL	71
Figure 4.6	The toolbar of the Feature Model Editor	71
Figure 4.7	Inserting a new feature to the model	72
Figure 4.8	A feature can be edited or deleted by double clicking on it	72
Figure 4.9	Popup menu for editing properties of a feature	73
Figure 4.10	Popup menu for editing feature types	74
Figure 4.11	Integrating knowledge base with reasoning engine through Protégé-SWRL adapter	76
Figure 4.12	An inferred correction for the inconsistent feature model	77
Figure 4.13	Performance Analysis	78
Figure 5.1	A reference variability model for semantic modeling of web service families	83
Figure 5.2	ebBP representation for BC-ID-DemandSurrenderOfDriverLicense	85
Figure 5.3	Service variability model is customized service feature model with the information extracted from BC-ID-DemandSurrenderOfDriverLicense.	86
Figure 5.4	A part of the BC-ID-DemandSurrenderOfDriverLicense Service Ontology	86
Figure B.1	Details of the Document Envelope structure	111
Figure B.2	ebBP definition for logical Business Documents	112

Figure B.3 High level view of the Business Transaction 113

Figure B.4 Graphical view of the schema of the Business Transaction Activity . . . 114

Figure B.5 Schema definition of the Complex Business Transaction Activity 115

Figure B.6 Model view of the Receipt Acknowledgement signal 116

Figure B.7 Schema of the exception elements found in ebBP documents 117

Figure B.8 Schema definition of the Business Collaboration 118

LIST OF ABBREVIATIONS

AML	Asset Modeling Language	HTML	HyperText Markup Language
AMM	Asset Meta Model	HTTP	Hyper Text Transfer Protocol
BPEL	Business Process Execution Language	JESS	The Rule Engine for the Java Platform
BPM	Business Process Management	OASIS	Organization for the Advancement of Structured Information Standards
BPMN	Business Process Modeling Notation	OWL	Web Ontology Language
COM	Component Object Model	PLC	Programmable Logic Controller
CORBA	Common Object Request Broker Architecture	REST	Representational State Transfer
DL	Description Logics	PSTN	Public Switched Telephone Network
DSA	Domain-Specific Artifact	RDF	Resource Description Framework
DSAT	Domain-Specific Artifact Type	RDFS	Resource Description Framework Schema
DSE	Domain-Specific Engine	SEI	Software Engineering Institute
DSK	Domain-Specific Kit	SFA	Software Factory Automation
DSL	Domain-Specific Language	SOA	Service-Oriented Architecture
DST	Domain-Specific Toolset	SOAP	Simple Object Access Protocol
ebBP	ebXML Business Process Specification Schema	SPL	Software Product Line
ebXML	Electronic Business using XML	UDDI	Universal Description, Discovery and Integration
FODA	Feature-Oriented Domain Analysis	UML	Unified Modeling Language
FOL	First-Order Logic	URI	Uniform Resource Identifier
FPML	Fundamental Business Process Modeling Language	WSFL	Web Service Flow Language
GSM	Global System for Mobile Communications	W3C	The World Wide Web Consortium
GUI	Graphical User Interface	WS-CDL	Web Services Choreography Description Language

WSDL	Web Service Definition Language
XML	The Extensible Markup Language
XPath	The XML Path Language
XSD	XML Schema Definition
XSL	The Extensible Stylesheet Language Family
XSLT	XSL Transformations

CHAPTER 1

INTRODUCTION

In today's highly competitive and demanding digital world of business, organizations should be more agile, self-sustainable and responsive to the changes in order to guarantee their survival. They should be able to adapt to rapid changes and innovations while reducing integration and interoperability costs. Unless organizations produce a decrease in time to market for new innovations with low development and maintenance costs, it would not be possible for them to obtain a sustainable competitive edge in their business.

Rapid adaptation to changing business parameters and technical innovation necessitate boosting software assets reuse and achieve productivity gains in system and service development. For example, contemporary challenges of telecom service providers can be listed as follows [46];

- How to facilitate mass development of services with reduced costs
- How to simplify supply management and mass partnering
- How to ensure reuse of assets in the future

On the other hand, today's service-oriented system realizations are often a direct result of wrapping the underlying legacy business logic as web services. Developers usually make applications and services once provided at the local level available for further use by means of web services. This process leads web services to be developed from the ground up i.e. service descriptions and accompanying data models have been already developed in an application-specific manner before exposed as web services. In order to enable automated service discovery and invocation, web service descriptions are annotated with semantic constructs and service process models built on top of the existing web service interfaces in order to describe how software agents will interact with the services.

As a matter of fact, this development process cannot fully exploit service-orientation in terms of business agility because, web service interfaces were created without considering the corresponding business requirements or the way in which the services might be used. However, moving to a more dynamic and competitive business ecosystem requires thinking and working in the opposite direction.

Recently, Business Process Management (BPM) and Service-Oriented Computing (SOC) combination is being advocated as a possible solution for setting proper level of service abstractions and reaching the desired agility and responsiveness to changing business parameters. In general, BPM provides the required metadata to be directly consumed by SOC meanwhile SOC provides BPM with an agile IT framework where changes in business parameters can be reflected dynamically. In BPM-SOC convergence, services are identified and described in terms of business processes. Service-Oriented Modeling and Architecture (SOMA) [5] provides a possible process specification for developing services from business process models.

Unfortunately, in order to enable the anticipated BPM-SOC convergence the knowledge of the domain in which services will reside should be first transferred from domain experts to developers. This necessitates common means for shared vocabulary and understanding of the business requirements which can ideally be achieved by using ontologies. Being a semantic model, a service ontology describes capabilities and requirements of a web service in an unambiguous and machine-interpretable form. Hence, it is important to provide developers with a semantic model of a web service during implementation, because agreed meaning and intended use of terms can be less ambiguously realized through exploiting formal definitions of concepts, entities, relationships and attributes. Moreover, during web service discovery, composition, mediation and monitoring, it is possible to employ automated reasoning methods whenever service ontologies are enabled. Products can be more rapidly customized and deployed by enabling the *build-by-integration* paradigm which encourage matching product capabilities with existing services. Nevertheless, ontology creation is an error-prone and time-consuming task [67] and requires extensive logic programming background or use of dedicated tools which domain experts may not be familiar with.

Accordingly, the motivating needs behind this work can be summarized as follows;

- Bridging the gap between business requirements and accompanying service models [58] and reflecting changes in business parameters to service realizations more rapidly
- Localizing, representing and disseminating the domain knowledge in a common way

that domain experts can share the knowledge with developers more easily

- Increasing the level of engineering in service-oriented computing by managing the differences and similarities across multiple service domains, fostering reuse and applying generative methods

In this respect, we exploit the concept of service ontology as a bridge between business process models and service interface implementations. By localizing the domain knowledge in terms of service ontologies, we provide the service developer with the necessary requirements of the services to be developed. Moreover, we automate the service ontology creation to some extent. With our automated approach, changes in the business parameters can be more rapidly reflected to service realizations. In addition, we enable the Feature-Oriented Domain Analysis (FODA) [44] to manage differences and similarities across multiple service definitions. By this way, we produce not only a single service ontology but a set of related service ontologies. It is desirable to generate the service ontologies automatically especially when we consider the complex and time-consuming nature of the ontology creation tasks. On the other hand, a service ontology may enable automatic discovery, execution, composition and interoperation of available services. Instead of developing each service from scratch, existing and semantically matching services can be discovered from service registries. High level representation of our product line is shown in the left side of the Figure 1.1. We provide the domain experts with the necessary environment in order to define business process models and feature models. These defined domain engineering outcomes are then mapped into corresponding service definitions. Service interface development and dynamic discovery of services based on their resulting ontological classifications however, are not considered within the scope of this work.

Major contributions of this research can be listed as follows;

- We enabled the ebBP Editor [64] tool which had been previously developed within the scope of a research project funded by the European Commission. The domain expert can model business processes in Business Process Modeling Notation (BPMN) [11] and the tool then exports these models to accompanying ebBP definitions.
- We developed an OWL [55] ontology to represent feature models. We also developed a set of SWRL [72] rules as axioms holding for relations among features. Using this ontology definition lead us to perform automated reasoning operations over the feature models. The verification and correction of the feature model customizations are performed by enabling the JESS [39] which is a rule-based automated reasoner.

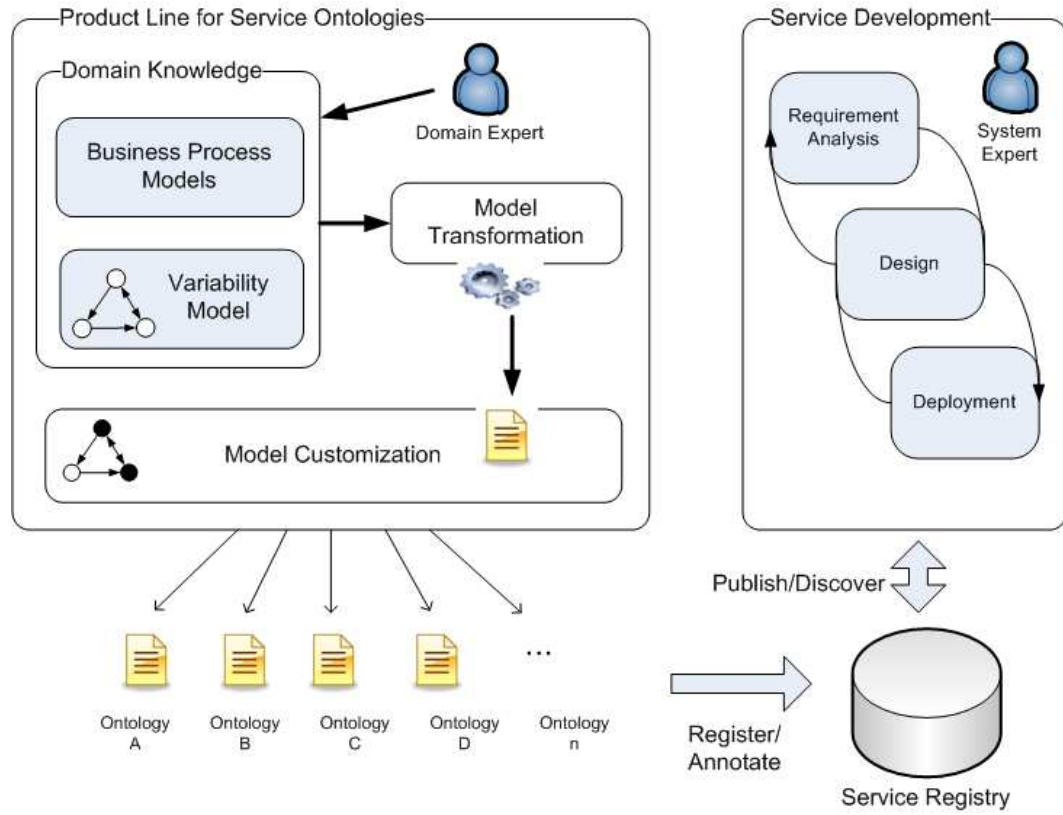


Figure 1.1: Product line approach for semantic modeling of web service families

- For the model customization part, we defined a sample feature model for web services. The variability points of this feature model were acquired from the ebBP and OWL-S specifications as well as from the previous studies.
- We developed a feature model editor in order to facilitate visual development of feature models. The tool imports and exports feature models formally defined by our feature model ontology.
- We defined model transformation rules from ebBP instances and feature models to accompanying service models. Generated service models are conceptualized as OWL-S [56] ontologies.

Main idea of enabling a semantic-based approach is to pursue automation and intelligence via reasoning on a domain-specific knowledge base about predefined concepts such as services. A service ontology is a conceptualization of the service specifications and is independent from the service interface implementation. A service based on ontology can be implemented by different service interfaces. Service ontology can be generated via service requirements auto-

matically extracted from domain engineering outcomes such as feature models and business workflows. Service requirements can be identified in two broad categories; functional and nonfunctional requirements such as quality of service (QoS). Basically, functional requirements identify what a system does and nonfunctional requirements describe how well those functions are accomplished. Several sub-categories like integrity, dependability, performance, security and safety can be listed under QoS requirements.

A standard business process specification notation such as ebBP defines (intra or inter) organizational business scenarios through specifying roles, collaborations, transactions and activities. These transactions are envisioned to be fulfilled by fine-grained web services. However, ebBP does not mandate any implementation technology specifically. Thus, in order to map ebBP constructs to service concepts we need further customizations beyond the ebBP especially for non-functional requirements of web services. In our approach, we employ feature modeling method to overcome these hindrances in a more business and end-user centric fashion. We provide a feature model for representing service concepts as features. Each feature is specified by the domain expert and is automatically mapped to its machine-readable semantic definition. Feature modeling paves the way for systematic reusability in defining services. A family of services can be easily defined through customizing features in a service feature model.

For automatic generation of a service ontology, the information related with the service such as input, output, precondition and effect is extracted from the business process and non-functional specifications such as QoS are acquired from the accompanying feature model. The collected information is then compiled into a service ontology which is ready to be realized as an implemented web service or be used when discovering existing services from a service registry. The generated service ontology conforms to OWL-S notation. In the scope of this research, verification of customizations and other reasoning activities are conducted using JESS rule engine and Protégé-OWL [60] is applied to build semantic web concepts.

This thesis is organized as follows; Chapter 2 summarizes the background on the enabling technologies and standards. In Chapter 3, the building blocks of the ebBP, the ebBP editor tool and the transformation rules from ebBP to OWL-S are given. We introduce our feature model ontology and feature model editor in Chapter 4. In Chapter 5, we combine feature-oriented domain analysis with our transformation method in order to generate families of service ontologies. In Chapter 6, the related work is presented on semantically enriched feature modeling and service ontology generation methods. Finally, Chapter 7 concludes this thesis and presents the future work.

CHAPTER 2

BACKGROUND ON ENABLING TECHNOLOGIES AND STANDARDS

2.1 Product Line Engineering and Software Product Lines

The popularity of the research on the software reuse did not show any symptoms of decline in academia and industry over years. Especially after the introduction of the Software Product Line (SPL) paradigm, the software industry witnessed the evolution of ad-hoc reuse practices into a more systematic approach. Keeping in mind the business and quality goals, SPL extends the boundaries of software reuse and exploits a broad spectrum of reusable assets spanning from program libraries and components to architecture blue-prints, test cases and even services. The key objective of this paradigm is to industrialize software development similar to other industries such as automotive and aviation in order to produce a number of relative systems that fulfill business requirements and end-user expectations in a prescribed way. The SPL paradigm promises to achieve productivity gains, reduce product development and maintenance costs, decrease time-to-market and increase product quality [13].

Major interest area of product line engineering is developing not just only one product but a family of related products in a specific domain. Consider a car entertainment system which is typically driven by the automobile vendors and the delivery company. Each client has its own distinct requirements than another has. Being the system vendor, an organization should deal with those variations and probably cannot afford to develop a unique system every time from scratch. Instead, the organization should reuse its previously implemented artifacts and existing assets as much as it is possible. This apparent industrial need leverages an important concern at the hearth of software engineering; *how to organize and manage the reuse in software development to reduce the development cost and time-to-market*. Another example of

product line engineering will be found in today's mobile phone business where in nearly every three weeks a new type of phone is introduced as a result of the incremental development over the previous ones. This picture leads on the fact that if an organization have a software product line which helps it to come up with new releases or new combinations of features (for example mobile phones with or without a digital camera) then the organization will increase its productivity. Again, achieving higher levels of productivity is one of the main objectives which product lines are often used for. In today's highly competitive and demanding digital world of business; organizations should be more agile, self-sustainable and responsive to the changes in order to guarantee their survival. Unless organizations release their high quality products quickly to market with low development and maintenance costs, it would not be possible for them to obtain and maintain a competitive edge in their business.

Main characteristics of product line engineering can be listed as follows;

- The primary reason behind using product lines is to make the product development activities cheaper, faster and better aligned with business needs based on the customer portfolio.
- Product line engineering is a strategic choice for a long term view and requires upfront investment and discipline.
- The scope of the domain should be well defined. It determines the what the product line is capable of producing, puts the focus on the business case and controls the investments on development.
- The domain should stay stable. If the domain is changing frequently then keep on track with those changes will be tedious and the upfront investment for domain engineering will not be enough or it will be wasted.
- Successful product line engineering needs organizational change, business process change and technology change [75]. Only excellence in technical issues will not protect product line architectures from failure if they are not effectively adopted by the organization.
- Products derived as the result of the SPL process pertain to a market strategy and application domain, share a common architecture and built by reusable components.

Basically, a software product line derives a reuse strategy that captures commonalities and manages variabilities among a set of products. Each product is targeted and customized

in order to address the needs and expectations of a specific customer in the customer portfolio. Even though the required productivity gains for development and maintaining issues can be achieved by the help of systematic reuse, establishing a precise and effective product line requires extra up-front investment for building reusable assets and organizational changes. A cost analysis of a software project with and without product line approach is placed comparatively in [80] (Figure 2.1). According to this work, a product line is best to be established when the domain can be supplied with at least three different products.

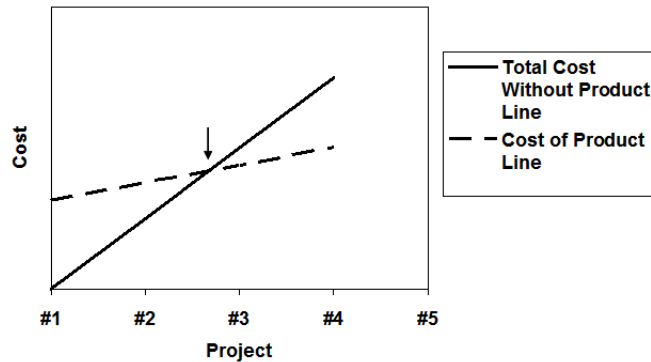


Figure 2.1: Cost analysis of a software project with or without SPL approach

A good indicator whether a specific domain needs a product line lies on the software development practices applied on that domain. When people have already started to build platforms that consist of common and various generic artifacts to foster reuse and prepare for potential variations, there is a good sign that they are actually developing a product line. However, without product line engineering, a single platform development approach for achieving high levels of software reuse addresses variability and customization issues of reusable assets in an ad hoc fashion instead of a systematic way. An important percentage of software development activities will be focused on platform development rather than product development and hence, actual products will not be completed in a timely manner. On the other hand, software product line engineering provides the frame for developing the platform for a specific domain. SPLE explicitly deals with variability concerns at all phases and brings variability management.

2.1.1 Software Product Line Process

In general, SPL practice is a two-life-cycle process that consists of domain and application engineering. An overview of this two-life-cycle process can be found at [75](Figure 2.2). The aim of domain engineering is to establish a mapping between problem space to solution space. It is responsible for constructing a domain model, establishing a reference architecture and implementation of reusable assets. Within the boundaries of the constructed domain

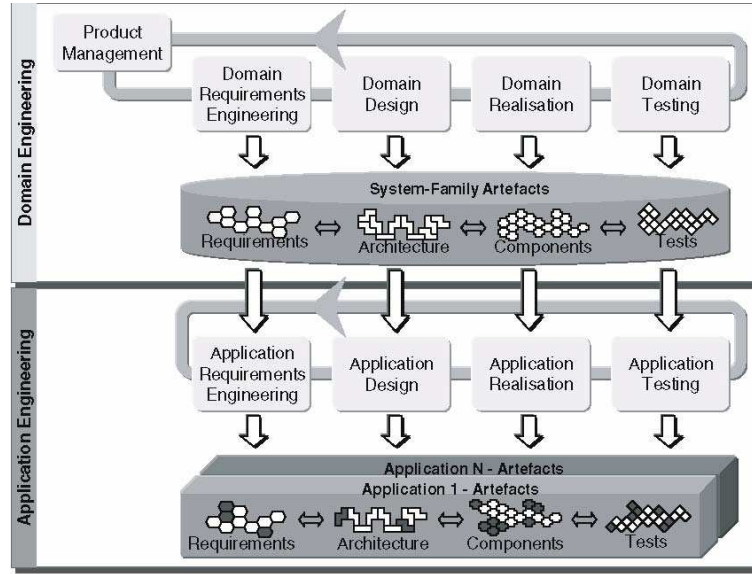


Figure 2.2: An overview of SPL's two-lifecycle process model. Adapted from *van der Linden et al "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering", 2007 [75]*

model, a reference architecture model is developed in order to realize features by mapping them to reusable assets. Reference architecture model will support more than one reference architecture specifying different connectors and interactions between architectural elements. A reusable asset is identified by the projection of relevant feature(s) to reference architecture. Thus, the reference architecture is an essential part of a Software Product Line (SPL) and as an outcome of the domain engineering, it is expected to establish a shared base among a range of products for reusing common software development assets, supporting variability and extension, and assembling both common and product-specific architectural components in an effective manner. The effectiveness of reference architecture is directly related with

how well the architectural components and their interactions are captured, integrated and managed through the development life-cycle.

Application engineering enables domain engineering outcomes and results in products that are ready to be delivered.

2.1.2 Domain Engineering

In the scope of the SPL context, domain engineering identifies, models, constructs, catalogs and disseminates a set of software artifacts that can be applied to existing and future software in a particular application domain. The emphasis is on developing artifacts for reuse. The aim of the domain engineering is to provide better means for delivering a family of products in a shorter time and at lower cost by first capturing the domain knowledge in the form of reusable assets and then reusing these assets in order to form the product. Domain engineering involves three main subprocess;

- *Domain Analysis:* In domain analysis, commonalities among products are identified and the scope of the product line is set through specifying variability points for particular products. The actors involved in domain analysis phase are stakeholders, end-users, domain experts and system analysts
- *Domain Design:* The reference architecture for the products in the domain is established in this phase.
- *Domain Implementation:* Development of reusable assets such as components, services and domain-specific languages takes place during domain implementation.

A comparison between conventional software engineering and domain engineering based on their outcomes is given in [16](Table 2.1).

Feature-Oriented Domain Analysis (FODA)

Feature-oriented domain analysis is a well-known domain analysis method which was first introduced by Kang et al in [44]. In FODA, variabilities and commonalities in the problem space are represented in terms of features. A feature can be considered as a product capability that is agreed by a consensus of stakeholders, end-users and engineers. One of the well-know methods for visualizing feature-oriented domain analysis is called feature modeling. A feature model is capable of representing all possible products in a product line. Typically, features are organized in a tree hierarchy in a feature model.

Table 2.1: Comparison between software engineering and domain engineering. Adapted from Czarnecki, "Generative Programming", 1998 [16]

Software Engineering	Domain Engineering
<i>Requirements Analysis</i> ➤ Requirements for one system	<i>Domain Analysis</i> ➤ Reusable requirements for a class of systems
<i>System Design</i> ➤ Design for one system	<i>Domain Design</i> ➤ Reusable design for a class of systems
<i>System Implementation</i> ➤ Implemented system	<i>Domain Implementation</i> ➤ Reusable components, infrastructures, and production process

Features are selected by considering the relationships among them in order to form a specific product. A feature model helps the domain expert to come up with a correct configuration of features that will result in a workable and usable product. Actually, feature modeling can be used in any level of the domain engineering; requirements engineering, designing the reference architecture or in the other levels close to programming.

A feature model provides graphical tree-like representation of the hierarchical organization among features within a concept. The root of the tree is named as the concept node and the others show different features of this concept. A simple feature model is capable of defining mandatory, optional, alternative, excludes, requires and OR relations among features. A common notation for visualizing some different feature relationships is given in Figure 2.3. Definitions for the relationships are introduced as follows;

- *Mandatory*: The feature must be included into the configuration of its parent concept instance.
- *Optional*: The feature may or may not be included into the configuration of its parent concept instance.
- *Alternative*: One instance from a set of features sharing a common parent concept can be included into its parent's configuration.
- *OR*: One or more features from a set of features can be included into the configuration of their shared parent concept instance.
- *Requires*: When a feature is included into a configuration, it requires another feature

to be included also.

- *Excludes*: When a feature is included into a configuration, it necessitates another feature to be excluded from the configuration.



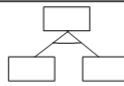
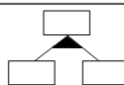
Relation	Notation
Mandatory	
Optional	
Alternative	
OR	

Figure 2.3: A common visual notation for some relationships among features

To make things more understandable, consider an example concept of car. Basically, it has two basic features; engine and transmission. Since, it is not sensible to sell a car without an engine and a transmission, these features can be considered as mandatory features for every member of the product line of the car concept. However, a car engine can work by consuming diesel fuel or gasoline but not by consuming both of them as a mixture. This will lead us the subfeatures of the engine feature; diesel and gasoline engine. These subfeatures are alternatives of each other. When building a car, engine feature has to be included and one of the engine types has to be specified among its alternatives. Similarly, the transmission feature brings two alternative types namely manual and automatic transmission. However, when we consider the sunroof feature, we can conclude that it is not a mandatory but an optional feature that will be included whenever there is a special customer request. A sample feature diagram of the car concept representing the mentioned features is depicted in Figure 2.4.

A feature model expresses and formally describes the configuration options in a problem space and the domain expert can form a specific product by configuring available features in the model. With this respect, a feature model is only capable of representing a certain kind of variability namely configuration or nonstructural variability. On the other hand,

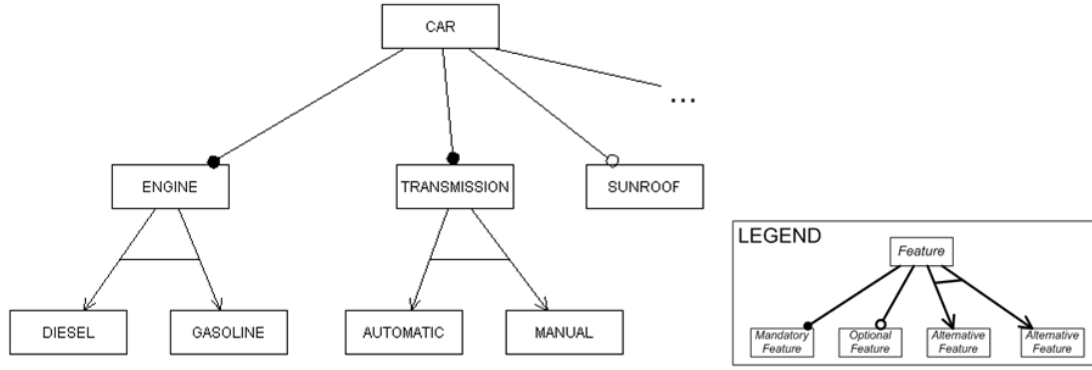


Figure 2.4: A sample and abbreviated feature diagram of the concept car

the domain expert cannot creatively construct a structured variability such as an arbitrary business process. Instead of feature modeling, she can use domain specific languages like activity diagrams or business process languages.

Whether representing a functional or a nonfunctional variability, a feature is abstracted in the same way in a feature model. Indeed, this distinction of variability is not apparent. When compared to other modeling approaches such as Unified Modeling Language (UML), feature modeling has advantageous especially at the early domain analysis phases because of this non-tricky variability representation property.

2.1.3 Application Engineering

Application engineering results in delivered products built by enabling domain engineering outcomes. In this phase, the focus is on a single product built with reused artifacts. In application engineering, features from the feature model, corresponding reference architecture and the reusable assets of the reference architecture are customized to form a particular product. Final products are generated by considering the *design-with-reuse* principle. Mainly, application engineering encompasses three process components;

- *Requirements Analysis*: A requirements model representing the needs of a specific customer is devised. The customer requirements are compared with the existing domain model and the matching features (requirements) are selected from the model. New requirements uncovered by the existing domain model are elicited and the core assets may be updated accordingly.
- *Design Analysis*: In this phase, reference architecture model is revised in order to

identify and capture the changes propagated from the new requirements and product specifications.

- *Integration and Test*: By using the reference architecture, identified reusable assets are integrated in order to produce the application code.

An example application engineering process is provided by Software Engineering Institute (SEI) in [4](Figure 2.5).

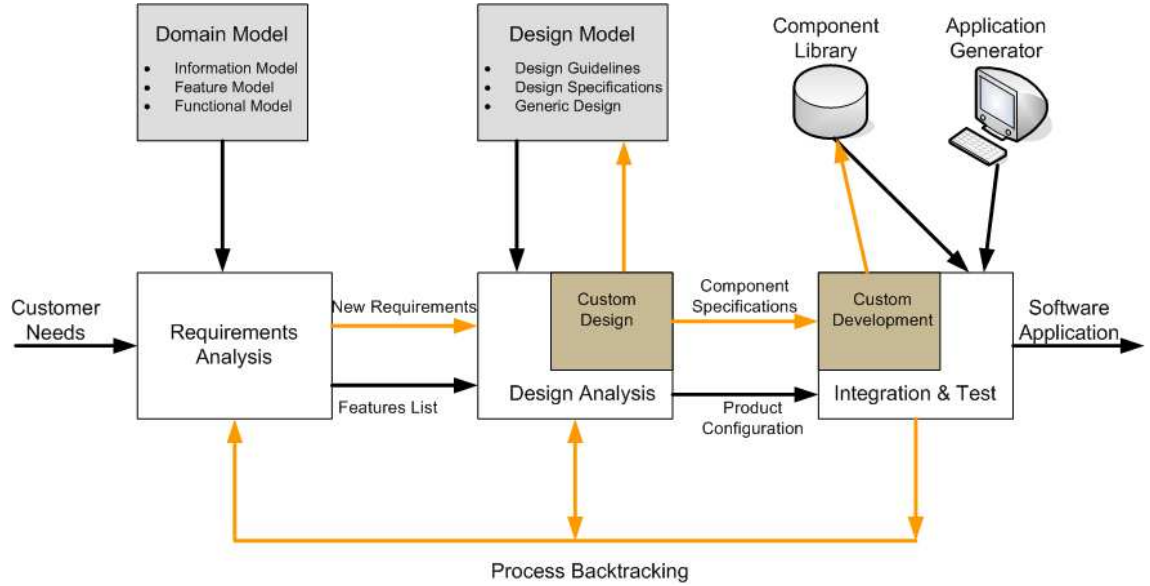


Figure 2.5: A sample application engineering process described by SEI

2.1.4 Software Factory Automation (SFA)

Software factory extends the concept of SPL by adapting, assembling and configuring extensible tools, models, frameworks and patterns using templates and schemas in order to automate the development and maintenance of product families [29]. Software Factory Automation was introduced by Altintas et al in [2, 3] to generalize the establishment of software factories as the way manufacturing industry has been doing. In other words, SFA is an industrialization model for establishing software product lines through constructing a domain design model based on Domain-Specific Kits (DSKs). Regarding the feature-oriented analysis of problem domain, SFA's design model encloses two major activities: first, building the product line reference architecture using DSK abstraction and then constructing a reusable

asset model based also on DSKs. The end results of all of these activities are the reference architecture and asset model for the product line. Major constituents of the approach have been given in [2](Figure 2.6).

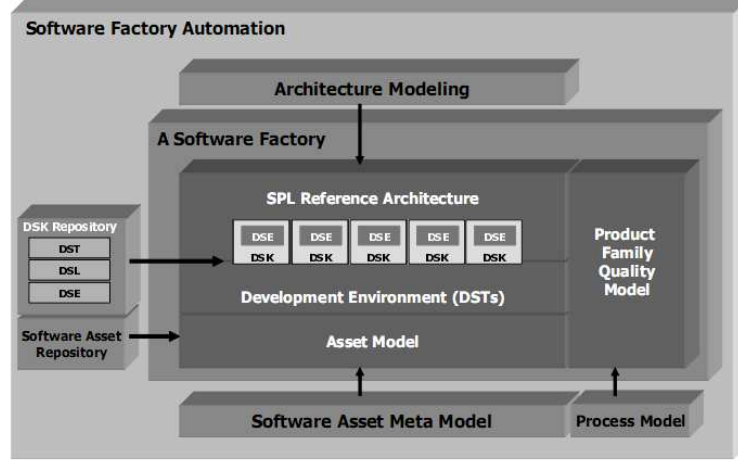


Figure 2.6: Overview of the software factory automation approach. Adapted from *Altintas, "Feature Based Software Asset Modeling with Domain Specific Kits", 2007* [2]

In contrast to general-purpose programming languages such as C and Java, Domain-Specific Languages (DSL)s focus on accomplishing specific kinds of tasks defined in a certain problem domain. DSLs allow developers to concentrate on the targeted problem domain's particular vocabulary, constraints, and concepts in higher levels of abstraction with specialized constructs and *syntactic sugar*. The aim of incorporating domain specific abstractions is to improve systematic reuse of software assets which leads to enhanced software development productivity, product quality and reduced per product development and maintenance costs.

The concept of *Domain Specific Kits* was first introduced by Griss and Wentzel [30]. DSKs as refined in [2, 3] are analogous to Programmable Logic Controllers (PLCs) utilized in industrial factory automation. PCLs and DSKs are sharing a common vision which is facilitating the production of domain specific artifacts in isolated units. SFA approach slightly modifies the traditional DSK definition and adds the following attributes;

- Kits are not specific to a product family; they can be reused across different product families.

- Kits cannot contain architectures for family of products; instead they are combined to form reference architecture of a product family.
- Kits contain logical connection points that let them collaborate with other kits in a choreography model.
- Artifacts of the kits are composable, but the kits can make use of generative approaches internally.

The language-oriented development terminology used in SFA is depicted in Table 2.2.

Table 2.2: Terminology of domain-specific kits

Domain Specific Language (DSL)	A language dedicated to a particular domain or problem with appropriate built-in abstractions and notations
Domain Specific Engine (DSE)	An engine particularly designed and tailored to execute a specific DSL
Domain Specific Toolset (DST)	An environment to design, develop, and manage software artifacts of a specific DSL
Domain Specific Kit (DSK)	A composite of a Domain Specific Engine (DSE) and a Toolset (DST)
Domain Specific Artifact (DSA)	An artifact that is expressed, developed, and executed by a DSL, DST, DSE, respectively
Domain Specific Artifact Type (DSAT)	A DSA type that a certain DSK can express, execute and facilitate the development

Reference Architecture Modeling in SFA correlates the architectural aspects and quality attributes of the problem domain with actual components and connectors of the solution domain by utilizing the *Symmetric Alignment* method as described in [12]. The architectural components and connectors in the solution domain are identified, and further abstracted by this alignment. SFA proposes a six-step reference architecture modeling approach that starts with the identification of quality requirements; architectural aspects and concern spaces of problem and solution domains followed by the symmetric alignment of both domains and ends with the reference architecture model with DSK abstraction.

The reference architecture completes the picture by providing a choreography model. The domain specific artifacts of separated concerns are composed through employed choreography model which is formed by a language and its engine. The envisioned choreography model of SFA relies on service-oriented paradigm. It ensures context management, state coordination, reliable message exchange and exception handling.

In order to enable knowledge-oriented software engineering, SFA employs an asset modeling method that aims to improve the commonality of features, and effectively manage the variations of them by exploiting a meta-model. SFA asset modeling method separates asset concerns by mapping features (DSATs) to DSAs, and later composes them using the SPL reference architecture. Hence, it generates more cohesive asset models to improve the asset reusability by reducing the interdependencies. In order to facilitate the modeling activities, SFA builds a modeling language (AML) to define the reusable assets for product assembly.

Conceptually, an asset is a composition of domain-specific artifacts specified by using different DSLs. AML based on the Asset Meta Model (AMM) compiles the asset definition and specifies variability points with collected artifacts and their choreography. The building blocks of AMM are DSATs, DSKs, context, constraints and dependencies among DSATs.

Variability management in software assets is the key factor to achieve high levels of systematic reuse, especially when considering the product family-based approaches. SFA enables the asset reuse not only within a product family but also across product lines. When the required DSKs definition exists in AMM definitions, an asset can be reused across different product lines. In other words, such assets can be reused if dependent artifact types are available.

In general, SFA approach exploits the concepts and methods that are coined in the generative software development paradigm. Generative software development promises automatic system generation from a specification written in one or more DSLs through modeling and implementing product lines [17]. SFA enhances the generative software development paradigm by DSK abstraction and choreography. In SFA, asset models are specified with Domain-Specific Artifact Types (DSATs) abstracted by DSKs. SFA approach encapsulates correlated artifacts and their interactions within more cohesive asset models and composes them through a choreography engine.

2.2 Service-Oriented Architecture (SOA)

By definition, Service-Oriented Architecture is an architectural style which enables reusable and encapsulated software services accessible over a network or a service bus in a loosely coupled and highly interoperable manner. The main idea behind SOA is to provide an agile IT framework that organizations can easily map their continuously changing business processes, requirements, evolving business needs and existing assets to IT capabilities represented as services. SOA promises to bring flexibility to IT assets, lower development cost by allow-

ing high levels of software reuse and provide necessary business agility through composition of services that are spanning multiple collaborative business partners in a standard-based manner [58].

A SOA realization is usually comprised of three main parties as listed below. The basic interactions between these parties are represented in [26] (see Figure 2.7):

- Provider (of services); basic service providers and aggregators
- Requester (of services); service aggregators and end users
- Broker (of services); middleware and registries

A service provider advertises its service through a service broker where service descriptions are published. Service broker or registry can be considered analogously as yellow pages for finding services and their providers. After publishing service description, it is ready to be discovered by the service requester via querying the service registry. Using the discovered service description, requester can bind with the provider in order to consume the service.

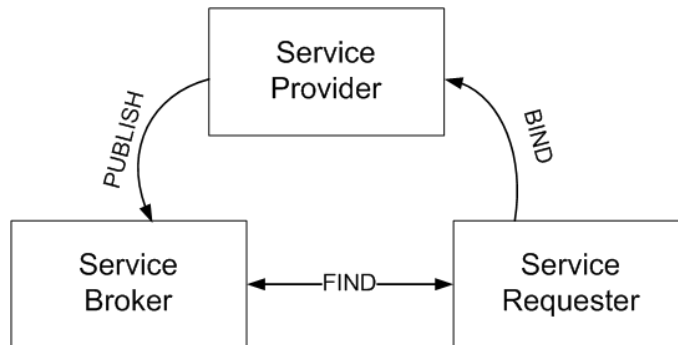


Figure 2.7: SOA actors in action

A service in SOA can be defined through the following agreed aspects [66]:

- Services are defined by explicit, implementation-independent interfaces
- Services are loosely bound and invoked through communication protocols that stress location transparency and interoperability
- Services encapsulate reusable business functions

- Services can be integrated via service composition mechanisms to provide extended collaborations. Static or run-time integration of services can be possible by using different service composition mechanisms.

From software architecture and integration viewpoint, a service oriented architecture is a key mechanism to support software reuse. A service can be defined as a unit of work done by a service provider to achieve desired end results for a service consumer [36]. Services have important characteristics enforcing Object-Oriented (OO) design principles such as ensuring separation of concerns and loose coupling. A service can be reachable through ubiquitous interfaces by all other participating software consumers. Only generic semantics are encoded at the interfaces. The interfaces are universally available for all providers and consumers by means of a service bus. Communication with a service is message-oriented and exchanged messages are delivered through the interfaces. Usually, messages are compiled according to an agreed schema and the message schema limits the vocabulary and structure of messages. This message-oriented communication via ubiquitous interfaces allows to change and extend the current versions of services and also introduce new versions without breaking existing services. As a consequence of the distributed nature of services, a SOA must have a mechanism that enables a consumer to discover a service provider, under the context of a service sought by the consumer. A SOA approach delivers key architecture goals in supporting:

- Benefits of OO including component re-use
- Reduce dependencies between system components to achieve loose coupling
- Extensibility
- Services distribution
- Scalability and extensibility

In order to get maximum benefit from a SOA approach, these fundamental issues should be addressed based on open standards:

- A common framework for service interactions based on open standards must occur at least for proper;
 - Communication
 - Description

- Registration
- Composition as Choreography and Orchestration
- An agreed set of vocabularies and interactions (common processes) for specific industries or common functions must be adopted

2.2.1 Web Services

The most common (but not only) form of services used extensively in SOA is Web Services, in which (1) service interfaces are described using Web Services Description Language (WSDL) [83], (2) payload is transmitted using Simple Object Access Protocol (SOAP) [71] over Hypertext Transfer Protocol (HTTP) [38], and optionally (3) Universal Description, Discovery and Integration (UDDI) [73] is used as the directory service. With these open standards developed by collaboration of various contributors from academia and industry, it was made possible to establish heterogenous network and service infrastructures that a diverse array of proprietary enterprise solutions of different vendors can communicate and collaborate with each other in an interoperable manner like in today's GSM and PSTN infrastructures. The main goal of such an interoperable service utility is to bring technology closer to people and organizational needs by hiding technology complexity and revealing functionality on demand. A visualization of the layered and interrelated technologies in the web service architecture is given in [81] (see Figure 2.8).

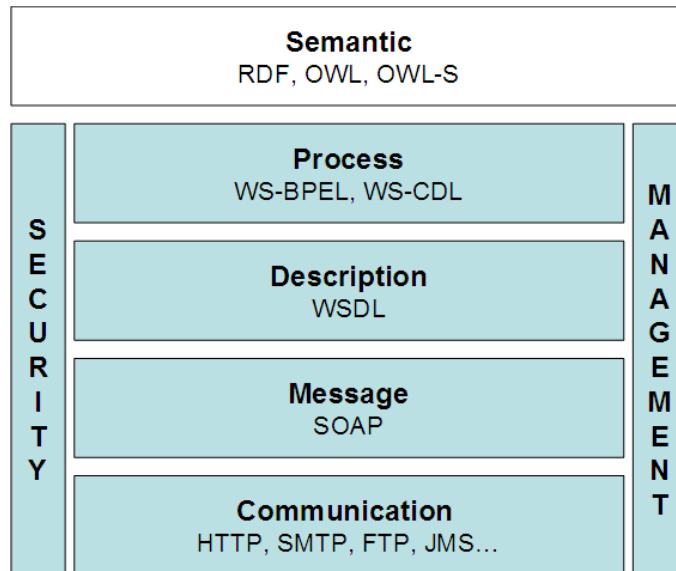


Figure 2.8: Web services technology stack

WSDL is a well known and widely adapted XML format for web service descriptions. Mainly, it describes the properties of the interface of a web service. In WSDL, a web service is defined as a set of network endpoints operating on document-oriented or procedure-oriented messages. WSDL separates the service meta definition from the concrete service realization. The abstract endpoints definitions allows flexibility in choosing the necessary bindings for network protocol and message schemas. The common network protocol bindings for WSDL are HTTP with SOAP 1.1.

As an XML based protocol, SOAP is targeted for information exchanges between agents in distributed environments. It consists of three essential components: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses [71]. Although SOAP makes uses of various Internet application layer protocol as a transport protocol, it is used in combination with HTTP and HTTP Extension Framework in usual.

UDDI protocol is an approved OASIS Standard for representing data and metadata about web services on registries. As a key member of the web services stack, it defines how to publish, discover, retrieve and manage information about network-based software entities of a service-oriented architecture through utilizing service registries. In general, a service registry is responsible for supporting the description and discovery of businesses, organizations, and other services providers, services made available to clients, and the technical interfaces which may be used to access those services.

More recently, REST (Representational State Transfer) [27] web services have been becoming popular. These kind of web services also meet the W3C definition, but instead of enabling XML based standards such as SOAP and WSDL, they rely on pure HTTP methods and constructs.

2.2.2 Service Composition

One of the promising benefits of service-oriented architecture (SOA) is to build complex composite applications or services by reusing other existing services according to a business process definitions. Moreover, SOA allows organizations to achieve different levels of collaboration among large numbers of services from heterogeneous environments without regarding to the details and differences of those environments. Service composition accomplishment depends heavily on the services having coarse-grained interfaces [35]. Coarse-grained services are intelligently structured to meet specific business needs and constructed from fine-grained

services which provide a small amount of business-process usefulness, such as basic data access.

The associations between service interface granularity, service compositions and business process definitions are shown in Figure 2.9. Proper abstraction of service endpoint implementations requires dealing with the differences in protocol, semantics, policy, availability, and should consider security, management, quality, and governance issues to guarantee reliable communications.

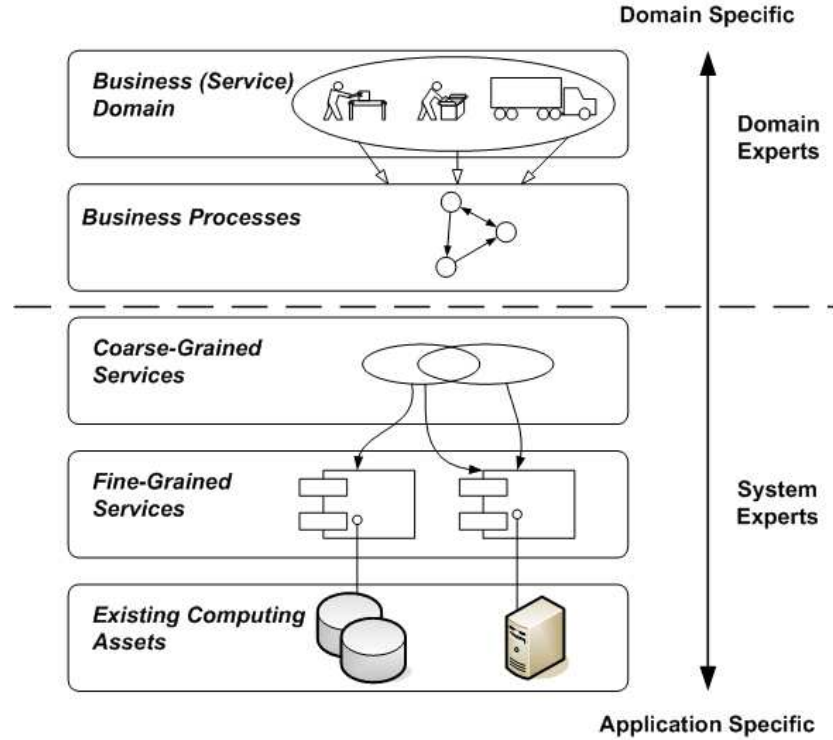


Figure 2.9: Service development life-cycle hierarchy

Primary characteristics of SOA from the service granularity perspective can be listed as follows [9]:

- SOA can offer fine-grained atomic services such as simple function calls as well as coarse-grained business services which are used to divide larger applications into smaller discrete modules.
- Services in SOA have minimum amount of interdependencies.

- SOA exploits service granularity principles to provide effective composition, encapsulation and management of services.

One approach for specifying the right level of service granularity is to start with accompanying business processes and decompose them into increasingly smaller subprocesses until reaching a set of atomic processes that cannot be divided into more smaller components. The resulting atomic processes then become candidate for being realized as services. The more processes are decomposed in this way; the more commonalities are captured among atomic processes. Hence the possibility to build more reusable services is also increased.

A number of open standard notations for web service composition have followed the establishment of common web service protocol stack and enhance it by addressing the process side of the service collaborations. According to their characteristics, service compositions can be categorized in two broad domains namely;

- *Orchestration* coordinates the flow of service interactions from the perspective of a single authority domain. A number of standard languages addresses the web services orchestration specification such as Business Process Execution Language (BPEL) [10]. Most likely, the defined coordination in service orchestrations are limited to intra-organizational borders.
- *Choreography* can be considered as an orchestration of service orchestrations. Generally, a choreography defines the sequence of the interactions, message flows, rules and conditions among multiple parties in coordination. The main difference from the orchestration is in choreography there are multiple authority domains and the defined coordination is inter-organizational. Web Services Choreography Description Language (WS-CDL) [82] is one of the foremost open and standard choreography definition language for web services.

2.2.3 ebXML Business Process Specification Schema

BPM and SOA are two independent initiatives. BPM is mainly a management discipline and strategy for business processes. A business process encapsulates business transactions and business documents and sets the collaboration rules among business partners. Business processes can be represented in a machine-readable format. Nowadays, the BPM-SOA convergence is proposed to organizations so as to facilitate a closer alignment between business processes and IT resources and reach the desired business agility and responsiveness to changing business parameters in today's highly competitive digital world of business [43].

ebXML Business Process Specification Schema (ebBP) [22] provides means for defining e-Business collaborations between collaborating business partners through a standard technical specification. ebBP is an initiative from OASIS [50] and it is based on XML [85]. Business systems of collaborating parties may be configured to execute the business transactions defined in ebBP documents.

Mainly, ebBP specifies the choreography of *Business Transactions* taking place among two or more collaborative business partners. The choreography definition and business transactions are compiled to form a *Business Collaboration*. The flow of exchanged business documents, signals and the decisions points are placed in business collaborations in a stateful manner. Each business collaboration specifies a set of roles collaborating through business transactions. Business transactions involves participation of two complementary abstract roles namely, *Requesting Role* and *Responding Role*. In order to specify the choreography of business activities, ebBP provides a number of states (*Start*, *Completion*) as well as a set of gateways (*Fork*, *Join*, *Decision*).

Business transactions in ebBP are atomic processes that cannot be further decomposed into lower level business transactions as in business collaborations. Additionally, *Business Signals* which are used for ensuring the state alignment between collaborating parties can be exchanged as a part of a business transaction.

An example multi-party collaboration is represented with Business Process Modeling Notation (BPMN) [11] and also QueryResponse business transaction definition with just a requesting and response document flow is provided in [22](see Figure 2.10).

ebBP definitions are independent from the underlying platform, software or services and provide a level of abstraction in order to gain in flexibility to be used with different technologies. ebBP refers to logical business document schemas and associates them through exchanged messages in business transactions. In brief, such capabilities of the ebBP can be stated as in [49];

- Standard and extensible business transaction patterns
- Support for multiple role bindings
- Flexibility for complex transaction activities
- Support for use of web service, hybrid and ebXML assets
- Late binding capabilities such as for timing
- Semantic tailoring for business processes and business documents

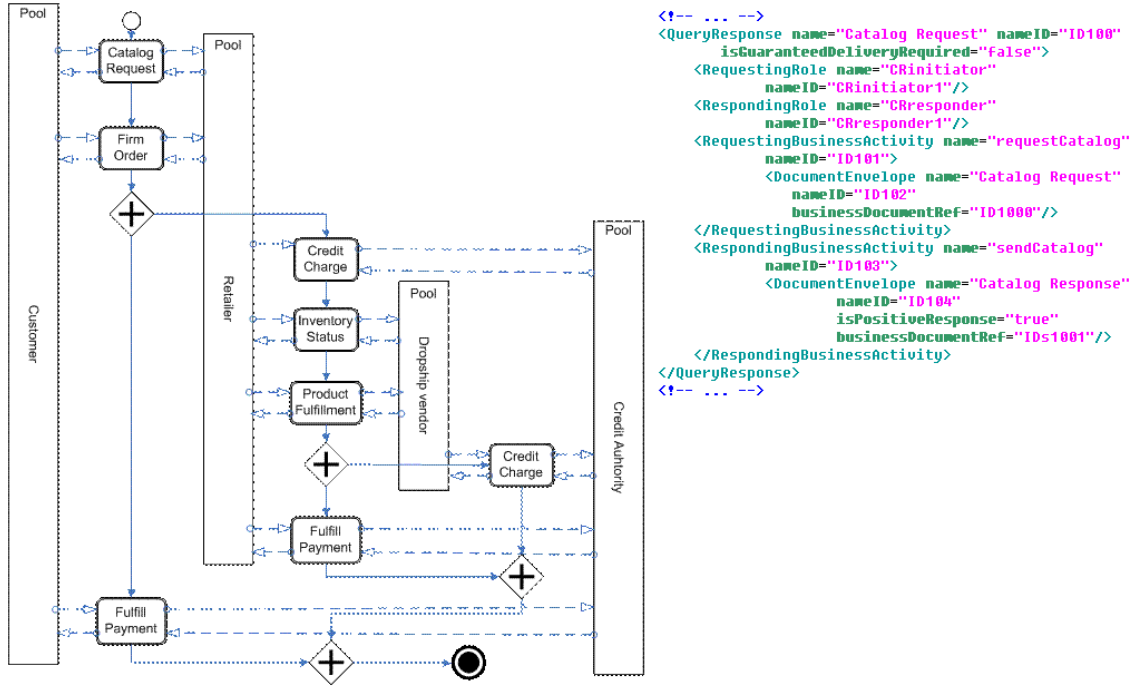


Figure 2.10: Representation of the "Drop Ship" multi-party collaboration with BPMN and the definition of a QueryResponse business transaction in ebBP format

Theoretically, ebBP standard can be enabled for representing business processes in almost every e-business domain with different systems collaborating to achieve a desired business goal. An example in the eHealth domain [1] demonstrates collaboration profiles among healthcare systems are expressed through ebBP language. Details of the ebBP specification and how it is related with this thesis work is explained in Chapter 3.

2.2.4 Semantic Interoperability

Another problem space which SOA is frequently being applied for is semantic and technical interoperability in heterogeneous environments. However, in order to achieve this goal, organizations should adopt open industry standards in their messaging, metadata, implementation and collaboration efforts to build service-oriented systems. These standards are defined by non-profit international standardization organizations like WC3 [76] and OASIS with the help of industry contribution. Nevertheless, different standards that have been defined by different consortiums may address similar problems, as it can be seen clearly in Electronic Healthcare Records [24]. An organization feels free to adopt any standard among its alternatives when building its proprietary system. Because of such variations, in order to

develop portable, re-usable and interoperable services that are guaranteed to work together with other proprietary services from a global perspective; we should not only conform to standards but also provide mediation mechanisms between different standards [74].

2.2.5 Historical Origins and Motivating Needs Behind SOA

In the seventies, companies like SAP started to develop IT stuff which were the initial steps in software engineering. Most of the developed software in those days were monolithic, batch-oriented applications for mainframes and terminals. In the eighties, PCs were introduced with the capabilities of running office applications such as spreadsheets and shrink wrap products and also terminal emulators used for getting access to legacy mainframes. In the nineties, mainframes and terminals still stood in the forefront of enterprise systems but spreadsheets, shrink wraps and other stand alone individual applications were enriched with graphical user interfaces (GUIs) used in an ad hoc fashion. At those times, Object-Oriented and client-server computing were materialized also in industrial practices to a certain extent.

Day by day, Object-orientation was becoming more important as a means of integrated development suit for analyzing the business, design and also programming. Object-oriented approach was promising attractive innovations like not only richer GUIs but also better modularity, maintainability and reusability of software entities. Although, enterprises attempted to get those advantages in their business, they encountered with severe challenges, especially when updating their mainframes on which had been invested previously. New features and extensions such as thin applications with GUIs could be implemented by using new technologies like C++ but the real business logic remained on the legacy mainframes. The main problem coming from the general IT perspective was that the Object-orientation could not be exploited throughout the company because the mainframes, databases and other legacy applications were developed based on traditional approaches rather than the Object-orientation. Moreover, leaving the previous investments and requesting newly developed systems with the same business logic would be expensive and cumbersome.

In general, migrating legacy applications to more modern systems holds several promises for an enterprise from maintenance and sustaining viewpoints such as [65];

- The technology which was used for implementing the application maybe obsolete in time and the personnel who have hands-on practice on this technology maybe rare and their salaries maybe over valued in accordance with the simple demand and sup-

ply principles. For example, in Y2K problem, updating and bug-fixing tasks for the legacy enterprise applications were outsourced to India because of scarce and overvalued resources of COBOL developers in western countries. COBOL, once a popular programming language for mainframes, was constantly becoming a legacy technology and losing its ground to other languages such as C++ and Java until Y2K.

- Migrating legacy applications increases the application reach and removes risks associated with running potentially unsupported hardware and software in legacy systems.
- A successful modernization offers not only technological modernization but also business process reengineering. In this respect, organizations can have opportunities to increase in efficiency with their usual business processes.

In the nineties, component-oriented distributed technologies were proposed to solve the legacy system migration problem. They brought remote procedure calls and object request broker middlewares to open the business logic to more modern systems. However, as the distributed computing gains more attention, the drawbacks of the distributed component-oriented systems such as CORBA [15] and COM [14] become more apparent in terms of complexity, security, handling latency, partial failures and concurrency and the lack of shared memory access [77, 37]. As a consequence of the experiences gained from these challenges and drawbacks, the technical background was prepared for the SOA.

With the emergence of the service-oriented architecture and web service technologies in the new millennium, legacy migration strategies have been evolved into this new distributed architecture proposed to address message orientation, self-description, platform-neutrality, and network orientation [79]. Currently, various vendors are offering legacy migration strategies based on SOA [59, 53, 69] and apart from being just only a system integrator, in near future SOA is being expected to have a more comprehensive role as a new computing paradigm [58] for software development.

2.3 Semantic Web Technologies

Semantic Web [8] technologies aim to represent and exchange information on the web through formal methods and definitions. Semantic web envisions a web of data where interoperability is realized between data sources and the web and the meaning of the data is given in a way that software agents can understand and reason with it. In the core of the Semantic Web there is the concept of ontology which is previously used in Artificial Intelligence and

Database communities, in order to formally model a conceptualization and enable knowledge sharing between information resources [32]. The motivating needs behind the semantic web initiatives are the difficulties to find, present, access, or maintain available electronic information on the web and to enable the software agents to provide intelligent access to heterogeneous and distributed information.

The word ontology comes from the Greek language; *ontos* (being) and *logos* (word). Formally, ontology can be defined as [31];

"An explicit formal specification of a conceptualization."

In practice, an ontology describes objects and concepts as classes. Typically, classes are formal and explicit description which are catalogued in a hierarchy. Class relationships are defined through class properties which describe intrinsic and extrinsic attributes of classes. For example, *IS-A* relation is extensively used for representing inheritance relations among classes thus for creating class hierarchies. A class property has two main concepts; *Domain* where classes which the property describes and *Range* where classes allowed to fill in the property value. Instances are individual occurrences of classes which they are member of. The properties associated with the class are filled in with instance values. Another interesting feature of ontologies are axioms which are ontological assumptions that cannot be described using only properties and property values. *Disjoint* is an axiom that represents the situation for two classes which has no instances in common. Consistency of an ontology according to defined axioms can be checked by using automated reasoners.

Generally, semantic web ontologies like OWL are based on open world assumption. In a closed world assumptions like Databases schemas, the information is assumed to be complete and a query engine will return a negative if it cannot find some data. However, a reasoner cannot determine something does not hold unless it is explicitly stated in the ontology. The reasoner does not make any assumption about the completeness of the given information.

Ontologies are exploited in order to establish a common vocabulary and understanding about any thing such as a concept or a domain. One of the most important benefits of ontological approaches for computer science is their supports for automated operations such as querying and reasoning. Ontologies are capable of describing the semantics of the data or meta-data in order to provide a uniform way to make different parties to communicate with each other. Both human and software agents can be use the domain knowledge described with an ontology.

Ontologies are widely used in semantic web. A common application area of ontolo-

gies is semantic annotations of web pages. World-Wide Web Consortium (W3C) is leading those initiatives on standard languages for ontology specification. In this section, well know domain-specific languages, frameworks and tools for enabling semantic web applications and semantic-based software development is presented as the building blocks of the semantic web.

2.3.1 Resource Description Framework (RDF)

Resource Description Framework (RDF) [61] is a framework for describing and interchanging metadata. RDF is the first and the simplest of the semantic web languages. It enables an XML based syntax and exploits URIs for resource identification. RDF provides machine processable semantics for metadata and associates them with resources on the web. This approach promises better precision in resource discovery than full text search and the interoperability of metadata. RDF has following aspects;

- Resource : Anything that can be named via a URI can be described by RDF as a resource. A URI makes the associated resource unique and globally known.
- Property : A property is a predicate used for building RDF instances with associated values such as aspects, characteristics, attributes or relations. A property is also a resource that has a name. The type of the an associated value (or an object) can be a literal or another resource.
- Statement : A statement is a triple that consists of a Resource, a Property, and an associated value.

RDF statements or the relationships between resources, properties, and the objects can be represented with directed labeled graphs. In this representation, resources and objects are identified as nodes, and properties are defined as edges. In Figure 2.11, an example RDF graph is given with accompanying definition in XML-based syntax.

RDF is in favor of using conventions that will facilitate modular interoperability among separate metadata element sets. The Dublin Core [21] is an example definition standard and convention for describing generalized web resources. It was named after the Metadata Workshop in Dublin, Ohio in 1995. Following list is the partial tag element list for Dublin Core initiative;

- Creator: Author of the content of the resource

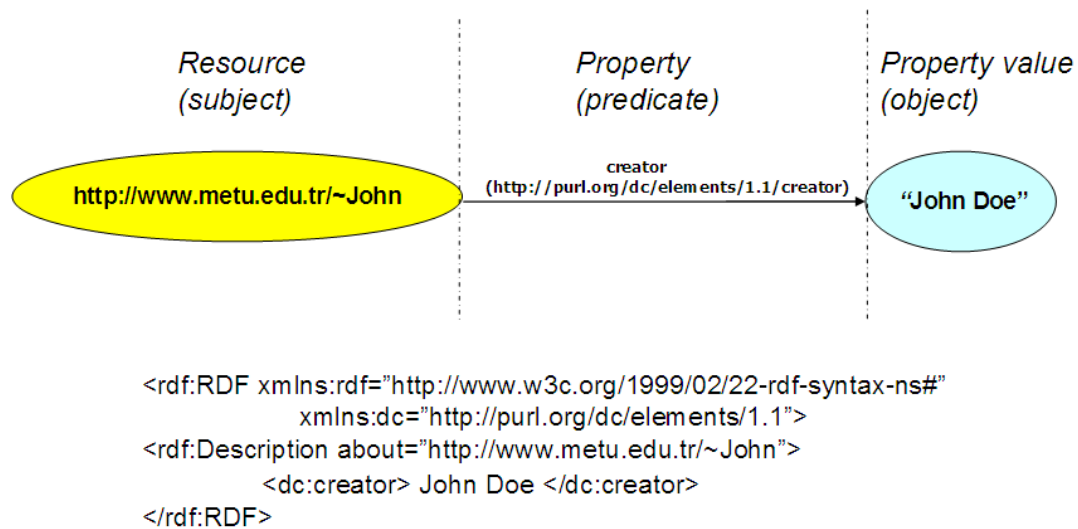


Figure 2.11: An example RDF statements which means that John Doe is the creator of the resource `http://www.metu.edu.tr/~John`. Property "creator" refers to Dublin Core Definition Standard. RDF statement is given in two different representations; directed labeled graph and XML notation.

- Date: The resource creation date
- Title: The name of the resource
- Subject: The domain related with the resource
- Description: An account of the content
- Type: The genre of the content
- Language: The natural language of the resource content

RDF Schema (RDFS) [62] is a rule system for building RDF instances and brings extensions to Resource Description Framework. RDF Schema provides a higher level of abstraction than RDF through its additional mechanisms to define specific classes of resources, properties, and the relationships between these properties and other resources. Moreover, RDFS provides a semantic capabilities basis for more expressive languages such as Web Ontology Language (OWL). RDF Schema provides three ways to characterize a property;

- Domain: Associates a property with a class
- Range: Indicates the range of values for a property
- subPropertyOf: Specializes a property

2.3.2 Web Ontology Language (OWL)

As a semantic web language, OWL [55] is used for defining terms and their relationships through its standard RDF/XML syntax. RDF Model and Syntax provides a recognizable metadata and RDF Schemas brings mechanisms for defining shared vocabularies and metadata interoperability. On the other hand, OWL is built on top of RDF, based on predecessor languages DAML+OIL [20] and extends RDF(S) semantically. All of the language constructs provided by RDF and RDFS can be used when creating an OWL document. Moreover, OWL documents exploit *rdfs:range*, *rdfs:domain*, and *rdfs:subPropertyOf* elements. Similarly, classes are the basic building blocks of an OWL ontology. Every instance in the universe is a member of *owl:Thing* class. OWL supports six main of describing classes;

- *Named Class* is the simplest way of describing an OWL class.
- *Intersection Class* is a combination of two or more classes via the logical intersection (AND) operator. An intersection class is defined with the *owl:intersectionOf* property.
- *Union Class* is a combination of two or more classes via the logical union (OR) operator. Union class is defined with the *owl:unionOf* property.
- *Complement Class* is formed by negating another class. Complement class is defined with the *owl:complementOf* property.
- *Restrictions* describe a class of individuals based on the type and possibly number of relationships that they participate in. Restrictions in OWL can be divided into three subcategories;
 - *Existential Restriction* describes a class of individuals that have at least one kind of relationship along a specified property to an individual that is a member of a specified class. Existential restriction can be read as "some values from" or "at least one".
 - *Universal Restriction* describes the class, whose individuals point only to members of the class specified in the restriction for a given property. It symbolized the meaning of "all values from" or "only".
 - *Cardinality Restriction* is about the number of relationships that a class of individuals participate in.
 - *Has Value Restriction* describes a class of individuals that participate in a certain relationship with a specific individual.

- *Enumeration Class* is formed by explicitly listing the individuals that are members of the enumeration class. An enumeration class is defined with the *owl:oneOf* property.

There are three types of OWL languages namely OWL-Lite, OWL-DL and OWL-Full. They are categorized according to their expressiveness power. OWL-Full has no restrictions on how/where language constructs can be used, but in turn, the OWL-Full ontologies are undecidable i.e. it is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full. OWL-DL corresponds to a description logic and has certain restrictions on how/where language constructs can be used in order to guarantee decidability. OWL-Lite is a subset of OWL-DL and is the simplest and easiest to implement of the three species.

OWL classes permit much greater expressiveness than RDF Schema classes. The benefit of OWL is that it facilitates a much greater degree of inferencing than RDF Schemas. For ontologies that fall into the scope of OWL-Lite and OWL-DL, a reasoner can be used to infer information that is not explicitly represented in an ontology. Subsumption testing, equivalence testing, consistency checking and instantiation testing are some of the main reasoning operations.

There are two types of properties in OWL namely *ObjectProperty* and *DatatypeProperty*. Each type of properties has single or multiple specified domains and ranges. An *ObjectProperty* relates one resource to another resource. A *DatatypeProperty* relates a resource to a literal or to an XML Schema datatype. Properties can have different characteristics such as;

- If a property P is *functional* then for all x, y, and z:

$$P(x, y) \wedge P(x, z) \rightarrow y = z \quad (2.1)$$

- If a property P1 is *inverseOf* P2 then for all x and y:

$$P1(x, y) \Leftrightarrow P2(y, x) \quad (2.2)$$

- If a property P is *inverse functional* then for all x, y and z:

$$P(y, x) \wedge P(z, x) \rightarrow y = z \quad (2.3)$$

- If a property P is *symmetric* then for any x and y:

$$P(x, y) \Leftrightarrow P(y, x) \quad (2.4)$$

- If a property P is *transitive* then for any x , y , and z :

$$P(x, y) \wedge P(y, z) \rightarrow P(x, z) \quad (2.5)$$

OWL provides additional axioms that can allow better reasoning capabilities on classes or properties. Some of the axioms defined by OWL are *rdfs:subClassOf*, *owl:equivalentClass*, *rdfs:subPropertyOf*, *owl:equivalentProperty*, *owl:disjointWith*, *owl:sameAs*, *owl:differentFrom*, *owl:inverseOf*, *owl:transitiveProperty*, *owl:functionalProperty*, *owl:inverseFunctionalProperty*.

2.3.3 Description Logics

Description Logics [6] is a family of logic based knowledge representation formalisms. Most of the semantic web ontology languages such as OWL are based on the description logics. Description logics is the decidable part of the first-order logic (FOL) and enables procedures for key reasoning problems such as satisfiability and subsumption. It describes the knowledge domain based on concepts (classes), roles (properties, relationships) and individuals. Basically, concepts represent unary predicates having only one free variable. Roles are equivalent to statements with two free variables. Individuals represent constants. Operators in DL are limited in order to foster decidability and reduce complexity.

DL brings the notion of knowledge base as a pair $\langle T, A \rangle$ where T refers to T-Box and A refers to A-Box. The T-box contains the terminological (or schema) axioms and A-Box contains assertional (or data) axioms. An example T-Box describing the concepts *Person*, *Man* and *Woman* with their structural relationships can be given in terms of FOL as follows;

$$Person \sqsubseteq \top \quad (2.6)$$

$$Man \sqsubseteq Person \quad (2.7)$$

$$Woman \sqsubseteq Person \quad (2.8)$$

$$Woman \sqcap Man = \perp \quad (2.9)$$

Semantic of description logics are defined by interpretations. An interpretation or model I is formed by a domain and an interpretation function. Domain is a set of the elements and objects which are subject to description or reasoning about where the interpretation function gives meaning to the members of this domain.

Well known reasoning problems that can be formed around a DL knowledge base $KB = \langle T, A \rangle$ will be as follows;

- Subsumption relation between two concepts w.r.t T

$$C \sqsubseteq D \Leftrightarrow \forall I. C^I \subseteq D^I \quad (2.10)$$

- Concept consistency w.r.t T
- Knowledge base consistency
- Instance checking; whether KB entails the individual a of the concept C

$$KB \models a : C \quad (2.11)$$

- Satisfiability; concept C is satisfiable iff

$$\exists I. C^I \neq \emptyset \quad (2.12)$$

- Equivalence of two concepts

$$C \equiv D \Leftrightarrow \forall I. C^I = D^I \quad (2.13)$$

- Disjointness; two concepts C and D are disjoint iff

$$\forall I. C^I \cap D^I = \emptyset \quad (2.14)$$

2.3.4 Protégé Ontology Editor and Knowledge Acquisition System

Protégé [60] is a popular and comprehensive tool for creating and modifying knowledge bases. Through its graphical development environment, Protégé supports development of ontologies and meta models and exports them to various languages such as XML schema, RDF(S) and OWL. Being a free and open source project, it allows extensions that can be implemented and plugged in the flexible environment. It is actively being developed and extended by a community of developers and academic, government and corporate users. Likely, Protégé can be enabled for any application areas where knowledge management and representation are musts such as biomedicine, intelligence gathering and corporate modelling. A screenshot of the Protégé tool is given in Figure 2.12.

In this thesis work, Protégé is exploited and reused as a third party component for accessing and modifying OWL instances in a highly effective way. Protégé provides users and developers with Protégé-OWL plugin which handles almost every possible actions over OWL documents. Moreover, Protégé supports Semantic Web Rule Language (SWRL) for OWL instances through its special plugins. Protégé is capable of converting and executing SWRL rules in the JESS rule engine.

Details on how Protégé and its plugins are enabled in our work is given in Chapter 4.

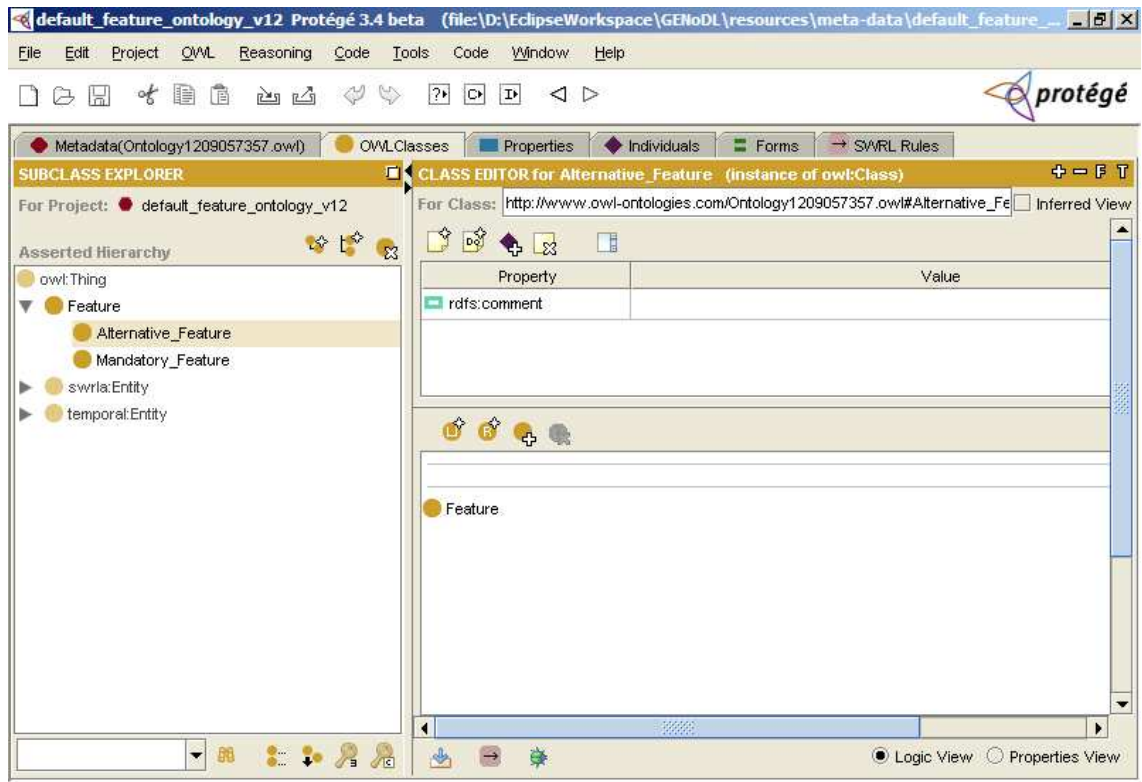


Figure 2.12: A screenshot from the OWL Classes view of the Protégé Ontology Editor

2.3.5 OWL-S: Semantic Markup for Web Services

Ideally, software agents should be able to automatically discover, invoke, compose, and monitor web services. However, there is a need to make the service understandable and interpretable by the software agents to achieve this goal. In this regard, OWL-S [56] is designed and developed to represent web services semantics and populate service description and functionalities so that software agents can perform reasoning for automated service discovery, invocation, composition and interoperation. The current version of OWL-S is built on the OWL.

The overall structure of the OWL-S ontology is divided into three main parts in order to provide a comprehensive model for advertising and discovering services; a detailed description of a service's operation; and the details on how to interoperate with a service via messages. Upper ontology of services is visualized in Figure 2.13.

Core components of the service ontology are explained as follows;

- *Service Profile* provides meta information about the service and its capabilities. Profile specifies the inputs and outputs of the service, pre-conditions for using the service and effects that service produce after its execution. Service profiles are used to populate service registries and automated service discovery and matching. They can also support

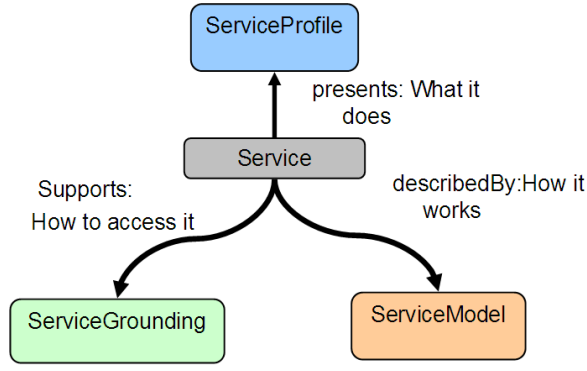


Figure 2.13: Upper ontology of services

non-functional properties such as service name, text description, quality rating and service category. An example service profile is given for a flight reservation service. In this markup segment, service provider is annotated within the *contactInformation* tag. Type of the service is enclosed in *serviceClassification*, product of the service is described by *product* and service classification in business taxonomies is given in *serviceCategory*. Typical inputs, outputs, preconditions and effects of the service is given in *hasInput*, *hasOutput*, *preconditions* and *effects* respectively. This service profile example is adapted from [57].

```

<profile rdf:ID="BravoAir">
  <serviceName>BravoAir </serviceName>
  <contactInformation rdf:resource="#BAco"/>
  <serviceClassification rdf:resource="#Airline"/>
  <product rdf:resource="#FlightReserv"/>
  <serviceCategory rdf:resource="#NAICS_Airline"/>
  <hasInput rdf:resource="#Dep_Airport"/>
  <hasInput rdf:resource="#Arr_Airport"/>
  <hasOutput rdf:resource="#Reservation"/>
  <preconditions/>
  <effects/>
</profile>

```

- *Process Model* is specification of ways how and when a client may interact with a service. A Process model can have one or more simple, atomic and composite processes. An atomic process is an interpretable description of a service that can be executed in single step. A composite process maintains the state of the process. A composite process may consist of sub composite or atomic processes. On the other hand, a Simple Process is an abstract definition which is non-invokeable and have no grounding. They can be considered as reusable process definitions that can be bound within an atomic process. Process model provides state alignment and dataflow of a composite process through its control constructs; *Sequence*, *Split*, *Split+Join*, *Choice*, *Any-Order*, *Condition*, *If-Then-Else*, *Iterate*, *Repeat-While* and *Repeat-Until*. An excerpt from the atomic process of the online purchase operation is given as follows;

```

<process:AtomicProcess rdf:ID="Purchase">
  <process:hasInput>
    <process:Input rdf:ID="ObjectPurchased"/>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="CreditCard"/>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="ConfirmationNumber"/>
  </process:hasOutput>
...

```

- *Service Grounding* specifies how to access a service in an interoperable way. Details on the access method such as communication protocols, schemas for exchanged messages and port numbers used to contact the service are specified in this part. Service grounding is built upon WSDL specification.

Details on the OWL-S ontology and how it is related to ebBP are explained in Chapter 3.

2.3.6 Semantic Web Rule Language (SWRL) and JESS

Semantic Web Rule Language (SWRL)[72] is an initiative from W3C which is designed to be the rule language of the Semantic Web. SWRL is based on the OWL DL and OWL Lite languages along with the Unary/Binary Datalog sublanguage of the Rule Markup Language. SWRL supports Horn-like rules expressed in terms of OWL concepts (classes, properties and individuals) and stored as a part of ontology. SWRL is intended to reason about OWL individuals. Moreover, SWRL rules can be used to infer new knowledge from an existing OWL knowledge base. SWRL specification does not mandate or impose a particular reasoner to be used in performing reasoning with SWRL rules.

SWRL is based on OWL's open world assumption and provides more expressivity than OWL DL alone. However, inference with SWRL rules is not guaranteed to terminate as a cost of the increased expressivity.

A SWRL rule is a combination of a head and a body part. Each of these parts consists of a conjunction of zero or more atoms. SWRL does not support more complex combination of atoms at the moment. There are a number of possible forms for the atoms in SWRL rules. $C(x)$, $P(x, y)$, $\text{sameAs}(x, y)$ and $\text{differentFrom}(x, y)$ are defined as atom forms in SWRL specification where C is an OWL description, P is an OWL property, and x, y are

either variables, OWL individuals or OWL data values. In the SWRL documentation it is stated that OWL DL becomes undecidable when extended in this way as rules can be used to simulate role value maps. SWRL also supports a range of built-in parameters in order to expand its expressiveness. All of these built-in predicates start with the namespace qualifier "swrlb:".

Consider an example SWRL rule which expresses a person with a female sibling. It requires capturing the concepts of "person", "female", "sibling" and "sister" in OWL. Intuitively, the concept of person and female can be captured using an OWL class called *Person* with a subclass *Female*; the sibling and sister relationships can be expressed using OWL properties *hasSibling* and *hasSister*, which are attached to *Person*. The rule in SWRL would then be:

$$Person(?x) \wedge hasSibling(?x, ?y) \wedge Female(?y) \rightarrow hasSister(?x, ?y)$$

Executing this rule would have the effect of setting the *hasSister* property to *y* in the individual that satisfies the rule, named *x*.

JESS [39], the rule engine for the Java platform, is a rule-based reasoning system for the Java platform which provides a rule language and a comprehensive Java based programmer's library for the seamless integration of its rule engine with Java applications. The rule language of the JESS can be considered as a dialect of the Lisp programming language. Moreover, JESS supports an XML-based rule language called JessML (JESS's XML rule language).

JESS is a well known and highly appreciated system which is well documented and is easy to use and configure. Although it is not free to common use, this rule system can be downloaded for a 30-day evaluation period and is available free for academic use. The rule engine can be embedded to any Java applications and provides a two-way runtime communication between JESS rules and the application where it is embedded. JESS is capable of directly manipulating Java object. Its rule processing facility is based on an enhanced version of the *Rete* algorithm [28]. Protégé-SWRL chose JESS as the first integration candidate for the reasoning operations.

Although a JESS rule is very similar to a conditional (if... then statement) which is a core part of any procedural language, it is not used in a procedural way. As it is stated in manual of the JESS, this is because of the designed rule execution mechanism. Conditionals are executed at a certain order and time, however JESS rules are executed whenever their if parts (their left-hand-sides) are satisfied. Therefore, a typical procedural program is more

deterministic than the JESS rules in this respect.

Mapping between OWL knowledge base with SWRL and JESS rules are possible and are already demonstrated in [51, 45]. Before representing OWL individuals as JESS knowledge, OWL classes and their properties must be first transformed into JESS. For this purpose, JESS's template facility can be used to represent the OWL class hierarchy. Our example OWL knowledge can be modeled in JESS as follows;

```
(deftemplate OWLThing (slot name))
(deftemplate Person extends OWLThing)
(deftemplate Woman extends Person)
```

Consider that "Simge" is a member of the *Woman* class as well as "Umut" is of the *Person*. Using the template definitions given above, these OWL individual can be asserted as a member of the class *Woman* and *Person* respectively;

```
(assert (Woman (name Simge)))
(assert (Person (name Umut)))
```

OWL properties are directly modeled as JESS facts. We can assert such a fact if "Umut" has a sibling called "Simge";

```
(assert (hasSibling Umut Simge))
```

The representation of our SWRL rule, which is given above, in terms of JESS is as follows;

```
(defrule exampleRule (Person (name ?x)) (Woman (name ?y))
(hasSibling(?x ?y))
=> (assert (hasSister(?x ?y)))
```

More detailed information on how to perform reasoning based on OWL, SWRL and JESS can be found in Chapter 4.

CHAPTER 3

ENHANCING DOMAIN KNOWLEDGE BASE WITH BUSINESS PROCESS DEFINITIONS

Although feature modeling is a well known and easy to adopt method for representing commonalities and variabilities among a set of end-user driven domain specifications, they are not suitable for capturing interactions, message sequence and data flows taking part in between two or more systems, services, processes or objects. In order to model the process model semantics of a web service properly, it is necessary to enhance developed approaches with additional domain engineering utilities.

In this respect, ebXML Business Specification Schema (ebBP) is exploited in capturing process definitions and business choreographies of web services from a high abstraction level where contributions of domain experts such as business analysts and process managers can be effectively incorporated in devising the domain model. An overview of the ebBP standard is given in Section 2.2.

The main advantage of using ebBP is its powerful built-in mechanisms for separating the definition of the process model from its realization and making it independent from its enablers namely business actors. This means that individual business systems, which are following the sequence, can be defined in a domain of control where changes can take place internally without changing the actual process model. Hence, ebBP provides business partners with business process definitions to collaborate and achieve a given business goal in an interoperable way. Because of these important properties, ebBP standard is decided to be our choice of formalism for business process modeling.

Today, the key enabler of the ebBP is the web service technology which makes it possible

to execute business collaborations defined in the specification. One way to achieve such executable business collaboration is through mapping business transaction activities to web services. ebBP is capable of specifying process model parameters for configuring service interfaces to execute and monitor business collaborations. However, ebBP does not specify how to associate a defined service interface to its real world implementation. There are several alternative methods, technologies and standards considering the business service interface implementation such as ebXML Message Service, Collaboration-Protocol Profile and Agreement etc.

In this chapter, we first introduce the building blocks of the ebBP in detail and then present the ebBP Editor, an open source initiative designed to help the domain experts in creating domain specific ebBP instances in a user friendly way, and finally explain the devised method for transforming a given business process model to the corresponding service model representation.

3.1 Core Components of the ebXML Business Process Specification Schema

The ebBP standard aims to specify the semantics as well as the structure of business collaborations in a machine readable format. Basically, business collaborations are composed of business transaction choreographies which are fulfilled through message exchanges between collaborating business partners. Each business partner plays a predefined role in order to participate in the business collaboration. In each business transaction, there is at least one business document flow. Business signals may be enabled for informing the collaborating parties about the current status of the business collaboration and business exceptions threw during the execution of the business transactions.

In the following subsections, the details of core components in the ebBP specification are described. The standard schema definitions of these core components are provided in the Appendix section.

3.1.1 Business Collaborations and Choreography

A business collaboration consists of business activities which execute business transactions among collaborating parties. Involved business partners play a predefined abstract role in the scope of the business collaboration. Business transactions define interactions between abstract partner roles based on transaction patterns. Abstract roles are bound to concrete

business parties through business activities. Business activities can be categorized as follows;

- *Business Transaction Activity*: The activity of executing a single business transaction.
- *Complex Business Transaction Activity*: The activity of executing nested business transactions in series.
- *Collaboration Activity*: The activity of executing another business collaboration within the scope of the current business collaboration.

In the ebBP technical specification, business collaborations can be divided into two sub-categories namely Binary and Multiparty Business Collaborations according to the number of involved business partner roles. A binary business collaboration requires exactly two abstract partner roles involved. On the contrary, a multiparty business collaboration involves more than two roles.

The choreography placed in a business collaboration specifies the ordering and transitions between a set of enabled business transactions. The usage of choreography is analogous to the activity diagram in UML context. The ebBP specification supports the visualization of the choreography in BPMN standard but does not mandate to do so.

Within the scope of the ebBP specification, a choreography can be devised by using the following constructs similar to the ones found in UML activity diagrams;

- *Start, Success and Failure States*: The initial state of a business collaboration is the Start state which links to a business transaction activity. The ebBP specifies two completion states for a business collaboration namely Success and Failure states. Completion states give the possibility to define whether a business collaboration has been performed as it was planned.
- *Direct Transitions*: Unless a business collaboration reaches one of its completion states, it is either in the state of performing a business activity or preparing to start a business activity. Once a business activity completes, the execution of the business collaboration navigates to another business activity.
- *Fork, Join and Decision Gateways for Transitions*: There are two types of the Fork gateway namely XOR and OR Forks. In XOR, only one transition from one business state to another is allowed although initially all business state transitions are possible. Whenever, one of the transitions is activated then all other are deactivated. An OR Fork may enable all transition paths in parallel and it does not specify the order

in which condition expression on a transition coming from a Fork will be evaluated. On the other hand, a Decision Fork selects only one transition path at start. This property differs it from the XOR Fork. A Fork gateway may have TimeToPerform element to specify the duration of the execution. At the end of the time interval given in the TimeToPerform element, the state of the business collaboration is moved to the corresponding Join. Join gateway has waitForAll attribute in order to indicate whether all transitions coming into it must be executed for the collaboration or not. AND-Join and OR-Join can be created by setting the waitForAll attribute to true or false respectively.

- *Guards on the Transitions:* Transitions may have guards to gate the navigation from one state to another. A guard stands for the status of an activity from which the transition originates. Guards can be ProtocolSuccess, AnyProtocolFailure, RequestReceiptFailure, RequestAcceptanceFailure, ResponseReceiptFailure, ResponseAcceptanceFailure, SignalTimeOut, ResponseTimeOut, Failure, BusinessSuccess, BusinessFailure and Success.
- *Variables and Condition Expressions:* Transitions can have a conditional expression element depending on variables. Variables which are named information elements are bound to concepts across business transactions.

3.1.2 Business Transactions, Transaction Activities and Business Document Flow

A business transaction is an atomic unit of work conducted between two collaborating parties playing opposite abstract roles. Abstract roles are generic and labeled as Requesting and Responding roles. In general, a business transaction is realized as a business document flow between requesting and responding roles. Business transactions achieve and support enforceable transaction semantics and state alignment between collaborating parties. Business signals can be enabled and used as a part of a message exchange in business transactions so as to ensure state alignment of the respective parties.

A business transaction consists of a Requesting Business Activity, a Responding Business Activity, one or two business document flow between them and several optional business signals. The abstract partner roles which are Requesting and Responding roles perform requesting business activity and responding business activity respectively. In business transactions, a requesting document flow is mandatory and the responding document flow is specified

whenever it is required.

ebBP brings eight main business transaction patterns which determine the exchange of business documents and signals to achieve the necessary business transaction;

- *Commercial Transaction*: Represents formal obligation between parties.
- *Notification*: Represents business notifications such as a failure or status order.
- *RequestConfirm*: Used where a confirmation about the status with respect to previous obligations or a responder's business rules is required by the requester.
- *RequestResponse*: Specifically used when the request for business information requires a complex interdependent set of results.
- *InformationDistribution*: Used for informal information exchange between parties.
- *QueryResponse*: Used when the requester party want to query for an information that the responder has.
- *DataExchange*: Extensible pattern for partner-specific data exchange and business transaction patterns.
- *Legacy Business Transaction*: This pattern is not recommended for using in concrete business transactions and retained for conversion only with previous versions of the ebBP specifications.

The main responsibility of a business transaction activity is to perform a business transaction within a collaboration. Business transaction can be associated to any number of business transaction activities which means that the same business transaction is subject to same or different business collaboration with different business transaction activities.

A business transactions is designed as a reusable protocol which can be referenced by business collaborations through the use of business transaction activities. In business transaction activities, specific and concrete roles of the business collaboration are bound to the generic partner roles in business transactions. An external role in a business collaboration is mapped to the role defined in the enclosed business transaction by using the "Performs" element found in the business transaction activity.

Business transaction activities can be nested in a recursive manner by the "Complex Business Transaction Activity" element defined in the ebBP specification. Complex business transaction activities execute the transaction activities in series and express the situation

occurs when a transaction activity can happen only after the request of the other transaction activity has been entirely processed. In this type of transaction activity, the nested business activities have a “statusVisibility” element to specify which state of the associated transaction are visible by the parent complex transaction activity.

Business transactions provide additional semantics that configure the particular performance of the referred business transaction. These semantics can be considered as the rules and the configuration parameters required for software components to realize the business transaction in a predictable and deterministic way. The following parameters are supported to substantiate and enforce preconditions on the business transaction activity;

- *Reliability* is the ability to specify reliable document and signal delivery.
- *Document Security* refers to authorized, authenticated, confidential and tamper detectable transactions.
- *Non-repudiation* specifies the keeping of transaction artifacts to aid in legal enforceability.
- *Authorization* refers to authorization requirements for the parties performing roles.
- *Predictability* refers to clear roles, precise transaction scope, understood time bounds, unambiguous determination of completion and business information semantics.

However, how these parameters are reflected to the implementation is not specified within the scope of the ebBP.

The software counterpart of the business transaction are business service interfaces which manage the business transaction, monitor the timers and requirements of the business collaboration, and enforce the semantics.

A business document flow is modeled indirectly as a “Document Envelope” associated with one requesting or responding business activity. A document envelope is sent by one role and received by the other in a business transaction. There is always a single document envelope for a requesting activity and may be zero or more for a responding activity.

Each document envelope encapsulates a business document with its attachments. Although a document envelope can refer to a logical business document, it defines neither the structure of the document nor the underlying semantics.

3.1.3 Business Signals and Exceptions

Business signals can be exchanged during the execution of a business transaction in order to inform the collaborating parties about the state alignment of the business collaboration explicitly calculated at run time. Business signals are computed by the collaborating parties and provide a mutual understanding of the business activity.

There are two important business signals namely “Receipt Acknowledgement” and “Acceptance Acknowledgement”. Business transaction pattern specifies whether a “Receipt Acknowledgement” and/or and “Acceptance Acknowledgement” signal is required.

- *Receipt Acknowledgment* signals that a request or response message has been properly received by the business service interface. This type of signal is necessary for reliable messaging between collaborating parties.
- *Acceptance Acknowledgement* signals that the received request or response message is subject to business processing and that processing has been completed successfully by the receiving party. This type of signal is used extensively for the successful synchronization of state between collaborating parties.

Business signals are different than the business messages. They have a fixed structure defined in the ebBP signal schema while the content of a business message can vary both at run-time and over time, and is under control of an application or service.

There are simply two causes of failures occurred during the execution of a business transactions; timeouts and exceptions. Since business transactions may be time-critical operations, they should have a distinct time boundary. The timeout parameters are normally associated with the response and each of acknowledgement business signals. After timeout occurs, the transaction must be set to null and void. On the other hand, the processing of the transaction cannot be completed successfully by the request in or responding role. In such cases, a series of protocol exceptions are used to indicate the failure.

3.2 The ebBP Editor

3.2.1 Introduction

The ebBP Editor is an open source tool designed for domain experts helping them in creating, modifying and validating ebBP instances in a user friendly way. It is implemented as a part of the IST 027065 RIDE (A Roadmap for Interoperability of e-Health Systems in Support

of COM 356 with Special Emphasis on Semantic Interoperability) project funded by the European Commission [64]. The author of this dissertation work actively participated in implementation of all components of the ebBP Editor, prepared a distribution release and presented it to the OASIS ebXML Business Process Technical Committee. Minutes of this presentation can be found in [23]. Within the scope of this thesis work, the editor tool is enabled for describing electronic business process definitions in a standard-based manner. These definitions are supposed to be exploited in the process of automatic service ontology generation.

The ebBP Editor is composed of two main components; XmlStylist and Visual Component. XmlStylist is used for creating, editing and validating process specifications conforming to the XML schema definition of the ebBP. GUI of the XmlStylist displays process specifications, business transactions, packages and the corresponding specification document in a tree structure. Visual Component displays Business Collaborations within a process specification by enabling BPMN standard. Each Business Collaboration is represented in a different section graphically.

The latest version of the ebBP Editor is available at the sourceforge.net portal. The necessary source codes and third party libraries are made accessible and downloadable by general public licence. The software requires a Java Virtual Machine and Apache Ant build tool in order to build and run the project. The editor tool has been tested and verified with Sun's JDK built 1.5.0_04 on Microsoft Windows XP platform.

In order to start the ebBP Editor, the user should first compile the source codes with the command "ant build" and then type the command "ant run" to the operating console. These commands are defined in the *build.xml* project file which can be found at the project folder. The details of the build mechanism is given in the Apache Ant's web site.

After starting the editor, user can create a new ebBP Process Specification or open an existing one by clicking on the respective menu bar buttons of the GUI. Both of these actions are done with the assistance of the XmlStylist. User can edit the details of a business process through the visual features supported by the XmlStylist. After closing the popup window of the XmlStylist, the modified business process definition is validated syntactically. User is informed whenever there is any problematic structure provided within the process definition. Validated process definitions are then visualized in the Visual Component of the ebBP Editor according to the BPMN standard. For each business collaboration given in a process specification, a tabbed pane is generated to visualize the business choreography i.e. display the interaction between the business activities and the business roles taking part in

the collaboration. A screenshot of the ebBP editor is depicted in Figure 3.1.

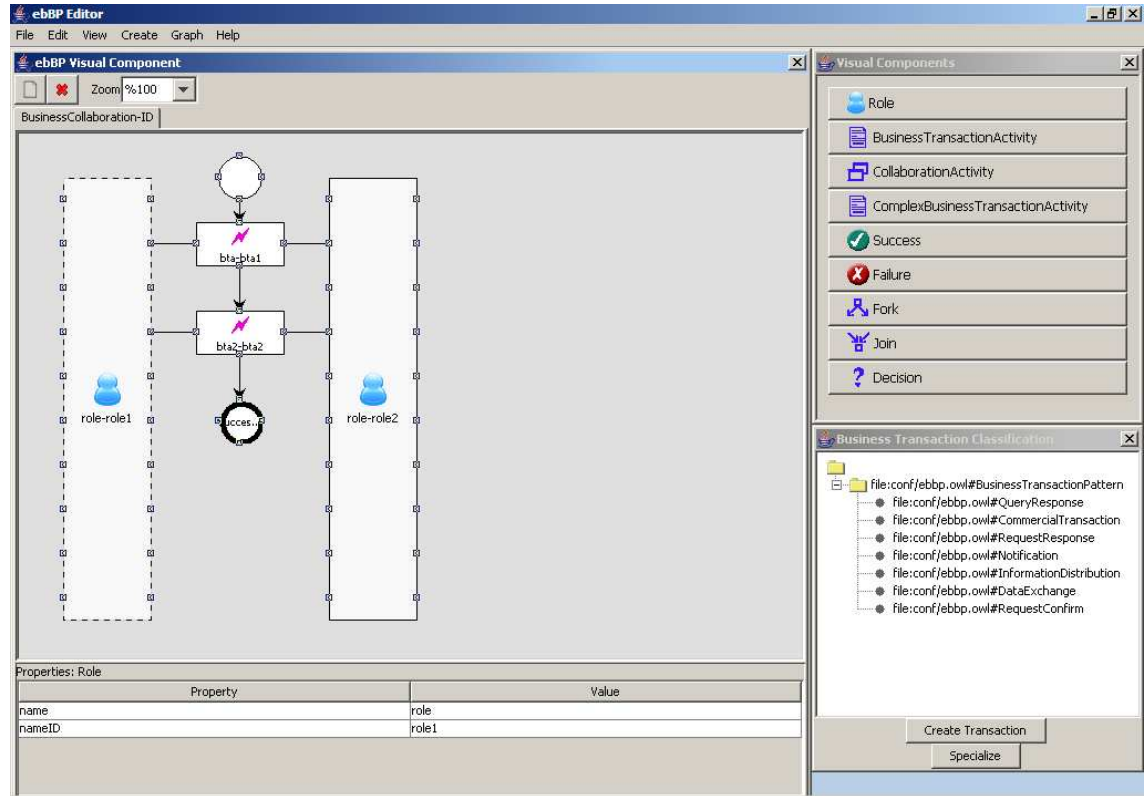


Figure 3.1: A screenshot of the ebBP Editor

3.2.2 Overview of the ebBP Editor Components

XmlStylist

XmlStylist component is designed for fast and easy editing of ebBP instances through their tree-like representations. Basically, XmlStylist parses a specified Xml schema definition and builds a user-editable form dynamically. The user can then fill attributes and fields of this form in order to create an Xml instance which conforms to the specified Xml schema. User is provided with the cardinality information of the attributes and elements. XmlStylist performs a validation check before serializing the graphical form into its Xml representation.

As it can be seen in Figure 3.2 the main window of the XmlStylist is divided into three subwindows. In the top-left window, the given Xml schema is visualized as a tree object. Below of this window, the ebBP instance which is being edited is given as tree object similar

to its schema representation. In the right of these two windows, the details of the selected ebBP element is visualized with its attributes and subelements.

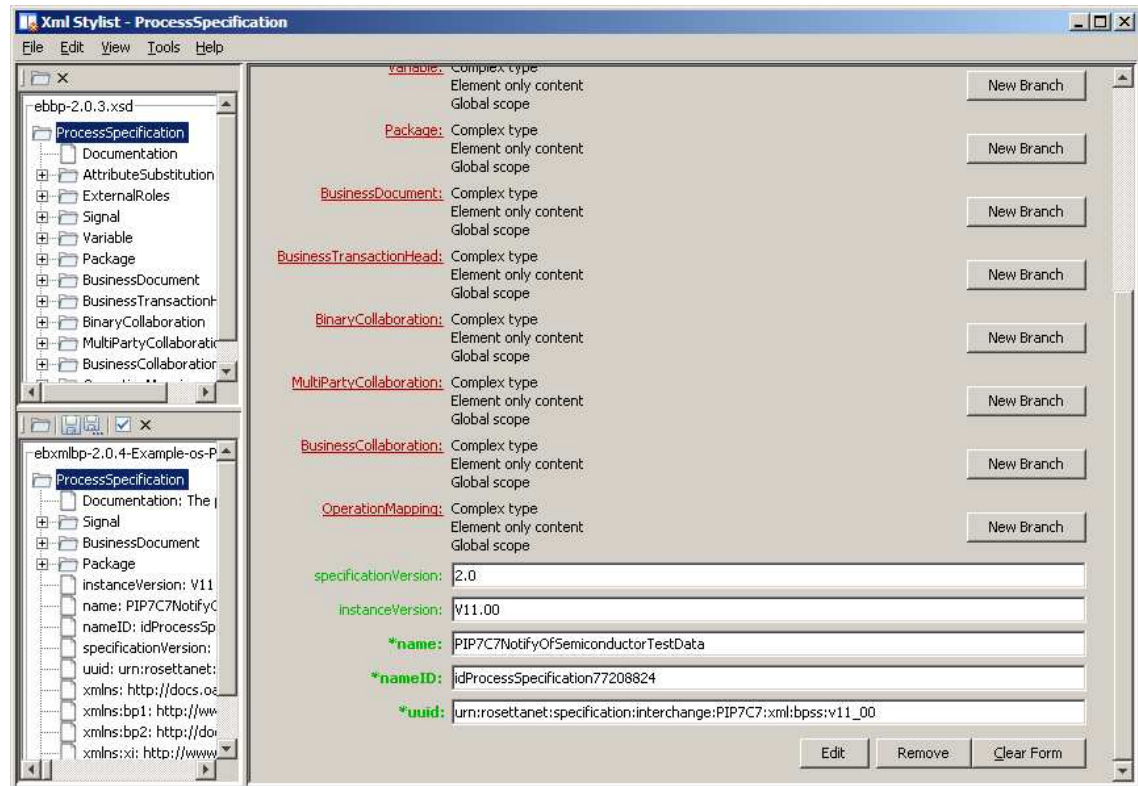


Figure 3.2: XmlStylist - Main Screen

As it is mentioned before, XmlStylist parses a specified Xml schema and provides the user with user-editable visual forms which are created dynamically. The user can create instances of the schema by filling these visual forms. When the user loads an Xml schema, the XmlStylist informs him/her whether the schema contains one or more global elements. If the schema contains more than one global elements then the user should specify one of them to be the basis of the Xml instances which will be created in future. This is necessary because according to the Xml standard, one Xml document has to contain only one global element. The user can select the root element from the menu shown in Figure 3.3.

In order to add a new element to the Xml instance, the user first selects the corresponding parent element from the Xml instance tree and then he/she selects the interested subelement from the schema tree. After selecting parent element from the instance tree, the schema tree is updated and all child elements of this parent element is listed to the user. The user then

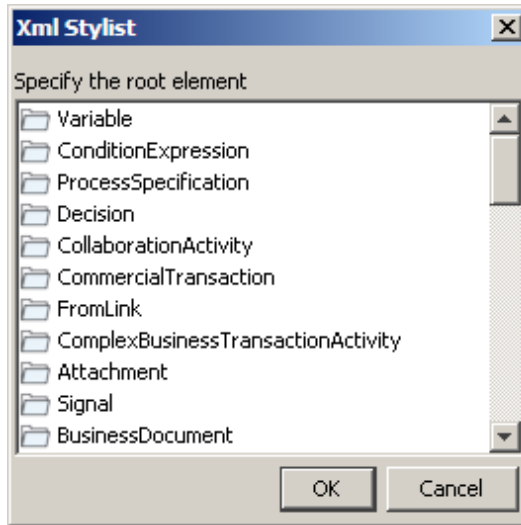


Figure 3.3: XmlStylist - Select Root Dialog

specifies the subelement to be added from the schema tree. The necessary form for the selected subelement is dynamically generated at the right part of the XmlStylist. Editable attributes of this elements are visualized as blank text boxes within this form. After filling these text boxes according to their cardinality restrictions, user can add this element to the Xml instance. The user-editable schema form is dynamically created according to the following simple rules;

- Attributes of the selected element are represented with text boxes.
- Simple subelements of the selected element are shown as text fields if they have a maximum cardinality of one or else they are represented with combo boxes.
- Complex subelements are represented with a link to their own user-editable forms. When the user clicks on this link, XmlStylist creates the necessary visual form and allows the user to specify this complex subelement.
- The required attributes and fields are represented with additional visual properties in order to inform the user. For example, an asterisk precedes the label of a required field and they are written in bold fonts. Moreover, a field can have a tooltip which gives additional information about the properties of that field such as namespace, scope and cardinality.

On the other hand, global elements can be freely added to a blank Xml instance without specifying any parent-child relationship.

Whenever an element from the ebBP instance window is selected by the user, the schema window and the user-editable form is updated accordingly. The visual form is filled with the values of the selected ebBP element and then the user can then edit these fields or remove the element from the ebBP instance document.

XmlStylist validates ebBP instances whether they conform to the ebBP standard by the help of its enhanced control mechanisms;

- Before adding a new element to the ebBP instance, parent-child relationship is checked. Based on the specified Xml schema, XmlStylist does not allow misplaced elements in an Xml document.
- After each update of an Xml element, XmlStylist validates whether the required fields of that element are fulfilled.
- Before serializing the tree representation of the ebBP instance into its Xml format, a full validation of the instance takes place.
- Whenever an erroneous document construct is figured out by the validation process, the user is informed about this situation.

A sample warning message produced after the validation process of an ebBP instance is shown in Figure 3.4.

ebBP standard definition introduces a document construct called *Package* in order to foster reusability among ebBP instances. Packages can be freely imported into a process specification or into another package. In XmlStylist package importing is supported also. Users can import packages through the *Import Package* menu item under the *File* menu.

ebBP Visual Component

ebBP Visual Component provides graphical representation of business choreographies among business partners. In ebBP specification, these choreographies are defined within the scope of business collaborations. In this respect, ebBP Visual Component parses a given process specification and visualizes all business collaboration stored in it. Each business collaboration is displayed in a separate tab section of the graphical pane. ebBP Visual Component exploits the BPMN standard for visualizing the business collaborations. This component is in coordination with the XmlStylist i.e. they work on the same process specification.

A new process specification can be created or an existing one can be opened by the clicking on the “New” and “Open” menu items respectively. They are listed under the “File” menu. A

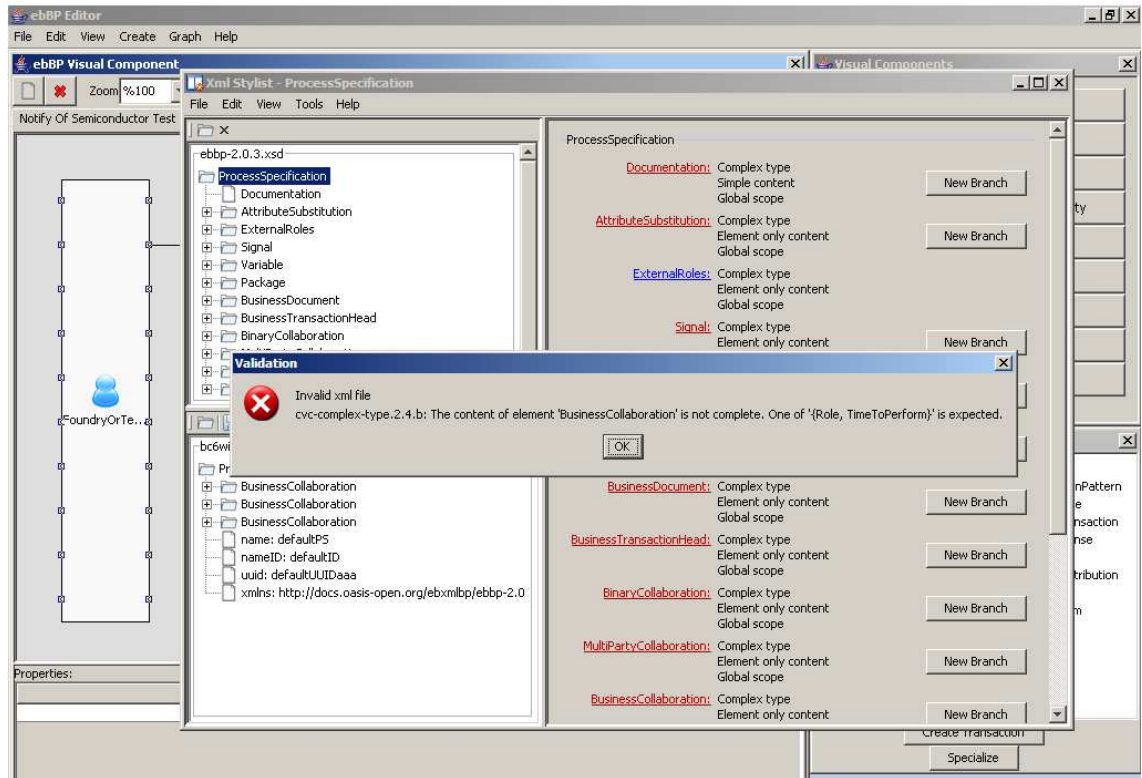


Figure 3.4: XmlStylist warns the domain expert about an invalid Business Collaboration instance

new business collaboration can be created by clicking on the “New Business Collaboration” icon on the ebBP Visual Component. Details of the newly created business collaboration can be modified through XmlStylist. Business collaborations can be deleted by clicking on the “Remove Business Collaboration” icon placed next to the “New Business Collaboration” icon.

Main visual constructs used in representing business collaborations are given in the graphical components window (see Figure 3.5). User can drag and drop these constructs on a business collaborations in order to add them to the definition by the help of the XmlStylist.

When the user clicks on of the visual constructs given in the business collaboration representation, the basic properties of the element are displayed in the bottom part of the ebBP Visual Component. Users are allowed to edit those properties by the help of this section in a more faster way.

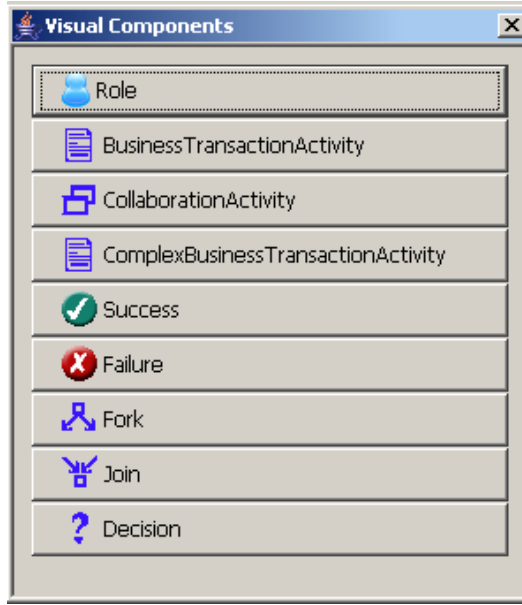


Figure 3.5: Graphical Components of the ebBP Editor

3.3 Mapping Business Collaborations to Web Service Process Models

Core components of the ebBP standard schema with the editor tool addressing the challenges for creating and modifying ebBP instances have been introduced in the previous section. In order to enhance our generative method for service ontology creation with process model semantics of web services, we have enabled the business collaboration and transaction of the ebBP and devised conceptual mappings for automatically transforming these ebBP constructs into OWL-S standard. The transformation method is implemented as a part of the open source GENODL project which is introduced in Chapter 4.

3.3.1 Motivation Behind the Transformation Method

The ebBP brings the notion of business service interface (BSI) as a logical definition for a collaborating party's actions exposed as business service and supports its implementation through web services and other software technologies. A BSI is compromised of a set of business processes, business object states of a business process and the rules governing transition between those states. In other words, BSI is the logical set of transactions required to achieve a common goal. The interface to the BSI is through business messages and signals like in message-oriented web services. Moreover, BSI is capable of providing non-functional

requirements such as quality of service and service configuration parameters. In the context of the ebBP, BSI is completely separated from implementation technology. Implementation choices are not specified and may include Java beans, web services etc. In brief, a BSI consists of the following entities;

- A discrete set of business process states shared and aligned between collaborating partners.
- A discrete set of business transactions and transitions between business transactions.
- Business rules and requirements governing the states, transactions and transitions.

Bearing in mind these properties, the ebBP provides domain experts with state-of-art modeling capabilities for encoding necessary transactional semantics among the collaborating systems that are surely subject to web service technology stack. However, the ebBP does not mandate or specify any refinement mechanisms for transforming the underlying semantics of an ebBP document into a more convenient structure that any implementation technology such as web services can realize them easily. Hence, there is a gap exists between business process modeling with ebBP and service-oriented development.

What we have achieved in this work is to develop conceptual mapping schemes between the generic ebBP instances and OWL-S ontologies. OWL-S is an emerging de-facto semantic web standard that supports automation of various web service related activities such as service discovery, composition, execution and monitoring. OWL-S provides a standard language for describing process models of atomic as well as composite web services. The role of OWL-S in bridging the gap between domain and application engineering while developing service-oriented systems is pictured in Figure 3.6.

As an outcome of this mapping, previously defined business processes are refined and brought one step closer to the realization phase automatically. Although it is important to note that a process is not a program to be executed, the proposed transformation will lead to a further point in the way of end-user driven development for software engineering. Moreover, mapping ebBP definitions to OWL-S fosters service reuse. Once the OWL-S model of a business service interface is described then the service implementation may be discovered from the existing assets instead of developing the service every time from scratch.

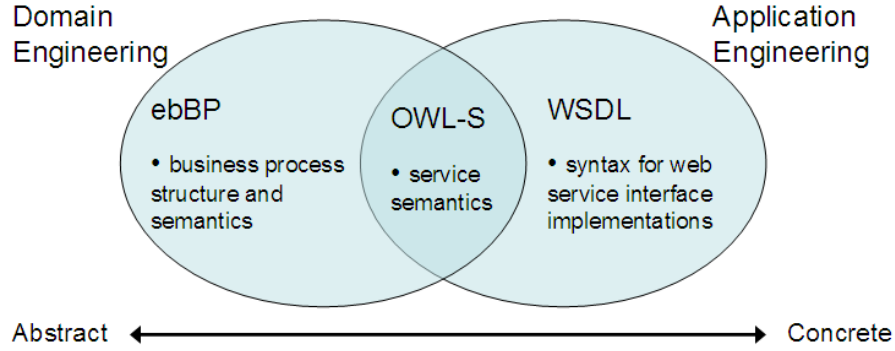


Figure 3.6: Bridging the gap between domain and application engineering in developing service-oriented system

3.3.2 ebBP to OWL-S Mapping

We can easily understand that there is no obvious one-to-one mapping between individual terms in ebBP and OWL-S process models. Because ebBP and OWL-S represent process decompositions from different abstraction levels as it is described in the previous subsection. When we compare the two standards, we treat the whole ebBP process specification as a model that may be decomposed into the corresponding OWL-S process models with some decomposition decisions depending on other parts of the model. A resulting service ontology can be related to web service implementation in three ways;

1. Provide requirements to the web service interface
2. Constrain implementation of the web service interface
3. Enable automatic discovery of semantically matching web services already implemented

Nevertheless, we put a basis for our mapping method through comparing common components sharing the similar semantics in both standards. Moreover, we adopt the ebBP's bottom up design approach for describing business collaboration and propose our transformation strategy inspired on this approach as follows;

1. Transform Business Transactions
2. Transform Business Document Flow for Business Transactions
3. Transform Binary (Business) Collaboration re-using the mapped Business Transactions
4. Transform the choreography for the Binary (Business) Collaborations

5. Transform higher level Business Collaborations re-using the lower level Business Collaborations translated previously

A high overview of the mapping specification is depicted in Figure 3.7. Firstly, we treat business collaborations as web services that process at least one business activity. Since business transaction activities are atomic processes, we can associate them with the OWL-S's *Atomic Process* concept. Similarly, a *Composite Process* can be considered as equivalent to Complex Business Transaction Activity, Collaboration Activity or a business collaboration that provides a choreography among two or more business activities. A Composite Process is built by integrating at least two Atomic Processes in a choreography definition i.e. maintains the state information between process transitions. OWL-S provides definitions for reusing atomic processes like in the ebBP. OWL-S defines abstract atomic processes with its *Simple Process* element. A Simple Process can be realized in various Atomic Processes. Hence, Simple Processes serve the same purpose as the Business Transactions in the ebBP.

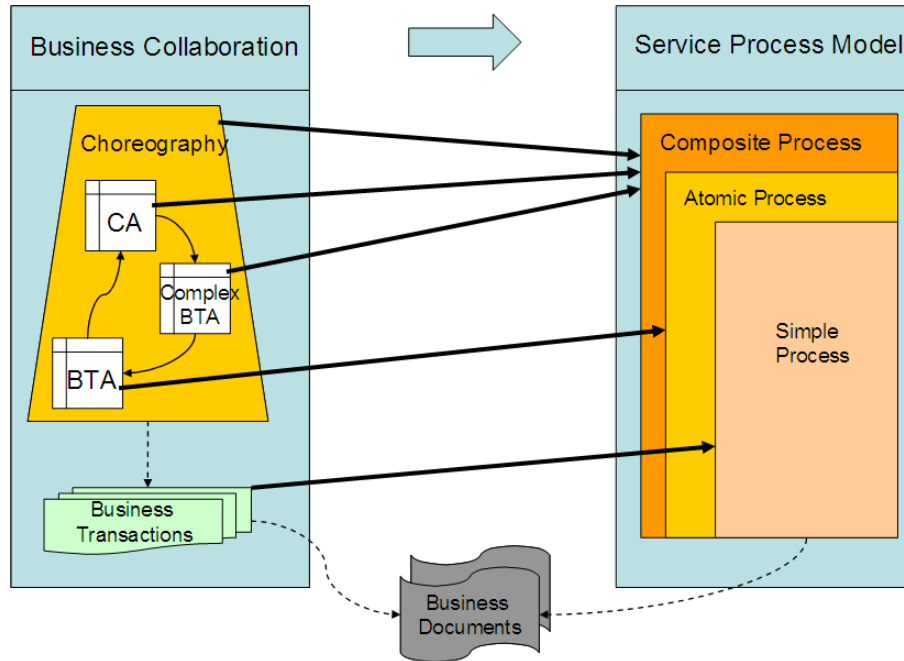


Figure 3.7: Overview of the mapping specification

In the following subsections, the conceptual mappings from ebBP elements to their OWL-S counterparts are listed. Note that the transformation method is intended for the direct creation of service ontologies from the given business process model. It is a self-contained

statement of core ebBP specification elements and relationships required to be able to create OWL-S compliant service process models.

Mapping Business Transactions and Business Document Flow

A business transaction can be considered as an atomic unit of work in a trading agreement between two collaborating parties playing opposite abstract roles. It consists of one or two predefined business document flows. Business transactions are pure reusable components of the ebBP. They can be associated within any business collaboration through business activities that set the proper performers for their declared roles.

Similar to business transaction, a simple processes is the abstract as well as the reusable part of the OWL-S process model. In the transformation specifications, a declared role for the business transaction is mapped to either *hasClient* or *performedBy* element of the simple process according to its initiating or responding position. A business transaction has exactly one requesting document flow while it may have a single responding document flow. Business document flows are represented as document envelopes. The mapping specification for the document envelope is described later in this subsection. One way of mapping business transactions to its OWL-S counterpart is shown in Table 3.1. The choreography of nested BTAs in a ComplexBTA can be transformed into a *Sequence* of *AtomicProcess* in terms of OWL-S. The general choreography mapping is given in the subsequent section.

Request and response document flows contain necessary business documents that pertain to the corresponding business transaction. In ebBP, a business document flow is not modeled directly instead it is modeled indirectly through document envelope structure. Each document envelope refers to a single business document and carries it over the document flow. A document envelope belongs to exactly one document flow.

For conceptual mapping, document envelopes can be modeled by the instances of the Input and the Output class of the OWL-S ontology based on whether the document flow is requesting or responding. A document envelope is defined as a property value of a document flow so as the *Input/Output* parameter of a message-oriented web service. As in the document envelope, *Input/Output* class refers to the structural specification or the schema definition of the business document. A possible mapping from document envelope to *Input/Output* is provided in Table 3.2. A document envelope can optionally have one or more attachments related to the business document. Although, *Input/Output* class does not provide an implicit support for this kind of attachments, a conceptual mapping may be satisfied by adding the necessary semantics to the document schema explicitly.

Table 3.1: Business Transaction to Simple Process

ebBP Term (BusinessTransaction)	OWL-S Counterpart (SimpleProcess)
/@name	/@name
/@nameID	/@rdf:ID
/RequestingRole	/hasParticipant
/RequestingRole	/hasClient
/RequestingRole/@nameID	/hasClient/@rdf:ID
/RespondingRole	/hasParticipant
/RespondingRole	/performedBy
/RespondingRole/@nameID	/performedBy/@rdf:ID
/RequestingBusinessActivity	/hasInput
/RequestingBusinessActivity/DocumentEnvelope	/hasInput/Input
/RespondingBusinessActivity	/hasOutput
/RespondingBusinessActivity/DocumentEnvelope	/hasOutput/Output

Table 3.2: DocumentEnvelope to Input (or Output)

ebBP Term (DocumentEnvelope)	OWL-S Counterpart (Input/Output)
RequestingB.A./DocumentEnvelope	SimpleProcess/hasInput/Input
RespondingB.A./DocumentEnvelop	SimpleProcess/hasOutput/Output
/@nameID	/@rdf:ID
BusinessDocument/Specification/@location	/parameterType
/Attachments	no direct mapping

Mapping Business Activities

A business activity in ebBP can be a business transaction activity, a complex business transaction activity or a collaboration activity. In order to give the essence of the overall conceptual mapping with principle notions, we address only the transformation requirements for the business transaction activity and complex business transaction activity in this study.

In general, a business transaction activity is an execution of a business transaction between specified collaborating parties. Business transactions are responsible for integrating business transactions to business collaborations. In the ebBP specifications, a business transaction refers to a business collaboration and sets the performers of the initiating and responding roles of the transaction with suitable business partners.

As it is stated before, business transaction activities can be associated with atomic processes in OWL-S. Atomic process corresponds to an action that a service can perform in a single interaction. This interaction should be executed in a single step by receiving exactly one message and sending zero or one response. Like business transaction activities, atomic processes can refer to simple abstract processes in a reusable manner. On the other hand, business activities is capable of defining business rules with the *BeginsWhen*, *EndsWhen*, *PreCondition* and *PostCondition* elements of the ebBP for annotation purposes. It is intended that the business service interface may use these elements at run-time whenever their expressions are coded in a machine-processable format. Atomic process's *hasPrecondition* and *hasResult* can be conceptually map to *PreCondition* and *PostCondition* respectively. Mapping specifications from business transaction activity to atomic process is given in Table 3.3.

Table 3.3: Business Transaction Activity to Atomic Process

ebBP Term (BusinessTransactionActivity)	OWL-S Counterpart (AtomicProcess)
/@businessTransactionRef	/realizes
/@name	/name
/@nameID	/@rdf:ID
/Performs	/hasParticipant
/Performs/RequestingRole	/hasClient
/Performs/RespondingRole	/performedBy
/PreCondition	/hasPrecondition
/PostCondition	/hasResult
/TimeToPerform	Handled in Choreography Mapping

A Complex Business Transaction Activity (ComplexBTA) allows for nested BTAs to happen one-by-one. This concept is a pure sequencing concept and does not affect the atomicity of the Business Transaction. When multiple activities are nested within ComplexBTA, these activities MUST be executed in series. The model supports for any number of nesting levels. The counterpart of the ComplexBTA in OWL-S domain is *CompositeProcess*. Mapping specifications from complex business transaction activity to composite process is given in Table 3.4.

Table 3.4: Complex Business Transaction Activity to Composite Process

ebBP Term (ComplexBTA)	OWL-S Counterpart (CompositeProcess)
/@name	/name
/@nameID	/@rdf:ID
/Performs	/hasParticipant
/Performs/RequestingRole	/hasClient
/Performs/RespondingRole	/performedBy
/PreCondition	/hasPrecondition
/PostCondition	/hasResult
/TimeToPerform	Handled in Choreography Mapping
<i>Sequence of BTAs</i>	Choreography Mapping for <i>Sequence</i>

Mapping Business Collaborations

In this subsection, we show how a business collaboration definition can be transformed into the OWL-S context. We first map common properties of the business collaboration element to matching attributes of the service class in the OWL-S ontology. Then, we transform the basic building blocks of the choreography described within business collaboration element into their counterparts in OWL-S enabled for constructing service process model.

In OWL-S, Service Profile provides high-level description of a web service. It can store human-readable properties (service name, text description, contact information) as well as machine-interpretable properties (inputs, outputs, preconditions and results). Basic properties of a business collaboration such as *name*, *nameID* and *Documentation* can be mapped to suitable service profile attributes of OWL-S. A possible mapping from ebBP's business collaboration to service class of OWL-S is given in Table 3.5.

Table 3.5: Business Collaboration to Service

ebBP Term (BusinessCollaboration)	OWL-S Counterpart (Service)
/@name	/presents/Profile/serviceName
/@nameID	/presents/Profile/@rdf:ID
/Documentation	/presents/Profile/textDescription
/Role/@nameID	/describedBy/CompositeProcess/hasParticipants
Choreography Transformation	Given in Table 3.6

According to the ebBP specifications, a choreography is an ordering of business activities within a business collaboration in order to specify which business state is expected to follow another state. Hence, a choreography definition removes any ambiguity in business document exchange among collaborating parties. We know that common control structures for establishing the choreography are *Start*, Completion (*Success*, *Failure*), *Transition*, *Fork* (OR-Fork, XOR-Fork), *Join* and *Decision*. Each choreography begins with a *Start* element and traverses a path through a graph until reaching a completion state. *Start* element has only one linking construct which is *ToLink*. By its *toBusinessStateRef* attribute, a *ToLink* construct refers to the next state where the current state can transition to. Conversely, a *FromLink* refers to the state where it can be transitioned to the current state via its *fromBusinessStateRef* attribute. *Start* element defines a special state that can only be transitioned from while completion elements such as *Success* and *Failure* cannot point any further state to transition. In general, linking constructs (*FromLink* and *ToLink*) should reference states in business collaboration (*Start*, *Success/Failure*, *Fork*, *Join* and *Decision*).

Basically, OWL-S's composite process consists of other atomic or composite processes. Control flow of a composite process is specified using control constructs which can be nested to an arbitrary depth. Like business collaborations in the ebBP, composite processes can be considered as state-oriented workflows. In the transformation method, *CompositeProcess* class is preferred for representing underlying semantics of business choreography within business collaboration. Main control construct of the whole choreography is the *Sequence* element. Linking constructs (*FromLink* and *ToLink*) are mapped according to their type and the class of the state they refer. For example, *FromLink* is transformed into a *ControlConstruct* that it can be further specified as a *Perform*, *Split*, *Split-Join* or *Choice* based on the type of the referred state; *Business Transaction Activity*, *Fork*, *Join* and *Decision* respectively. In ebBP, a choreography starts by linking to a business state so, we can associate *Start* with a *Sequence* instance whose *list:rest* element refers to the state that *ToLink* of the *Start* linking as well. Overall mapping from choreography construct of ebBP to their OWL-S counterparts are given in Table 3.6. *Transition*, *Fork*, *Join* and *Decision* have at least one *FromLink* and one *ToLink* but maximum occurrence of these linking constructs can vary depending on the choreography type. *Fork* and *Decision* include at least two *ToLink* on the other hand, *Join* have at least two *FromLink*.

Table 3.6: Choreography to Service Process Model

ebBP Choreography	OWL-S Counterpart (ControlConstruct)
/FromLink	/ControlConstructList/list:first/ControlConstruct
/ToLink	/ControlConstructList/list:rest/ControlConstructList
/Start	CompositeProcess/@composedOf/Sequence
/Start/@nameID	/Sequence/@rdf:ID
/Transition/@nameID	ControlConstruct/@rdf:ID
/Fork/@nameID	/Split/@rdf:ID
/Join/@nameID	/Split-Join/@rdf:ID
/Decision/@nameID	/Choice/@rdf:ID
/B.T.A	/C.C.L./list:first/Perform/AtomicProcess
/Success	No suitable match
/Failure	No suitable match

3.3.3 Limitations of the Transformation Method

The proposed transformation method has a number of drawbacks due to the limitations of the OWL-S specifications and the implementation of the mapping rules. Counterparts for some ebBP features such as exception handling, business signals, conditional expressions, document attachments, non-functional and configurational parameters are not directly supported by OWL-S. On the other hand, the implementation of the transformation method should cover the mappings of complex business transaction activities and collaboration activities. Those mapping rules are planned to be implemented as a future work. Moreover, the transformation method should be continuously updated according to newer version of the related standards.

CHAPTER 4

ADDING FORMAL SEMANTICS AND REASONING SUPPORT TO FEATURE MODELS

In order to utilize a product line approach which exploits feature-oriented domain analysis, there is a need for formal and machine-processable feature models. Feature models play a key role in the domain analysis process; they set the scope of the product line by capturing commonalities and variabilities among products, and provide a basis for future steps in product line processes such as defining common architecture, creating reusable system components etc. Therefore, representation of the feature models in a machine-processable way is considered first.

Moreover, it is necessary to verify a feature model before using it in further activities. After defining a formalism for feature models, it is possible to automate a verification process through reasoners which can check inconsistencies among features as well as automatically correcting them to a certain extent. On the other hand, automated analysis of the formalized feature models are not limited only with the verification operation. There is a number of reasoning problems that can be proposed along with verifying feature models such as checking model satisfiability and dead feature detection.

There is, however, no shared agreement on;

- how to represent and disseminate feature models in a standard way,
- how to analyze feature models automatically [7],
- how to localize features into reusable assets and tailor them for application engineering usage.

This chapter presents the Semantic Web approach for modeling and verifying feature-oriented domain analysis. It is clear that there are several opportunities to be gained in developing a formalization of feature models to make them semantic aware. OWL is decided to be used for feature model formalism because it is a well known standard in the area and is supported by various technologies, tools and development environments like Protege-OWL, SWRL and JESS. In this respect, we will exploit the OWL's richer semantic constructs as well as its reasoning capabilities. OWL reasoning engines such as JESS can be deployed to check for inconsistencies within a feature model and correct them automatically.

We first introduce the basics of our feature model ontology. We describe the feature model ontology by enabling OWL DL along with Semantic Web Rule Language (SWRL) for stating axioms. Then, we present our editor tool dedicated for creating, modifying and verifying the feature model ontology through its user friendly GUI. This editor tool has been completely developed within the scope of this academic study. The feature model editor exploits Protege-OWL API for handling ontology parsing tasks as well as the JESS rule engine for making inferences over feature models based on the predefined axioms.

4.1 Feature Model Ontology

In order to gain expressive power and enable automated operations over feature models, we exploit OWL ontology constructs. Firstly, we define a *Feature* class having two object properties, *hasParentFeature* and *hasChildFeature* respectively, which are transitive properties that are inverse of each other. These properties are required to express IS-A relations in structured view or OR relations in common feature model understanding. The concept of the feature model can be considered as an ordinary feature which has no parent feature.

In order to fully represent the mandatory and alternative relations, we derive two specific child classes from the *Feature* class. Each class is a subclass of the *owl:Thing* class. We assert that *Alternative Feature* and *Mandatory Feature* are mutually disjoint. *Feature* class encapsulates *isSelected* attribute which can be set to a boolean value indicating that whether the feature is enabled in a particular product. A mandatory feature must be selected if its parent is already selected and only one feature among its alternatives can be selected. A feature's alternatives can be specified by the transitive *alternativeOf* object property. Overview of the classes and their properties defined within the Feature Model Ontology is given in Figure 4.1.

$$Feature \sqsubseteq \top \quad (4.1)$$

$$AlternativeFeature \sqsubseteq Feature \quad (4.2)$$

$$MandatoryFeature \sqsubseteq Feature \quad (4.3)$$

$$AlternativeFeature \sqcap MandatoryFeature = \perp \quad (4.4)$$

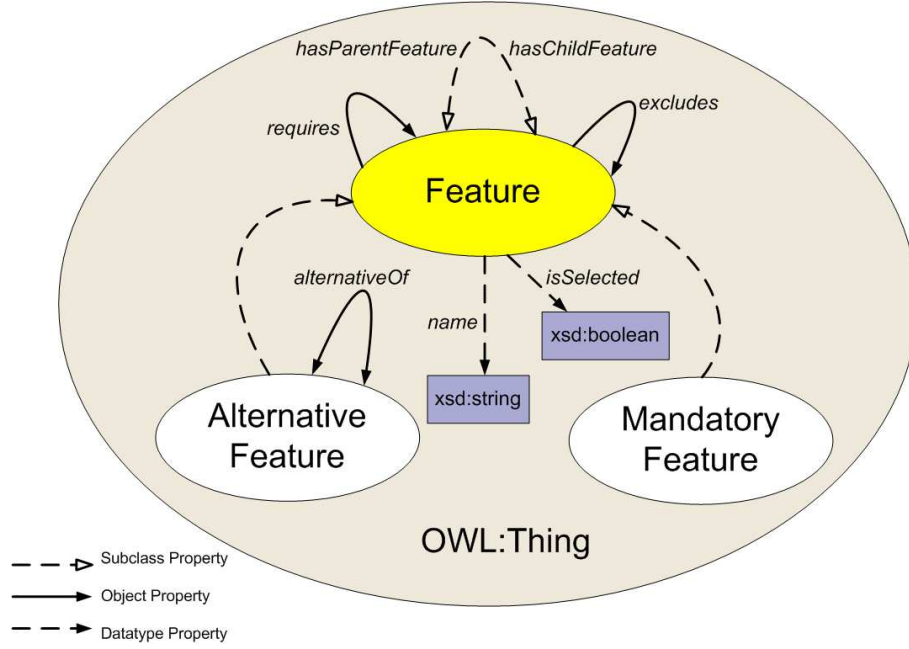


Figure 4.1: Classes and properties of the feature model ontology

A number of axioms for checking the consistency of the feature model customizations are formally defined within the scope of this work. In general, these axioms are exploited not only for verifying the feature model but also correcting any inconsistency found. These axioms are described as SWRL rules. SWRL plugin for Protege-OWL can translate these rules to JESS rule language in order to execute them in a JESS rule engine. Inconsistencies within a feature model can be automatically corrected after the reasoning has been performed and necessary inferences have been defined by the JESS engine. Protege-SWRL plugin is responsible for modifying the OWL document according to the changes inferred by the JESS. Defined axioms and their SWRL implementations are given below;

1. A feature cannot be selected unless its parent feature has been selected already. For all x and y;

$$Feature(?x) \wedge isSelected(?x, false) \wedge hasChildFeature(?x, ?y) \wedge \\ isSelected(?y, true) \rightarrow isSelected(?y, false)$$

2. A mandatory feature must be selected whenever its parent feature has been selected. For all x and y;

$$Mandatory_Feature(?x) \wedge hasParentFeature(?x, ?y) \wedge isSelected(?y, true) \wedge \\ isSelected(?x, false) \rightarrow isSelected(?x, true)$$

3. Only one feature must be selected among its alternatives. For all x and y;

$$Alternative_Feature(?x) \wedge isSelected(?x, true) \wedge alternativeOf(?x, ?y) \wedge \\ isSelected(?y, true) \rightarrow isSelected(?y, false)$$

4. A feature is selected whenever the other feature which requires it has been selected. For all x and y;

$$Feature(?x) \wedge requires(?x, ?y) \wedge isSelected(?x, true) \rightarrow isSelected(?y, true)$$

5. A feature is deselected whenever the other feature which excludes it has been selected. For all x and y;

$$Feature(?x) \wedge excludes(?x, ?y) \wedge isSelected(?x, true) \rightarrow isSelected(?y, false)$$

SWRL rules are normally stored as OWL individuals which can refer to the resources within the associated knowledge base. Class definitions of these OWL individuals are introduced in SWLR ontology. Main class of the SWRL ontology is the *swrl:Imp* which is used for defining a single SWRL rule. The *swrl:Imp* consists of two other classes namely *swrl:head* and *swrl:body*. Each of these classes is an instance of the *swrl:AtomList* class where a list of rule atoms are presented. A rule atom is represented via subclasses of the abstract *swrl:Atom* class. Another important SWRL ontology class is *swrl:Variable* that can be used for representing variables. Feature model ontology with encapsulated SWRL ontology is shown in Figure 4.2.

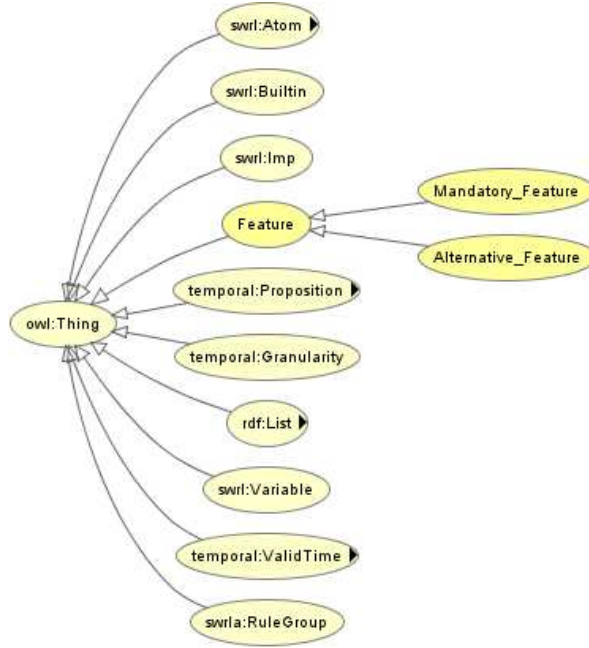


Figure 4.2: Classes of the feature model ontology with encapsulated SWRL ontology

Possible Reasoning Operations for the Automated Analysis of Feature Model Ontology

Feature Models are subject to automated analysis since it was reported in FODA but currently, there is no common understanding for what kind of operations should be supported or how they can be implemented. We have already identified automated verification and correction operations and here, we list other important reasoning problems identified previously in [7] as follows;

- *Determining feature model satisfiability:* Satisfiability problem is about finding any suitable instances that is not inconsistent with the given TBox. We can claim that a feature model is satisfiable when there is at least one product which can be derivable from the given feature model. In general, a feature model without any cross-tree constraints (requires, excludes) cannot be unsatisfiable.
- *Finding a particular product:* A satisfiable FM will constitute one or more products. With this operation, possible consistent products can be acquired from the FM by enabling customizations over features.
- *Calculating the number of products:* As the name implies, this operation returns the number of products that a FM can specify. This operation can exploit product finding

operation and can be used for deciding whether the FM is satisfiable since the number of products is greater than zero then we can conclude that the FM is satisfiable.

- *Dead features detection:* A feature that should not be included in any product customization can be categorized as dead feature. The automated detection of such features is subject to the FM analyzes also.
- *Variability Analysis:* This operation gives the ratio between the number of products of a feature model and the 2^n where n is the number of the features except the root concept. A big ratio represents a flexible product line whereas a smaller one represents a more strict one.
- *Commonality Analysis:* This analysis applied to a feature of a feature model gives the percentage of products where this particular feature is included.

4.2 Feature Model Editor and Reasoner

The feature model editor and reasoner is realized through the scope of an ongoing open source project called GENoDL (Automated Service Ontology Generator Tool based on Description Logics) (<http://sourceforge.net/projects/genodl/>). This initiative was started by the author of this thesis work. With its user-friendly GUI, GENoDL aims to support a rich user experience and joy of use in order to ease the domain analysis and service ontology generation tasks on behalf of the domain engineers. The tool takes feature ontology and business processes as input and refine them in order to compile possible service ontologies. However, for this section we limit ourselves with feature ontology editing and reasoning capabilities implemented as a part of the GENoDL tool. In the next chapter, GENoDL's automated service ontology generating feature is explained in details.

GENoDL requires a Java Virtual Machine (Java 5 or 6), Apache Ant and Protégé 3.4 Beta Build 506 installed before. In order to enable reasoning capabilities, JESS library must be downloaded and copied into the Protégé 3.4's OWL plugin directory. GENoDL's source codes can be downloaded from its source control repository. After download, user should first change the "protege.dir" property found in the *build.xml*, the project configuration file, according to the Protégé's installation directory. The tool can be started from console by first typing the command "ant build" to build the source codes and then typing the "ant run" to execute the binary codes.

4.2.1 System Design

The tool supporting our operations for feature model editing and automated analysis is constructed over a Model-View-Controller (MVC) pattern. This architectural pattern allows us to separate visualization, data modeling and data access concerns from each other. The third party components such as ontology parser and reasoning tool are integrated through their specific APIs. The class diagram of the tool is given in Figure 4.3.

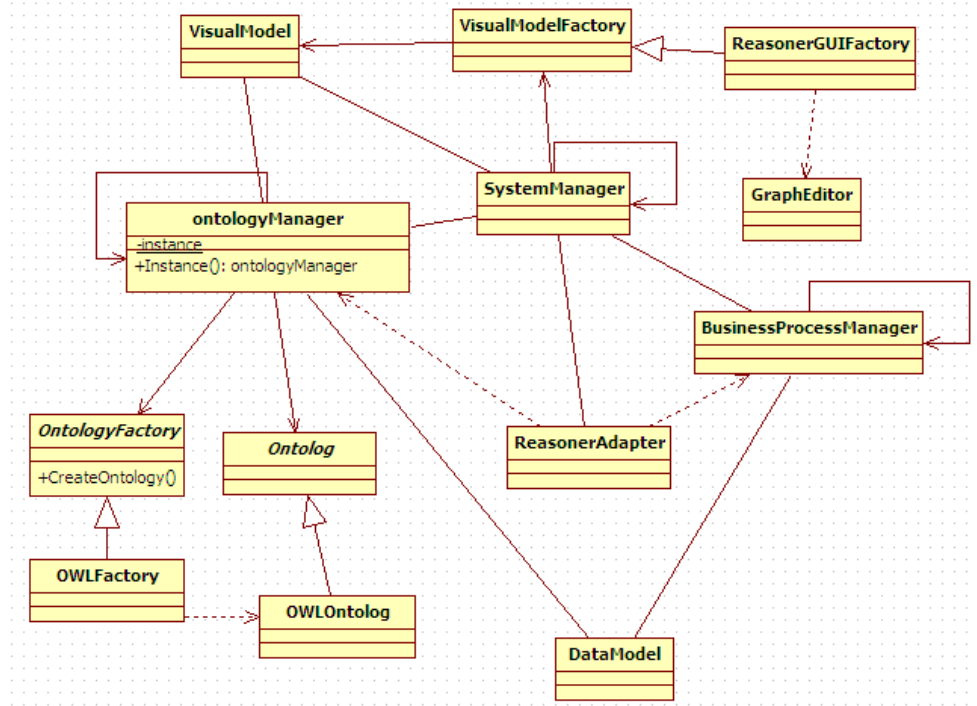


Figure 4.3: Class diagram of the GENoDL

The main class of the GENoDL is the *SystemManager* class. *SystemManager* is responsible for initiations of *OntologyManager* and *BusinessProcessManager* objects. These manager classes establish the control part of the MVC architecture. The design of each manager class conforms to Singleton design pattern where the instantiation of a class is limited to one object only. This pattern is useful in our case because we need exactly one dedicated manager object to coordinate the following separate tasks across the whole systems; system initiation, ontology editing and reasoning, and information extraction from business process definitions.

The model section of the architecture composed of two specific classes namely *DataModel* and *VisualModel*. Those two classes provide necessary access mechanisms to the resources

planned to be used by other classes in the control or view section of the architecture.

The graphical user interface of the GENoDL is produced by the *VisualComponentFactory* class. The graph editor for drawing feature model diagrams are mainly based on the example *GraphEditor* class provided by the JGraph software [40].

4.2.2 User Guide

The main window of the GENoDL is composed of three subcomponents (see Figure 4.4). At the top of the window, a menu bar is located for handling common file operations. Below of the menu bar, the rest of the window is divided into two panel. In the left side, feature model editor and its toolbar is placed and in the right side the business process manager part of the GENoDL can be found.

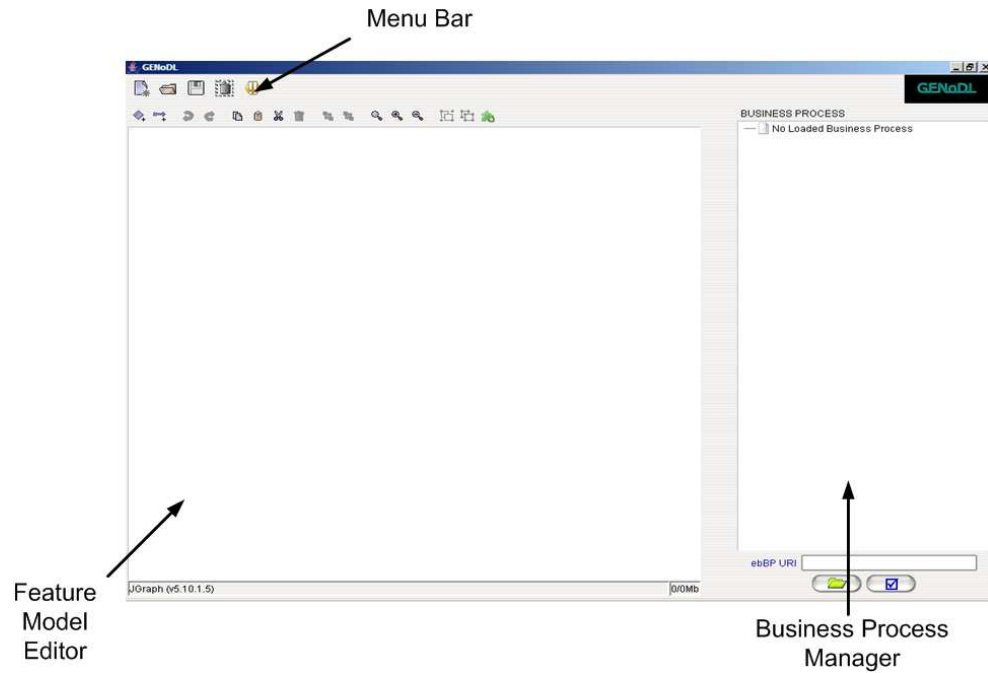


Figure 4.4: A screenshot from the GENoDL

Feature model editor of the GENoDL has drag and drop support for features and their associated relationship links. Moreover, visual components representing features are resizable.

New/Save/Load Feature Model

A new feature model ontology can be created by clicking on the "New" menu item which is placed on the menu bar. By creating a new feature model ontology, the default feature model ontology without any feature instance is loaded to the system by the Protégé-OWL.

A modified feature model can be saved by clicking on the "Save" menu item of the menu bar. As a result, the feature model is exported to its ontological representation as an OWL document. Saved feature models can be loaded to the editor through the "Load" menu item.

A screenshot of GENoDL's menu bar is given in Figure 4.5.

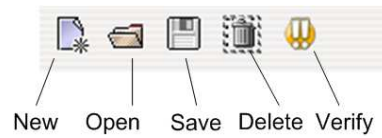


Figure 4.5: Menu items of the GENoDL

Add/Delete/Change Features

There are two possible ways to insert new features to a feature model. One method is by clicking on the "Insert" item on the feature model editor toolbar. A new feature is added to the top-left corner of the feature model. The toolbar of the feature model editor is shown in Figure 4.6. The other way is using the popup window of the feature model editor displayed

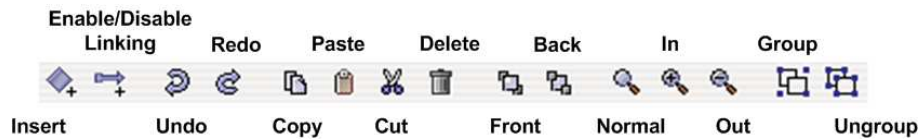


Figure 4.6: The toolbar of the Feature Model Editor

after pressing the right button of the mouse over the feature model panel. From the displayed popup window, user can select the "Insert" option. As a result, a new feature is created at the position where the mouse is currently pointing. Inserted feature is named by the "Feature_ n " convention where n is initially one and incremented by one after a new feature

is added. Similar to adding new features, deleting existing ones can be done through the

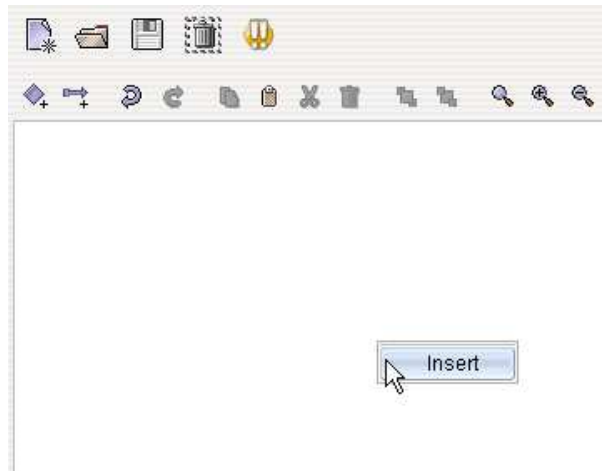


Figure 4.7: Inserting a new feature to the model

popup window or the specific toolbar icon. Moreover, user can delete a feature by using the shortcut key; "Delete" on the keyboard. Before deleting, feature or a group of features must be selected from the graph editor.

Properties of a feature can be changed through double clicking on it or pressing on the right mouse button over the feature and then selecting the "Edit" option from the popup window (see Figure 4.8). A new popup window follows the user's edit request. From this

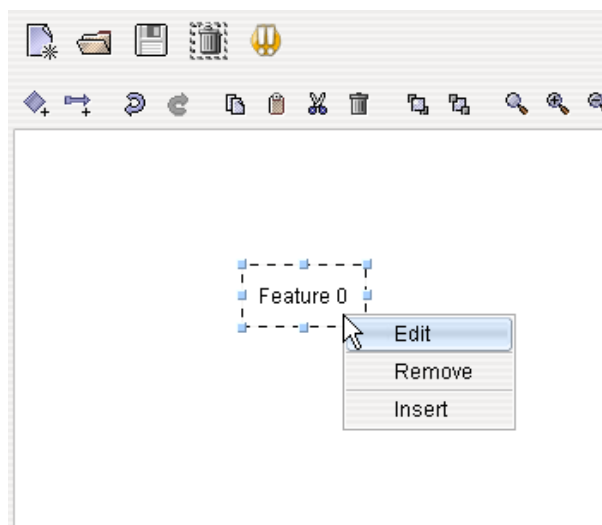


Figure 4.8: A feature can be edited or deleted by double clicking on it

new popup window, user can change the name of the feature as well as he/she can include the feature to the feature model customization through checking the "Select" box. There is a disable part in the editing popup window which corresponds to links that specify relationships among features. Popup menu for editing feature properties is depicted in Figure 4.9.

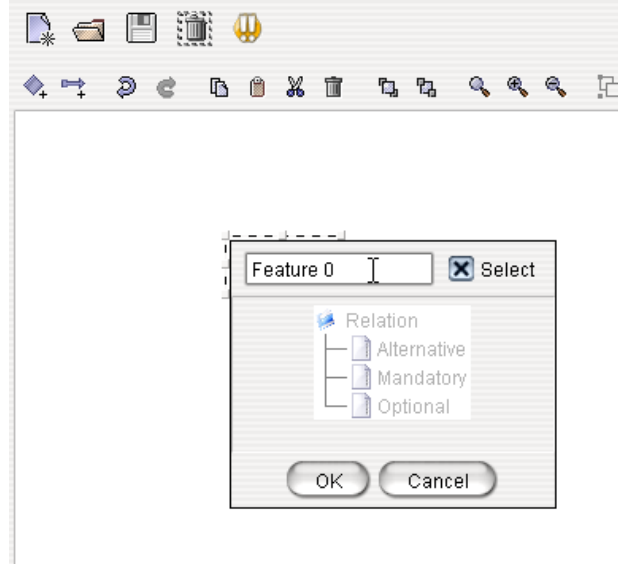


Figure 4.9: Popup menu for editing properties of a feature

Specify Feature Types

The rational behind using a feature model editor is to organize features under a root concept and to specify relationships and constraints among features in a graphical way. In this respect, we provide a simple method for specifying feature types. First of all, user decides on a parent and its corresponding child feature from the graph editor. Then, user points the center of the parent feature where the mouse pointer is changed its shape. At this moment, user can draw an arrow beginning from the parent feature to the child feature by holding the left mouse button until pointing to the center of the child feature. By default, "Alternative" property is assigned to the child feature as a result of drawing a link between parent and child features.

In order to change this property or edit it, user right-clicks on the link and selects the "Edit" option from the popup window. There is neither a name nor selection property of a link so, user can only change the link's type from this popup window. Please note that

this section is disabled for editing features. User can change a relation or feature type to Alternative, Mandatory or Optional from this section. In order to create a group of features that are alternative of each other, user expands the Alternative tree node and specifies a feature to be alternative of the current feature. Alternative relation is a transitive property so, it is enough to do a single specification in order to add the feature to an alternative group or create a new group. Popup menu for editing feature types is given in Figure 4.10. If it

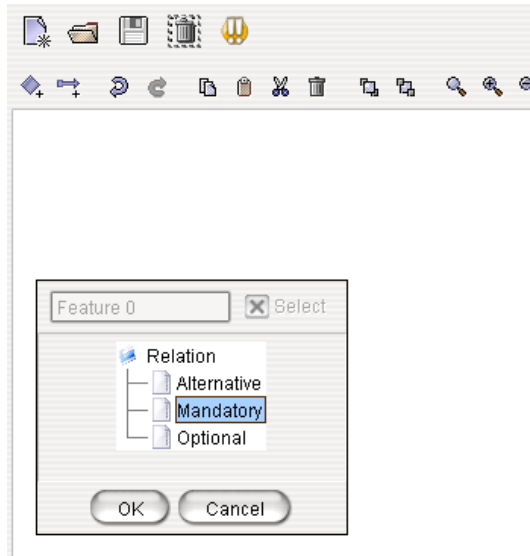


Figure 4.10: Popup menu for editing feature types

is not necessary to place relationships among features or user does not want to bother with drawing links mistakenly when doing drag and drops then the feature type specification can be disabled from the dedicated toolbar option. It can be enabled afterwards by a second click on the same toolbar icon.

Miscellaneous Editing Operations

Other actions that can be performed through the feature model editor's toolbar are listed as follows;

- **Undo/Redo:** User can undo changes that were executed so far and can redo a modification after an undo.
- **Cut/Copy/Paste Features:** Like in any word processor, our feature model editor supports essential editing actions on features.

- **Bring to Front/Send to Back:** During feature model editing, two features may be placed like one in front of the other. To prevent such blocking issues and provide a more coherent editing environment, users are provided with facilities to bring a selected feature to front or send it back.
- **Zoom In/Out:** User can change the scale of the graph by using the zoom in/out buttons on the graph editor toolbar.
- **Group/Ungroup:** To group features provides important simplicities during feature model editing such as easier drag and drop and protection from changes done mistakenly. A set of feature can be group after selecting them one by one or through drawing a selection box by mouse. User can ungroup those grouped features also.

4.2.3 Verification and Correction

Feature model, generated as an outcome of domain analysis, is customized in application engineering phase and specifically aligned to form a single product. The customized feature model has no variability points but a definite set of selected features. In order to verify a customized feature model represented with OWL instances, we employ JESS rule engine. Once the relationships among the features are described and formalized as SWRL statements, the implemented tool converts these statements to JESS rules by the help of the Protégé-SWRL plugin. At this stage, JESS rule engine can verify the interested feature model by checking each axiom among features described as rules. Whenever an axiom is not holding for a customization, the JESS engine can automatically infer the necessary corrections over this customization. The verification of a feature model can be invoked by clicking on the "Verify" button on the menu bar of the GENoDL.

Protégé-SWRL plugin provides necessary Java API called "SWRL Factory" for manipulating SWRL rule instances in a OWL knowledge base. Moreover, SWRL Factory is responsible for facilitating the mapping from OWL individuals representing the SWRL rules to associated Java objects. Each class described in SWRL ontology has a counterpart in SWRL Factory. For example, SWRL Factory supports *SWRLImp* and *SWRLAtomList* Java classes that can be used to mirror instances of the equivalent *swrl:Imp* and *swrl:AtomList* OWL classes.

Main concern domain of the SWRL rules is simply about the A-Box of the OWL knowledge base in terms of OWL classes and properties. Protégé-SWRL plugin has no reasoning capability however, it supports API level integration with existing rule engines such

as JESS. SWRL Factory provides integration functionality in its *SWRLRuleEngineAdapter* class. Thus, users can configure and use a rule engine over this bridge synchronized with the SWRL rule base and OWL knowledge base. User can load SWRL rules and OWL ontology knowledge base into the rule engine, execute these rules on the knowledge base and store inferred results back into the knowledge base.

In this work, we exploit the JESS environment which contains a rule base, a fact base and an execution engine. The execution engine associates rules with facts in corresponding bases. Rules may assert new facts or execute Java functions. In order to enable the JESS rule engine, Protégé-SWRL plugin performs three main tasks;

1. Represent OWL knowledge base as JESS facts
2. Represent SWRL rules as JESS rules
3. Invoke rule engine to perform these transformed rules and reflect the results of the inference in the OWL knowledge base

The necessary transformation methods from OWL and SWRL concepts to JESS constructs and vice versa are given in [52]. Once the OWL and SWRL concepts are transformed to JESS context, the execution engine can perform reasoning. The overall view of the integration between SWRL, Protégé-OWL and JESS rule engine is shown in Figure 4.11.

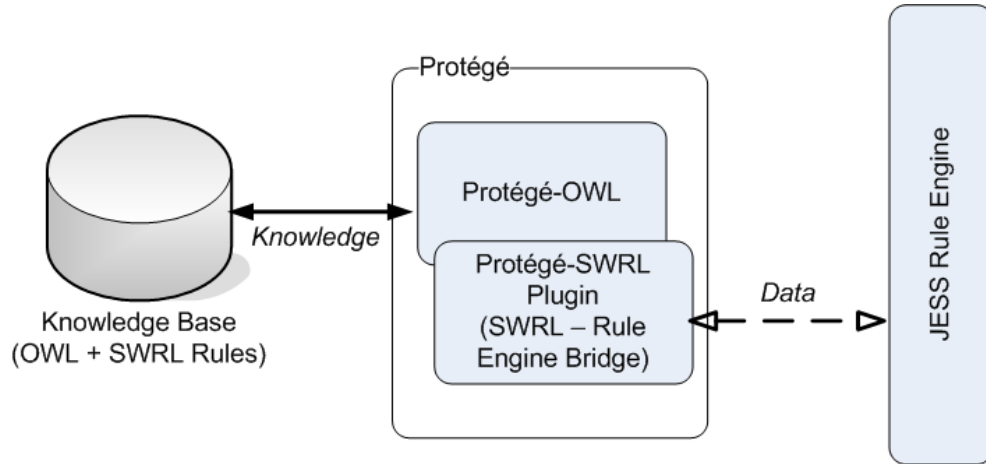


Figure 4.11: Integrating knowledge base with reasoning engine through Protégé-SWRL adapter

Within the scope of the *OntologyManager* class of the GENoDL, feature models can

be verified and corrected by using the Protégé-SWRL's *SWRLRuleEngineBridge* facility. We first initialize a new instance of it as a "SWRLJessBridge" and then the initialized SWRLRuleEngineBridge imports SWRL rules and OWL knowledge base from the feature model ontology. Implementation of the GENoDL and Protégé-SWRL integration is given below.

```
// Initialize a new bridge instance
SWRLRuleEngineBridge bridge = BridgeFactory.createBridge("SWRLJessBridge", featureModelOntology);
// import SWRL rules and OWL knowledge base
bridge.importSWRLRulesAndOWLKnowledge();
// Infer axioms and individuals...
bridge.run();
// Get inferred axioms after reasoning
Set<OWLAxiom> inferredAxioms = bridge.getInferredAxioms();
```

If there are any inferred axioms received after reasoning operation, then GENoDL gets the user's decision whether those changes will be applied automatically or executed one by one according to the user's preference for each inferred fact. An screenshot from the GENoDL is given in Figure 4.12 representing a correction for an inconsistent feature model in which a mandatory feature is not selected whereas its parent has been already selected. If the user clicks on the "OK" button on the dialog window then the feature model is updated and the child feature is selected automatically.

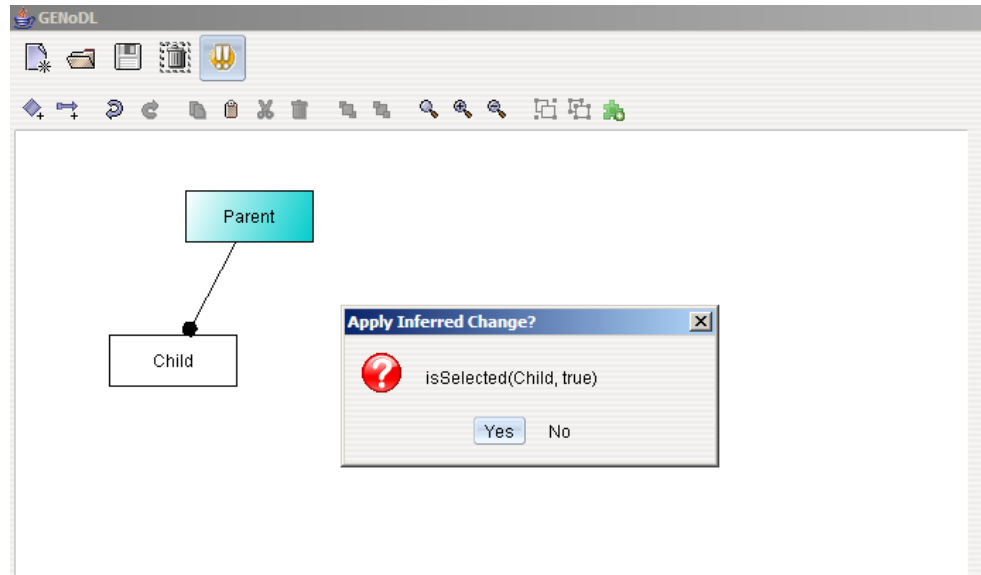


Figure 4.12: An inferred correction for the inconsistent feature model

4.2.4 Performance Evaluation

For the performance analysis, we first describe five feature models with 5, 9, 13, 17 and 21 features respectively, and represent them as OWL ontologies. The first feature ontology has four variability point and variability points are increased by four for each subsequent feature model. A variability point indicates an OR or Alternative relation between features. The experiment is conducted through the reasoning operation; verifying feature model. In a computing environment of 1.8 GHz Intel(R) Pentium Core2 Duo(TM) with 1 GB RAM, the time for performing this operation over sample inputs by JESS are comparatively depicted in Figure Figure 4.13. As it can be seen in the chart, the reasoning performance is closely related with the number of features in the feature model. This is because of the *Rete* algorithm which JESS implements. The complexity of the inference operation can be generalized as the order of $O(RF)$, where R is the number of axioms and F is the number of features on the working memory. The performance dramatically decreases as the number of features increases. The detailed analysis for Rete algorithm can be found at [39].

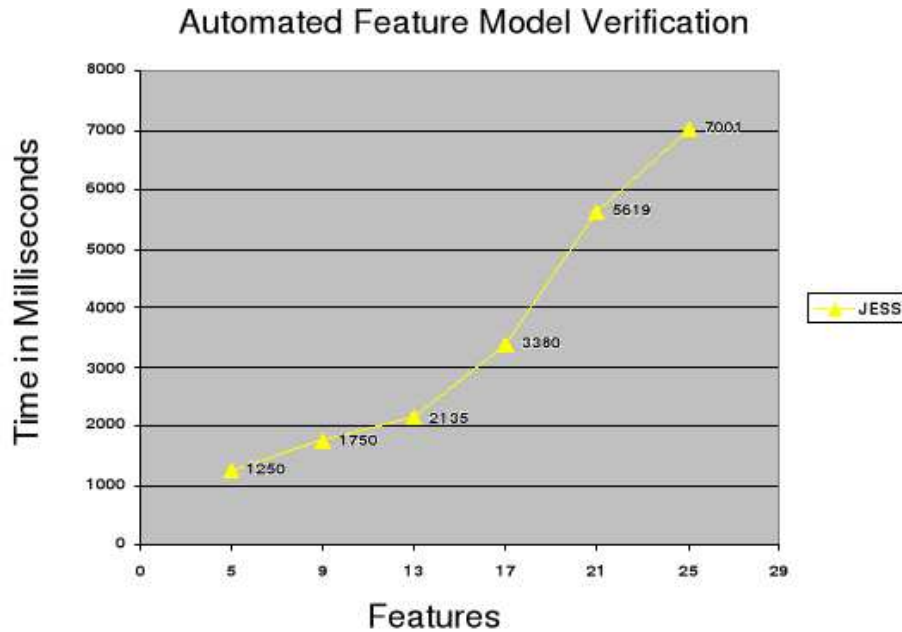


Figure 4.13: Performance Analysis

CHAPTER 5

EXPLOITING SEMANTICALLY ENRICHED FEATURE MODELS FOR SERVICE ONTOLOGY DEVELOPMENT

Both SPL and SOC paradigms promise high levels of software reuse in order to reduce time-to-market and development/maintenance costs of software intensive systems. In spite of sharing a common goal, they tackle with software reusability challenges from different perspectives by genuine practices and concepts. The diversity in independent approaches for making software reusable creates a wide range of opportunities for these paradigms to evolve in supporting more dynamic, systematic and adaptable reuse strategies through complementing and exploiting each others' existing methods and best practices.

Instead of developing each product from scratch, a planned and controlled reuse infrastructure which manages commonalities and differences among products is employed in product line engineering to develop resulting products in a more faster and cheaper way. In general, variability management enables systematic reuse of core assets including source code, requirements specifications, software architectures, design models, test cases and processes. Reusing core assets in a systematic way will lead large-productivity gains, short time-to-market, higher product quality, increased customer satisfaction, decreased development and maintenance costs.

Unlikely in SPL, current engineering approaches for developing services and service-oriented applications largely neglect managing commonalities and variabilities among service definitions in sufficient details. In order to foster productivity gains and to cope with maintainability problems and build future-proof service-oriented systems, SOC should employ necessary patterns, practices and concepts.

We believe that SPL’s domain analysis methods have great potentials in employing variability management during service design and development life-cycle. In this respect, we employ our semantically enriched feature models in order to achieve productivity gains, extensibility, maintainability and applicability goals in semantic modeling of web service families in a systematic fashion. This chapter presents our variability modeling approach for web service semantics along with a reference variability model and a case study given in subsequent sections.

5.1 A Variability Modeling Approach for Web Service Semantics

Systematic reuse within the web service development context encloses customization of a domain model in order to produce a family of services with related functionality. In the case of service semantics, this also includes production of service ontologies after refining the domain outcomes. A need for such reusable and specifiable web services seems obvious. For instance, a web service middleware for the tourism industry [42] may represent and implement a number of travel services where the exposed functionalities can be parameterized through different QoS requirements or mileage types.

Another good example for service family approach can be found in telecommunication service providing business. Based on the network bandwidth and other limitations, a low (or high) bandwidth voice codec can be preferred for the same Interactive-Voice-Response (IVR) service to transmit and receive voice data.

The notion of systematic reuse in developing families of web service are largely examined in Jiang et al’s work [41]. The work proposes a categorization of possible variation points among web services, and introduces a pattern-based approach for managing the variation points and specifying a web service framework. The variation points are defined on service endpoints, WSDL documents, and business logic.

Similarly, we analyze and manage the variability in web service definitions from three broad perspectives namely *Service Grounding*, *Service Profile* and *Service Model* which have been once introduced by the OWL-S service ontology. We adopt the Jiang et al’s notion of families of web services and bring the feature-oriented domain analysis (FODA) to it instead of the pattern-based variability management approach.

In basic terms a variability model employs variation points, different variation types and constraints among variations in order to represent variabilities defined within a specified con-

cept. Differences are placed in a concept through variation points which can be satisfied by variants. In FODA, the types of variations are generally classified to mandatory, alternative and optional features. Moreover, there can be additional constraints such as requires and excludes among features.

Semantic modeling of web service families and fostering mass development of service ontologies are based on identifying and managing the points of variability. In the following subsections we present three different categories of variation in terms of OWL-S and give example variability points for each of them through enabling ebBP specifications and the previously identified ones in [41].

Variation in Service Grounding

In general, Service Grounding describes how to access to the service through concrete specifications such as binding protocol, address, message formats etc. When we consider a bottom-up approach, service grounding can be generated from existing service interfaces, mostly from WSDL documents.

The provided binding mechanism can vary for the same service conceptualization based on the service invocation types. Main variability points of service grounding are identified in [68] as follows;

- *Binding Protocol*: A number of different application layer and transport layer protocols can be used for service interactions over a network such as *SOAP/HTTP*, *SOAP/HTTPS*, *SOAP/JMS* etc. These alternatives for the binding protocol selection constitute a main variability point for Service Grounding.
- *Binding Time*: Participating business service interfaces to be invoked during a business activity can be selected either in *design-time* or *run-time*. SOA supports dynamic selection of services to make a composite process model adaptable to changes in the execution environment.

Variation in Service Model

Service Model describes the semantics of how a service interacts with its clients, and the data and control flow of corresponding process specification. OWL-S process models; Simple, Atomic and Composite are subclasses of the Service Model. The ways a client may interact with a service through exchanging messages provides a basis for Service Model variability.

- *Message Exchange*: During the execution of a service process model, business documents can be exchanged through conforming two different patterns namely *synchronous* and *asynchronous*. The main difference between these two patterns is their effects on the initiator of the business transaction. In synchronous message exchange the requester party is blocked till it receives a response from the other party whereas in asynchronous pattern, the requester is not blocked.

Variation in Service Profile

Service Profile describes the what is done by service and presents necessary information such as service name, its text description and contact information. Service profiles are generally enabled in automated operations like dynamic service discovery. It can be considered as a yellow page entry of the service functionality. Information about inputs, outputs, preconditions and effects of the service are given the profile part. One important aspect of the service profile is its service parameter option which give the characteristic features of the service such as QoS and classification of service functionality in taxonomies provided by service registries. OWL-S's service profile can be directly mapped to UDDI registry data model [48].

In order to produce the appropriate exceptions, the ebBP specifications mandate a business service interface to conform to the following service parameters during the execution of the corresponding business activity. Indeed, each of these parameters creates a source of variability in service profiles;

- *AuthorizationRequired*: Exchanged business document must be signed by the sender and the receiving party must validate and approve the authorizer.
- *NonRepudiationRequired*: An exception should be raised whenever a business document has not properly delivered.
- *NonRepudiationOfReceiptRequired*: Both business partners agree to mutually verify receipt of a business document and that the receipt MUST be non-reputable. Non-repudiation of receipt provides the data for the following additional verifications; *Authenticate* and *Content Integrity*.

Reference Variability Model

We expose the previously identified variability points to our semantically enriched feature model as shown in Figure 5.1. During the service ontology generation, the feature model's

selected, or in other words customized, nodes are automatically transformed into accompanying OWL-S construct which is in this case the *serviceParameter* element of OWL-S Profile. OWL-S provides an unbounded list of service parameters that can contain any type of information. Thus, *serviceParameter* construct is very suitable for representing feature customizations. A *serviceParameter* consists of two attributes *serviceParameterName*, the name of the actual parameter, which could be just a literal, or perhaps the URI of the process parameter, and *sParameter* which points to the value of the parameter within some OWL ontology. A customized feature placed as a leaf node of the feature model can be mapped to a *serviceParameter* instance with the *Condition* type for the *sParameter*.

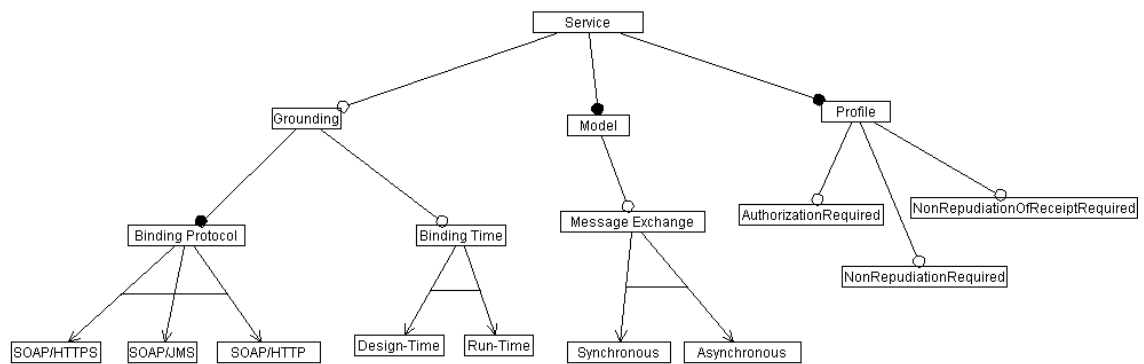


Figure 5.1: A reference variability model for semantic modeling of web service families

Moreover, new features can be included under the related variability category by the domain experts in order to extent the scope of the default web service variability model. After a new feature is added to the model, it can be transformed into the service ontology definition as it is explained for the default features. Variabilities captured within a feature model can be reflected to a business collaboration's OWL-S counterpart which is *Service*. The generic mapping table from Service Grounding, Model and Profile features to OWL-S Profile service parameter is given in Table 5.1.

Table 5.1: Features to OWL-S parameters

Feature	OWL-S Counterpart (serviceParameter)
Feature/@name	Profile/serviceParameter/serviceParameterName
	serviceParameter/sParameter/expr:Condition

5.2 An Example Walkthrough with the GENoDL

The first step in generating web service ontologies is to provide our reasoning tool with an ebBP instance as input. The tool then refines business process constructs such as *BusinessCollaboration* in order to compile them in possible service ontology representations. To start the transformation process, first a *BusinessCollaboration* is selected from the *Business Process Manager* section of the user interface. The tool then loads the default service feature model to the feature model editor. If the domain expert has any extension or further customization requests, she can modify the feature model by inserting/deleting or (de)selecting features through using the feature model editor bundled within our tool. Newly inserted features should be associated with their formal and machine readable definitions. After modifications, reasoner checks the resulting feature model's consistency and automatically corrects any invalid customization with domain expert's empowerment. Finally, the verified feature model customization along with the selected business collaboration specification is transformed into OWL-S notation according to the mappings given in Chapter 3.

Consider an example ebBP instance from the Dutch Criminal Justice project called ePV [25]. In this business process, the business collaborations taking part in the the process of demanding the surrender of a driver license for drunk driving among police, prosecution, court and RDW (the organization responsible for maintaining the national driver license database) are modeled. After loading the ebBP instance into the GENoDL, the tool refines *BusinessCollaboration* instances which are candidates for being realized in a web service protocol stack. From the list of ebBP elements, we select a *BusinessCollaboration* element called *BC-ID-DemandSurrenderOfDriverLicense*. A part of this business collaboration instance is represented in Figure 5.2. The *BC-ID-DemandSurrenderOfDriverLicense* is responsible for the control and data flow of the performed business activities for demanding surrender's driver license. After specifying which business collaboration is subject to be transformed into corresponding OWL-S representation, GENoDL loads the default service feature model to the feature model editor as depicted in Figure 5.3. By modifying the feature model, the user can customize and extend the feature model in order to meet the further needs that ebBP cannot fully consider such as low level service grounding issues which are specific to service realizations. Each different feature model customization results in different service ontology. Finally, GENoDL verifies the feature model and compiles it in OWL-S notation by clicking on the last toolbar icon of the feature model editor. An excerpt from the resulting service ontology for *BC-ID-DemandSurrenderOfDriverLicense* is given in Figure 5.4.

```

<!-- Demand surrender of driver license -->
<BusinessCollaboration name="Demand surrender of driver license" nameID="BC-ID-DemandSurrenderOfDriverLicense">
  <Role name="Police" nameID="IR-ID-Police"/>
  <Role name="Prosecution" nameID="IR-ID-Prosecution"/>
  <Role name="Court" nameID="IR-ID-Court"/>
  <Role name="RDV" nameID="IR-ID-RDV"/>
  <TimeToPerform/>

  <Start nameID="ID-IR-Start">
    <ToLink toBusinessStateRef="ID-IR-ReportDemandSurrenderOfDriverLicense"/>
  </Start>

  <BusinessTransactionActivity name="ReportDemandSurrenderOfDriverLicense" nameID="ID-IR-ReportDemandSurrenderOfDriverLic
    businessTransactionRef="ID-BT-ReportDemandSurrenderOfDriverLicense" hasLegalIntent="true">
    <TimeToPerform/>
    <Performs currentRoleRef="IR-ID-Police" performsRoleRef="ReportDemandSurrenderOfDriverLicenseInitiator"/>
    <Performs currentRoleRef="IR-ID-Prosecution" performsRoleRef="ReportDemandSurrenderOfDriverLicenseResponder"/>
  </BusinessTransactionActivity>

  <Transition>
    <FromLink fromBusinessStateRef="ID-IR-ReportDemandSurrenderOfDriverLicense"/>
    <ToLink toBusinessStateRef="ID-IR-OfficialReportDemandSurrenderOfDriverLicense"/>
  </Transition>

  <ComplexBusinessTransactionActivity nameID="ID-IR-OfficialReportDemandSurrenderOfDriverLicense"
    businessTransactionRef="ID-BT-OfficialReportDemandSurrenderOfDriverLicense" name="Official Report Demand surren
    hasLegalIntent="true">
    <TimeToPerform/>
    <Performs currentRoleRef="IR-ID-Police" performsRoleRef="OfficialReportDemandSurrenderOfDriverLicenseInitiator"/>
    <Performs currentRoleRef="IR-ID-Prosecution" performsRoleRef="OfficialReportDemandSurrenderOfDriverLicenseRespo

    <BusinessTransactionActivity nameID="ID-IR-ReportRevokeReturn" businessTransactionRef="ID-BT-ReportRevokeReturn
      name="ReportRevokeReturn" hasLegalIntent="true" >
      <TimeToPerform/>
      <Performs currentRoleRef="IR-ID-Prosecution" performsRoleRef="ReportRevokeReturnInitiator"/>
      <Performs currentRoleRef="IR-ID-RDV" performsRoleRef="ReportRevokeReturnResponder"/>
    </BusinessTransactionActivity>
    <StatusVisibility nameID="IR-ID-SV1" name="SV"/>
  </ComplexBusinessTransactionActivity>

  ...

```

Figure 5.2: ebBP representation for BC-ID-DemandSurrenderOfDriverLicense

It can be easily understood that a family of related service ontologies can be quickly generated by applying the proposed generative method. For example, by selecting different children of the *BindingProtocol* feature can result in various service conceptualizations that each of their implementations will be subject to be used in a distinct application scenario specifically.

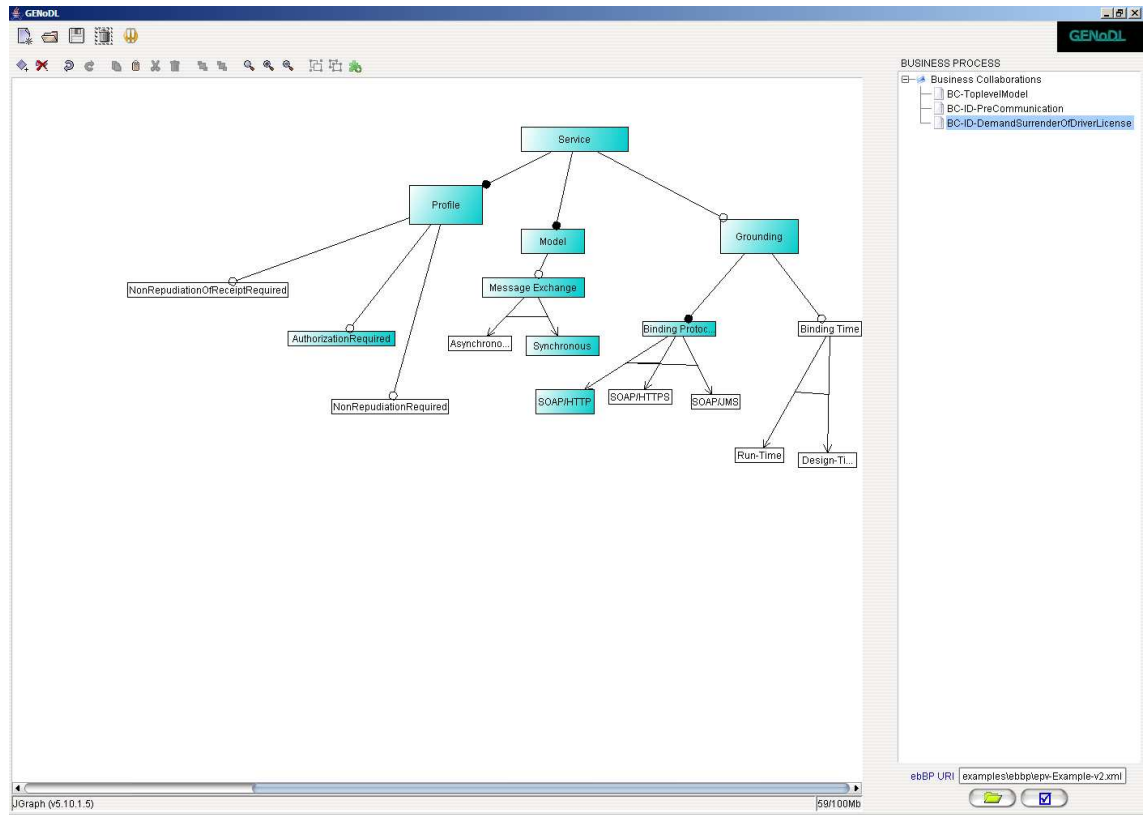


Figure 5.3: Service variability model is customized service feature model with the information extracted from BC-ID-DemandSurrenderOfDriverLicense.

```

<Service rdf:ID="Service_1">
  <describedBy rdf:resource="#CompositeProcess_3"/>
  <presents rdf:resource="#BC-ID-DemandSurrenderOfDriverLicense"/>
</Service>
<p1:Profile rdf:ID="BC-ID-DemandSurrenderOfDriverLicense">
  <p1:serviceName rdf:datatype="xsd:string">
    Demand surrender of driver license</p1:serviceName>
  <p1:serviceParameter rdf:resource="#ServiceParameter_27"/>
  <p1:serviceParameter rdf:resource="#ServiceParameter_2"/>
  <p1:serviceParameter rdf:resource="#ServiceParameter_4"/>
  <presentedBy rdf:resource="#Service_1"/>
</p1:Profile>
<p1:ServiceParameter rdf:ID="ServiceParameter_2">
  <p1:serviceParameterName rdf:datatype="xsd:string">Synchronous</p1:serviceParameter>
  <p1:sParameter rdf:resource="#Condition_3"/>
</p1:ServiceParameter>
<j.0:CompositeProcess rdf:ID="CompositeProcess_3">
  <j.0:hasParticipant rdf:resource="#IR-ID-RDW"/>
  <j.0:hasParticipant rdf:resource="#IR-ID-Court"/>
  <j.0:hasParticipant rdf:resource="#IR-ID-Police"/>
  <j.0:hasParticipant rdf:resource="#IR-ID-Prosecution"/>
  <j.0:composedOf rdf:resource="#ID-IR-Start"/>
  <describes rdf:resource="#Service_1"/>
</j.0:CompositeProcess>
<j.0:Sequence rdf:ID="ID-IR-Start">
  <j.0:components rdf:resource="#ControlConstructList_9"/>
</j.0:Sequence>
<j.0:AtomicProcess rdf:ID="ID-IR-ReportDemandSurrenderOfDriverLicense">
  <j.0:name rdf:datatype="xsd:string">
    ReportDemandSurrenderOfDriverLicense</j.0:name>
  <j.0:hasParticipant rdf:resource="#ReportDemandSurrenderOfDriverLicenseResponder"/>
  <j.0:hasParticipant rdf:resource="#ReportDemandSurrenderOfDriverLicenseInitiator"/>
  <j.0:hasClient rdf:resource="#IR-ID-Police"/>
  <j.0:performedBy rdf:resource="#IR-ID-Prosecution"/>
  <j.0:realizes rdf:resource="#ID-BT-ReportDemandSurrenderOfDriverLicense"/>
</j.0:AtomicProcess>
...

```

Figure 5.4: A part of the BC-ID-DemandSurrenderOfDriverLicense Service Ontology

CHAPTER 6

RELATED WORK

Feature-Oriented Domain Analysis utilizes propositional logic for defining feature models. On the other hand, various extensions to feature modeling can bring feature models closer to ontology formalism as in [19, 47]. Feature models can be represented in OWL DL ontology and then an OWL reasoning engine such as RACER [34] can be used to perform automated analysis over the feature model as described in [78]. Verification of feature models by using semantic web tools allows domain engineers to detect possible inconsistencies in feature configurations more efficiently than the traditional approaches. However, previous studies do not address the automated correction of inconsistent feature models. We enable this operation in our work as a contribution to the current state-of-art in automated feature model analysis.

Although it is shown that feature models can be conceptualized by formal methods, how to enable a feature model in the reusable asset development process is still remaining as an open issue. Because the level of abstraction at which a feature can denote entities or concepts will be ambiguous [18].

Mapping business process specification standards to accompanying OWL-S representations has been studied before in several studies. In two of them [33, 70], FPML and BPEL are transformed into OWL-S respectively. They provide a starting point for enriching the business process semantics in the form of OWL-S ontology for flexible integration and automation of workflows. We extend the current state in transformation approaches with employing variability management to be able to consider mass customization of a set of related ontologies.

An alternative method for automatic service ontology generation is given in [86]. The work exploits the UML class and state-chart diagrams for formally extracting the domain knowledge of atomic services and service compositions. UML is widely adopted in software

engineering as a standard for modeling, which most developers are familiar with. Once necessary information is extracted from UML diagrams, XSLT applications automatically transform the UML diagrams into OWL-S specification according to predefined rules. Even though the functional requirements especially the process side of the analysis model are automatically extracted and localized into OWL-S service process model, wider business and architectural issues such as non-functional requirements are not considered within the scope of this work.

There are a number of research studies [54, 63] for transforming business process specifications to more lower abstraction levels such as web service choreography or orchestration representations. However, these studies do not cover the variability issues and neglect the reusability opportunities that may arise after enabling service ontologies.

As it is understood from the previous research studies, the automated domain-specific knowledge analysis methods are still stands at an early stage. In this study, we separate the identification, specification and realization concerns of a service-oriented system by means of domain engineering outcomes such as business processes for identification, service ontologies for specification and service implementations for realization. This separation of concerns, which well fits in Service-Oriented Modeling and Architecture (SOMA) [5], allows us to automate the mapping from identification step to specification step through utilizing semantic web technologies and also it provides means for capturing commonalities and variabilities in service models by exploiting feature models.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Conventional development methodologies such as object-oriented development and component-based development do not fully address key engineering challenges of today's service-oriented systems such as setting the proper level of service interface granularity and facilitating mass deployment of services. In fact, service-oriented design and development requires an interdisciplinary approach fusing concepts of business process management with traditional software development methods [58].

In this respect, providing better means for the business process management (BPM) and service-oriented computing (SOC) convergence is one of the major concerns that get a lot of attention in contemporary software engineering research. Despite their differences, when combined together, they become synergetic through addressing the challenges of rapidly changing business environments, reducing cost and increase efficiency in implementing innovations. SOC provides the necessary agility and loose coupling to IT infrastructures and BPM provides its business case where business processes, which are considered as reusable elements and are independent from implementation technology, are viewed as federation of services connected via standard protocols in a service-oriented architecture. While BPM-SOC convergence looks promising, how to map the process models to service definitions is still an open issue.

7.1 Conducted Work

In order to address this challenge, we exploit software product line (SPL) approach where domain engineering outcomes are transformed into particular products in application engineering. In our case, domain engineering outcomes consist of business process models and feature models whereas products in application engineering represent the resulting service

ontologies. As the result of the introduced product line process, a set of web service ontologies based on business process models and feature model customizations are produced. Promises of our SPL dialect of BPM-SOC convergence problem is given as follows;

- Knowledge-based elicitation and refinement of domain engineering outcomes as service ontologies allows designing and deploying services better aligned with business goals, stakeholder’s concerns and end-user’s viewpoints.
- Changes in the business parameters can be more rapidly reflected to the service realizations in the supervision of SPL
- Service-oriented systems can be more rapidly deployed by enabling *build-by-integration* paradigm which encourages matching product capabilities to existing services by utilizing semantic web technologies.
- A set of related web services can be more rapidly developed by managing commonalities and variabilities among them.

Within the scope of our product line, a transformation method for automatic generation of service ontologies is presented in order to formalize and ease the mapping among the concepts derived by domain and application engineering. Business process models are consumed as the meta knowledge and the service ontologies are populated using transformation method over this knowledge base.

7.2 Concluding Remarks

The mechanisms that were developed, were experimented through an example. This example included the mapping from an existing business process model, that represents the domain-level knowledge. The outcome has been promising; in general it has been observed that automated generation of web service ontologies is possible and usable. However, our observation also states that very high-abstraction level elements of the domain is not easy to map directly.

7.3 Future Work

Our novel generative method for service ontology creation has not reached its maturity yet. Although the algorithm for mapping business transaction definitions to OWL-S service process models has been already implemented, it should be revised and extended in favor of

generating more coarse-grained service ontologies including more detailed process models. Moreover, the generative method will be extended with the mapping rules of other ebBP concepts that have not been involved in the current implementation such as guards, exceptions and signals.

Our area of interest for this work is not limited with one semantic model of a single web service indeed, we focus on a family of services and their ontological representations. Therefore, we provide a variability management method for service ontologies based on feature modeling. Commonalities and variabilities among semantic models are captured within a reference variability model in order to foster mass development of ontologies by enabling reusable assets. The axioms are evaluated over the feature model and combined with the business process model inference data to enable easy and rapid development of a family of formal service models. Current version of the feature model editor and feature model ontology does not support cardinality-based relations among features as proposed in [18]. This will be implemented as a possible improvement. Furthermore, an evaluation of memory allocated for varying input sizes will be performed in order to assess the feasibility of the feature model verification operation from a different perspective.

Although our product line approach for service ontology generation is based on two well known standards of their domains (ebBP and OWL-S), this dependence makes the approach tightly coupled with these schema definitions. A more generic approach which is independent from implementation technologies is considered as a future work. Finally, evaluation of the quality and domain coverage of the resulting service ontologies should be explicitly justified through anticipated metrics and methods, so that domain experts and developers can assess them easily.

REFERENCES

- [1] A. Dogac A, Y. Kabak, O. Gulderen, T. Namli, A. Okcan, O. Kilic, Y. Gurcan, U. Orhan, and G. Laleci. ebBP Profile for Integrating Healthcare Enterprise (IHE). *Submitted to OASIS ebXML Business Process Technical Committee*, 2006.
- [2] N. I. Altintas. *Feature Based Software Asset Modeling with Domain Specific Kits*. PhD thesis, Middle East Technical University, August 2007.
- [3] N. I. Altintas, S. Cetin, and A. H. Dogru. Industrializing Software Development: The Factory Automation Way. *Lecture Notes in Computer Science*, 4473:54–68, 2007.
- [4] An Example Application Engineering Process. http://www.sei.cmu.edu/domain-engineering/appl_eng_example.html, last visited on October 2008.
- [5] A. Arsanjani. Service-oriented modeling and architecture. Technical report, IBM developerWorks, 2004.
- [6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [7] David Benavides, Antonio Ruiz Cortés, Pablo Trinidad, and Sergio Segura. A survey on the automated analyses of feature models. In *JISBD*, pages 367–376, 2006.
- [8] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284:34–43, 2001.
- [9] Joseph Bih. Service oriented architecture: A new paradigm to implement dynamic e-business solutions. *ACM Ubiquity*, 7(30), 2006.
- [10] Web Services Business Process Execution Language Version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpelv2.0OS.html>, last visited on October 2008.

- [11] Business Process Modeling Notation (BPMN), Object Management Group/Business Process Management Initiative. <http://www.omg.org/docs/dtc/060201.pdf>, last visited on October 2008.
- [12] Semih Cetin, N. Ilker Altintas, and Cevat Sener. An architectural modeling approach with symmetric alignment of multiple concern spaces. *International Conference on Software Engineering Advances*, 0:48, 2006.
- [13] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison Wesley Professional, August 2001.
- [14] COM: Component Object Model Technologies, Microsoft/COM Initiative. <http://www.microsoft.com/com/default.mspx>, last visited on October 2008.
- [15] Common Object Request Broker Architecture (CORBA), Object Management Group/CORBA Initiative. <http://www.omg.org/corba/>, last visited on October 2008.
- [16] K. Czarnecki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. PhD thesis, Technical University of Ilmenau, October 1998.
- [17] K. Czarnecki. Overview of Generative Software Development. *Lecture Notes in Computer Science*, 3566:326–341, 2005.
- [18] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. In *Software Process Improvement and Practice*, 2005.
- [19] Krzysztof Czarnecki, Chang Hwan Peter Kim, and Karl Trygve Kalleberg. Feature models are views on ontologies. In *SPLC '06: Proceedings of the 10th International on Software Product Line Conference*, pages 41–51, Washington, DC, USA, 2006. IEEE Computer Society.
- [20] DAML+OIL Web Ontology Language. <http://www.w3.org/Submission/2001/12/>, last visited on October 2008.
- [21] Dublin Core Metadata Initiative. <http://www.dublincore.org/>, last visited on October 2008.
- [22] ebXML Business Process Specification Schema. <http://docs.oasis-open.org/ebxml-bp-2.0.4/OS/>, last visited on October 2008.

- [23] ebBP Teleconference 8 August 2006. <http://www.oasisopen.org/committees/download.php/19736/ebxmlbp-v2.0.3Minutes-update.txt>, last visited on October 2008.
- [24] Marco Eichelberg, Thomas Aden, Jörg Riesmeier, Asuman Dogac, and Gokce B. Laleci. A survey and analysis of electronic healthcare record standards. *ACM Comput. Surv.*, 37(4):277–315, 2005.
- [25] ePV Netherlands Criminal Justic ebBP Example. http://www.oasis-open.org/committees/document.php?document_id=16436wg_abbrev=ebxml-bp, last visited on October 2008.
- [26] K. Gottschalk et al. Web services architecture overview. Technical report, <http://www.ibm.com/developerworks/webservices/library/wovr/>, September 2000.
- [27] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
- [28] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.
- [29] J. Greenfield, K. Short, S. Cook, and S. Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, August 2004.
- [30] M. L. Griss and K. Wentzel. Hybrid domain specific kits for a flexible software factory. In *Proceedings of the Ann. ACM Symp. Applied Computing*, pages 47–52, 1994.
- [31] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisitons*, 5:199–220, 1993.
- [32] T. R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human Computer Studies*, 43:907–928, 1995.
- [33] Li Guo, Yun-Heh Chen-Burger, and Dave Roberston. Mapping a business process model to a semantic web service model. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 746, Washington, DC, USA, 2004. IEEE Computer Society.
- [34] Volker Haarslev and Ralf Muller. Racer: An owl reasoning agent for the semantic web. In *In Proc. of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with 2003 IEEE/WIC International Conference on Web Intelligence*, pages 91–95. Society Press, 2003.

- [35] J. Hanson. Coarse-grained interfaces enable service composition in soa. In *JavaOne*, August 2003.
- [36] Hao He. What is service-oriented architecture. Technical report, <http://www.xml.com/lpt/a/1292>, September 2003.
- [37] Michi Henning. The rise and fall of corba. *ACM Queue*, 4(5), 2006.
- [38] Hypertext Transfer Protocol 1.1. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>, last visited on October 2008.
- [39] JESS: the Rule Engine for the Java Platform. <http://www.jessrules.com/>, last visited on October 2008.
- [40] Java Graph Visualization and Layout. <http://www.jgraph.com/>, last visited on October 2008.
- [41] Juanjuan Jiang, Anna Ruokonen, and Tarja Systa. Pattern-based variability management in web service development. In *ECOWS '05: Proceedings of the Third European Conference on Web Services*, page 83, Washington, DC, USA, 2005. IEEE Computer Society.
- [42] Y. Kabak, M. Olduz, G. B. Laleci, T. Namli T, V. Bicer, N. Radic, and A. Dogac. A semantic web service based middleware for the tourism industry. In *Book Chapter, to appear*.
- [43] Faouzi Kamoun. A roadmap towards the convergence of business process management and service oriented architecture. *Ubiquity*, 8(14):1–1, 2007.
- [44] 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.
- [45] M. Kuan. Using swrl and owl dl to develop an inference system for course scheduling. Master’s thesis, Chung Yuan Christian University, Taiwan, 2004.
- [46] M. LeClerc. Service layer transition towards a service oriented architecture (soa) - expanding the value of operators’ telecom assets. In *EMEA Conference*, 2005.
- [47] Soon-Bok Lee, Jin-Woo Kim, Chee-Yang Song, and Doo-Kwon Baik. An approach to analyzing commonality and variability of features using ontology in a software product

- line engineering. In *SERA '07: Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, pages 727–734, Washington, DC, USA, 2007. IEEE Computer Society.
- [48] Jim Luo, Bruce Montrose, Anya Kim, Amitabh Khashnobish, and Myong Kang. Adding owl-s support to the existing uddi infrastructure. pages 153–162, 2006.
 - [49] Dale Molberg and Monica J. Martin. The ebbp (ebxml business process specification schema. Technical report, OASIS, April 2006.
 - [50] Organization for the Advancement of Structured Information Standards (OASIS). <http://www.oasisopen.org/home/index.php>, last visited on October 2008.
 - [51] Martin O’Connor, Holger Knublauch, Samson Tu, Benjamin Grosf, Mike Dean, William Grosso, and Mark Musen. Combining swrl rules and owl ontologies with protégé owl plugin, jess, and racer. *7th International Protégé Conference*, 2004.
 - [52] Martin O’Connor, Holger Knublauch, Samson Tu, Benjamin Grosf, Mike Dean, William Grosso, and Mark Musen. Supporting rule system interoperability on the semantic web with swrl. *Fourth International Semantic Web Conference (ISWC-2005)*, 2005.
 - [53] Oracle. Oracle it modernization series: The types of modernization. Technical report, 2006.
 - [54] Chun Ouyang, M. Dumas, Ter, and W. M. P. van der Aalst. From bpmn process models to bpel web services. pages 285–292, 2006.
 - [55] OWL Web Ontology Language Overview. <http://www.w3.org/TR/owlfeatures/>, last visited on October 2008.
 - [56] Semantic Markup for Web Services . <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>, last visited on October 2008.
 - [57] M. Paolucci, W. Goix, A. Andreetto, M. Luther, and M. Wagner. Representing Services for Mobile Computing using OWL and OWL-S. *WWW Service Composition with Semantic Web Services (WSComp)*, 2005.
 - [58] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, , and B. J. Kramer. Service-oriented computing research roadmap. Technical report, March 2006.

- [59] Artem Papkov. Develop a migration strategy from a legacy enterprise it infrastructure to an soa-based enterprise architecture. Technical report, <http://www.ibm.com/developerworks/webservices/library/ws-migrate2soa/>, April 2005.
- [60] Protégé Ontology Editor and Knowledge Acquisition System. <http://protege.stanford.edu/>, last visited on October 2008.
- [61] Resource Description Framework (RDF) Model and Syntax Specification. <http://www.w3.org/TR/1999/RECrdfsyntax19990222/>, last visited on October 2008.
- [62] RDF Vocabulary Description Language 1.0: RDF Schema. <http://www.w3.org/TR/rdfschema/>, last visited on October 2008.
- [63] Jan Recker and Jan Mendling. On the translation between bpmn and bpel: Conceptual mismatch between process modeling languages.
- [64] RIDE Deliverable 5.3.1 - Contribution to Standards: ebBP Editor v1.0.4 User Manual. <http://www.srdc.metu.edu.tr/webpage/publications/2007/ebBPEditor-UserManualv1.0.4.pdf>, last visited on October 2008.
- [65] RIDE Deliverable 4.4.5 - Integrating the Legacy eHealth Applications of the Member States into the RIDE Technical Framework. http://www.srdc.metu.edu.tr/webpage/projects/ride/deliverables%20-%20RIDE_D4.4.5_v1.1.doc, last visited on October 2008.
- [66] Rick Robinson. Understand enterprise service bus scenarios and solutions in service-oriented architecture. Technical report, <https://www.ibm.com/developerworks/library/ws-esbscen/>, June 2004.
- [67] Marta Sabou, Chris Wroe, Carole Goble, and Gilad Mishne. Learning domain ontologies for web service descriptions: an experiment in bioinformatics. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 190–198, New York, NY, USA, 2005. ACM.
- [68] S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad. A taxonomy of variability in web service flows. In *Service Oriented Architectures and Product Lines (SOAPL - 07)*, Kyoto, Japan, September 2007.
- [69] SEI. Smart: The service-oriented migration and reuse technique. Technical Report CMU/SEI-2005-TN-029, 2005.

- [70] Jun Shen, Yun Yang, Chengang Wan, and Chuan Zhu. From bpel4ws to owl-s: Integrating e-business process descriptions. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 181–190, Washington, DC, USA, 2005. IEEE Computer Society.
- [71] Simple Object Access Protocol (SOAP). <http://www.w3.org/TR/soap/>, last visited on October 2008.
- [72] Semantic Web Rule Language. <http://www.w3.org/Submission/SWRL/>, last visited on October 2008.
- [73] UDDI Version 3.0.2. http://uddi.org/pubs/uddi_v3.htm, last visited on October 2008.
- [74] Emanuele Della Valle, Dario Cerizza, Veli Bicer, Yildirak Kabak, Gokce Banu Laleci, and Holger Lausen. The need for semantic web service in the ehealth. *W3C Workshop on Frameworks for Semantics in Web Services*, 2005.
- [75] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 1 edition, July 2007.
- [76] The World Wide Web Consortium (W3C). <http://www.w3.org/>, last visited on October 2008.
- [77] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical report, Sun Microsystems Laboratories, Mountain View, CA, USA, 1994.
- [78] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. A semantic web approach to feature modeling and verification. In *In Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, 2005.
- [79] Web Services Architecture. <http://www.w3.org/TR/wsarch/>, last visited on October 2008.
- [80] David M. Weiss and Chi Tau Robert Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley Professional, August 1999.
- [81] Web Services Architecture. www.w3.org/TR/wsarch/, last visited on October 2008.
- [82] Web Services Choreography Description Language Version 1.0. <http://www.w3.org/TR/2004/WDwscdl10-20041217/>, last visited on October 2008.

- [83] Web service description language (WSDL). <http://www.w3.org/TR/wsdl/>, last visited on October 2008.
- [84] Web Service Modeling Ontology (WSMO). <http://www.w3.org/Submission/WSMO/>, last visited on October 2008.
- [85] The Extensible Markup Language (XML). <http://www.w3.org/XML/>, last visited on October 2008.
- [86] Jin Yang and In Chung. Automatic generation of service ontology from uml diagrams for semantic web services. pages 523–529. 2006.

APPENDIX A

FEATURE MODEL ONTOLOGY

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:temporal="http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl#"
  xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
  xmlns:query="http://swrl.stanford.edu/ontologies/built-ins/3.3/query.owl#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlx="http://swrl.stanford.edu/ontologies/built-ins/3.3/swrlx.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
  xmlns="http://www.owl-ontologies.com/Ontology1209057357.owl#"
  xmlns:swrlm="http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlm.owl#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:abox="http://swrl.stanford.edu/ontologies/built-ins/3.3/abox.owl#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
  xmlns:tbox="http://swrl.stanford.edu/ontologies/built-ins/3.3/tbox.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1209057357.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.3/abox.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.3/swrlx.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.4/swrlm.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/3.3/swrla.owl"/>
    <owl:imports rdf:resource="http://www.w3.org/2003/11/swrlb"/>
    <owl:imports rdf:resource="http://www.w3.org/2003/11/swrl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.3/tbox.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/built-ins/3.3/query.owl"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Feature"/>
  <owl:Class rdf:ID="Alternative_Feature">
```

```

<rdfs:subClassOf rdf:resource="#Feature"/>
<owl:disjointWith>
  <owl:Class rdf:ID="Mandatory_Feature"/>
</owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Mandatory_Feature">
  <owl:disjointWith rdf:resource="#Alternative_Feature"/>
  <rdfs:subClassOf rdf:resource="#Feature"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="requires">
  <rdfs:domain rdf:resource="#Feature"/>
  <rdfs:range rdf:resource="#Feature"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="excludes">
  <rdfs:domain rdf:resource="#Feature"/>
  <rdfs:range rdf:resource="#Feature"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasParentFeature">
  <rdfs:domain rdf:resource="#Feature"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasChildFeature"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#Feature"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="alternativeOf">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Alternative_Feature"/>
  <rdfs:range rdf:resource="#Alternative_Feature"/>
  <owl:inverseOf rdf:resource="#alternativeOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasChildFeature">
  <rdfs:domain rdf:resource="#Feature"/>
  <owl:inverseOf rdf:resource="#hasParentFeature"/>
  <rdfs:range rdf:resource="#Feature"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.w3.org/2003/11/swrl#argument2"/>
<owl:DatatypeProperty rdf:ID="isSelected">
  <rdfs:domain rdf:resource="#Feature"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Feature"/>

```

```

</owl:DatatypeProperty>
<swrl:Imp rdf:ID="Requires">
  <swrl:body>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:IndividualPropertyAtom>
              <swrl:argument2>
                <swrl:Variable rdf:ID="y"/>
              </swrl:argument2>
              <swrl:argument1>
                <swrl:Variable rdf:ID="x"/>
              </swrl:argument1>
              <swrl:propertyPredicate rdf:resource="#requires"/>
            </swrl:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                  <swrl:argument1 rdf:resource="#x"/>
                  <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                    >true</swrl:argument2>
                  <swrl:propertyPredicate rdf:resource="#isSelected"/>
                </swrl:DatavaluedPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            </swrl:AtomList>
          </rdf:rest>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </swrl:body>
  <swrl:head>
    <swrl:AtomList>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:argument1 rdf:resource="#y"/>
          <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >true</swrl:argument2>
          <swrl:propertyPredicate rdf:resource="#isSelected"/>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
</swrl:Imp>

```



```

        </swrl:DatavaluedPropertyAtom>
    </rdf:first>
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
</swrl:AtomList>
</swrl:head>
</swrl:Imp>
<swrl:Imp rdf:ID="General">
    <swrl:head>
        <swrl:AtomList>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                    <swrl:propertyPredicate rdf:resource="#isSelected"/>
                    <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                    >false</swrl:argument2>
                    <swrl:argument1 rdf:resource="#y"/>
                </swrl:DatavaluedPropertyAtom>
            </rdf:first>
        </swrl:AtomList>
    </swrl:head>
    <swrl:body>
        <swrl:AtomList>
            <rdf:first>
                <swrl:ClassAtom>
                    <swrl:classPredicate rdf:resource="#Feature"/>
                    <swrl:argument1 rdf:resource="#x"/>
                </swrl:ClassAtom>
            </rdf:first>
            <rdf:rest>
                <swrl:AtomList>
                    <rdf:first>
                        <swrl:DatavaluedPropertyAtom>
                            <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                            >false</swrl:argument2>
                            <swrl:propertyPredicate rdf:resource="#isSelected"/>
                            <swrl:argument1 rdf:resource="#x"/>
                        </swrl:DatavaluedPropertyAtom>
                    </rdf:first>
                    <rdf:rest>
                        <swrl:AtomList>
                            <rdf:first>
                                <swrl:IndividualPropertyAtom>
                                    <swrl:argument2 rdf:resource="#y"/>
                                    <swrl:argument1 rdf:resource="#x"/>
                                    <swrl:propertyPredicate rdf:resource="#hasChildFeature"/>
                                </swrl:IndividualPropertyAtom>
                            </rdf:first>
                            <rdf:rest>

```

```

        <swrl:AtomList>
          <rdf:first>
            <swrl:DatavaluedPropertyAtom>
              <swrl:propertyPredicate rdf:resource="#isSelected"/>
              <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >true</swrl:argument2>
              <swrl:argument1 rdf:resource="#y"/>
            </swrl:DatavaluedPropertyAtom>
          </rdf:first>
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </rdf:rest>
</swrl:AtomList>
</rdf:rest>
</swrl:body>
</swrl:Imp>
<swrl:Imp rdf:ID="Mandatory">
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:argument1 rdf:resource="#x"/>
          <swrl:propertyPredicate rdf:resource="#isSelected"/>
          <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >true</swrl:argument2>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <swrl:ClassAtom>
          <swrl:classPredicate rdf:resource="#Mandatory_Feature"/>
          <swrl:argument1 rdf:resource="#x"/>
        </swrl:ClassAtom>
      </rdf:first>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                  <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"

```

```

        >true</swrl:argument2>
        <swrl:propertyPredicate rdf:resource="#isSelected"/>
        <swrl:argument1 rdf:resource="#y"/>
    </swrl:DatavaluedPropertyAtom>
</rdf:first>
<rdf:rest>
    <swrl:AtomList>
        <rdf:first>
            <swrl:DatavaluedPropertyAtom>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</swrl:argument2>
                <swrl:propertyPredicate rdf:resource="#isSelected"/>
            </swrl:DatavaluedPropertyAtom>
        </rdf:first>
        <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    </swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
</rdf:rest>
<rdf:first>
    <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="#hasParentFeature"/>
        <swrl:argument2 rdf:resource="#y"/>
        <swrl:argument1 rdf:resource="#x"/>
    </swrl:IndividualPropertyAtom>
</rdf:first>
</swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
</swrl:body>
</swrl:Imp>
<swrl:Imp rdf:ID="Alternative_Parent">
    <swrl:head>
        <swrl:AtomList>
            <rdf:first>
                <swrl:IndividualPropertyAtom>
                    <swrl:argument1>
                        <swrl:Variable rdf:ID="z"/>
                    </swrl:argument1>
                    <swrl:argument2 rdf:resource="#y"/>
                    <swrl:propertyPredicate rdf:resource="#hasParentFeature"/>
                </swrl:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </swrl:AtomList>
    </swrl:head>
    <swrl:body>

```

```

<swrl:AtomList>
  <rdf:rest>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:rest>
                <swrl:AtomList>
                  <rdf:first>
                    <swrl:DifferentIndividualsAtom>
                      <swrl:argument2 rdf:resource="#y"/>
                      <swrl:argument1>
                        <swrl:Variable rdf:ID="w"/>
                      </swrl:argument1>
                    </swrl:DifferentIndividualsAtom>
                  </rdf:first>
                  <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                </swrl:AtomList>
              </rdf:rest>
            <rdf:first>
              <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate rdf:resource="#hasParentFeature"/>
                <swrl:argument1 rdf:resource="#z"/>
                <swrl:argument2 rdf:resource="#w"/>
              </swrl:IndividualPropertyAtom>
            </rdf:first>
          </swrl:AtomList>
        </rdf:rest>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:argument2 rdf:resource="#z"/>
          <swrl:propertyPredicate rdf:resource="#alternativeOf"/>
          <swrl:argument1 rdf:resource="#x"/>
        </swrl:IndividualPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </rdf:rest>
</rdf:first>
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate rdf:resource="#hasParentFeature"/>
  <swrl:argument2 rdf:resource="#y"/>
  <swrl:argument1 rdf:resource="#x"/>
</swrl:IndividualPropertyAtom>
</rdf:first>
</swrl:AtomList>
</rdf:rest>
<rdf:first>

```

```

        <swrl:ClassAtom>
            <swrl:classPredicate rdf:resource="#Alternative_Feature"/>
            <swrl:argument1 rdf:resource="#x"/>
        </swrl:ClassAtom>
    </rdf:first>
</swrl:AtomList>
</swrl:body>
</swrl:Imp>
<swrl:AtomList>
    <rdf:first>
        <swrl:ClassAtom>
            <swrl:classPredicate rdf:resource="#Alternative_Feature"/>
            <swrl:argument1 rdf:resource="#x"/>
        </swrl:ClassAtom>
    </rdf:first>
    <rdf:rest>
        <swrl:AtomList>
            <rdf:first>
                <swrl:IndividualPropertyAtom>
                    <swrl:argument2 rdf:resource="#y"/>
                    <swrl:argument1 rdf:resource="#x"/>
                    <swrl:propertyPredicate rdf:resource="#hasParentFeature"/>
                </swrl:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
                <swrl:AtomList>
                    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                <rdf:first>
                    <swrl:IndividualPropertyAtom>
                        <swrl:argument1 rdf:resource="#x"/>
                        <swrl:argument2 rdf:resource="#z"/>
                        <swrl:propertyPredicate rdf:resource="#alternative0f"/>
                    </swrl:IndividualPropertyAtom>
                </rdf:first>
            </swrl:AtomList>
        </rdf:rest>
    </swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
<swrl:Imp rdf:ID="Alternative">
    <swrl:head>
        <swrl:AtomList>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                    <swrl:propertyPredicate rdf:resource="#isSelected"/>
                    <swrl:argument1 rdf:resource="#y"/>
                    <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"

```

```

        >false</swrl:argument2>
    </swrl:DatavaluedPropertyAtom>
</rdf:first>
</swrl:AtomList>
</swrl:head>
<swrl:body>
    <swrl:AtomList>
        <rdf:rest>
            <swrl:AtomList>
                <rdf:first>
                    <swrl:DatavaluedPropertyAtom>
                        <swrl:propertyPredicate rdf:resource="#isSelected"/>
                        <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                        >true</swrl:argument2>
                        <swrl:argument1 rdf:resource="#x"/>
                    </swrl:DatavaluedPropertyAtom>
                </rdf:first>
                <rdf:rest>
                    <swrl:AtomList>
                        <rdf:rest>
                            <swrl:AtomList>
                                <rdf:first>
                                    <swrl:DatavaluedPropertyAtom>
                                        <swrl:argument1 rdf:resource="#y"/>
                                        <swrl:propertyPredicate rdf:resource="#isSelected"/>
                                        <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                                        >true</swrl:argument2>
                                    </swrl:DatavaluedPropertyAtom>
                                </rdf:first>
                                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                            </swrl:AtomList>
                        </rdf:rest>
                    </swrl:AtomList>
                </rdf:rest>
            </swrl:AtomList>
            <swrl:IndividualPropertyAtom>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:resource="#y"/>
                <swrl:propertyPredicate rdf:resource="#alternativeOf"/>
            </swrl:IndividualPropertyAtom>
        </rdf:first>
    </swrl:AtomList>
</rdf:rest>
<rdf:first>
    <swrl:ClassAtom>
        <swrl:classPredicate rdf:resource="#Alternative_Feature"/>
        <swrl:argument1 rdf:resource="#x"/>
    </swrl:ClassAtom>

```

```

        </rdf:first>
    </swrl:AtomList>
</swrl:body>
</swrl:Imp>
<swrl:Imp rdf:ID="Excludes">
    <swrl:head>
        <swrl:AtomList>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                    <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                    >false</swrl:argument2>
                    <swrl:argument1 rdf:resource="#y"/>
                    <swrl:propertyPredicate rdf:resource="#isSelected"/>
                </swrl:DatavaluedPropertyAtom>
            </rdf:first>
        </swrl:AtomList>
    </swrl:head>
    <swrl:body>
        <swrl:AtomList>
            <rdf:rest>
                <swrl:AtomList>
                    <rdf:first>
                        <swrl:IndividualPropertyAtom>
                            <swrl:argument2 rdf:resource="#y"/>
                            <swrl:propertyPredicate rdf:resource="#excludes"/>
                            <swrl:argument1 rdf:resource="#x"/>
                        </swrl:IndividualPropertyAtom>
                    </rdf:first>
                    <rdf:rest>
                        <swrl:AtomList>
                            <rdf:first>
                                <swrl:DatavaluedPropertyAtom>
                                    <swrl:propertyPredicate rdf:resource="#isSelected"/>
                                    <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                                    >true</swrl:argument2>
                                    <swrl:argument1 rdf:resource="#x"/>
                                </swrl:DatavaluedPropertyAtom>
                            </rdf:first>
                            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                        </swrl:AtomList>
                    </rdf:rest>
                </swrl:AtomList>
            </rdf:rest>
            <rdf:first>
                <swrl:ClassAtom>
                    <swrl:classPredicate rdf:resource="#Feature"/>
                    <swrl:argument1 rdf:resource="#x"/>
                </swrl:ClassAtom>
            </rdf:first>
        </swrl:AtomList>
    </swrl:body>
</swrl:Imp>

```

```
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:body>
</swrl:Imp>
</rdf:RDF>
```

```
<!-- Created with Protege (with OWL Plugin 3.4, Build 506) http://protege.stanford.edu -->
```


APPENDIX B

SCHEMA DEFINITIONS OF THE CORE EBBP COMPONENTS

Details of the document envelope structure is placed in Figure B.1.

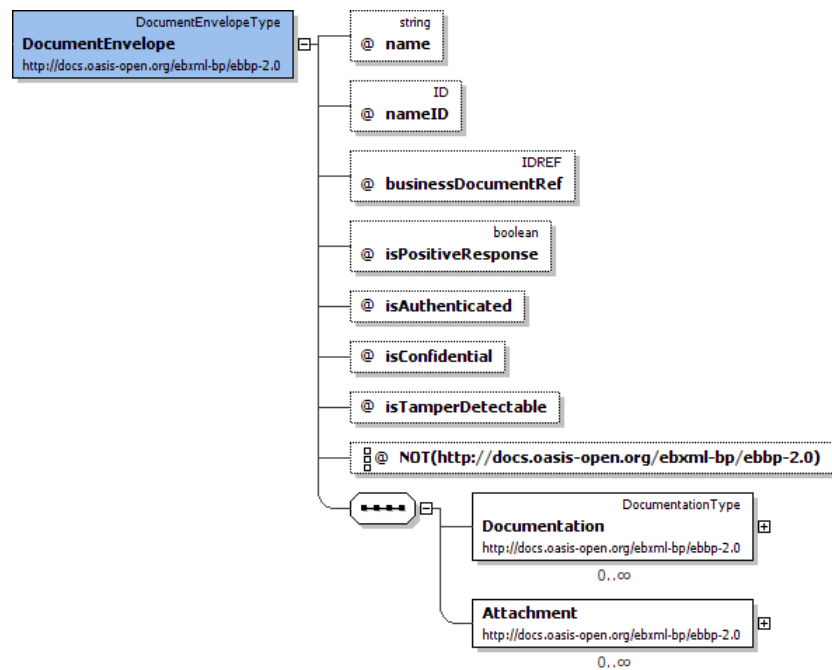


Figure B.1: Details of the Document Envelope structure

Business documents can be referenced through the dedicated ebBP structure given in Figure B.2.

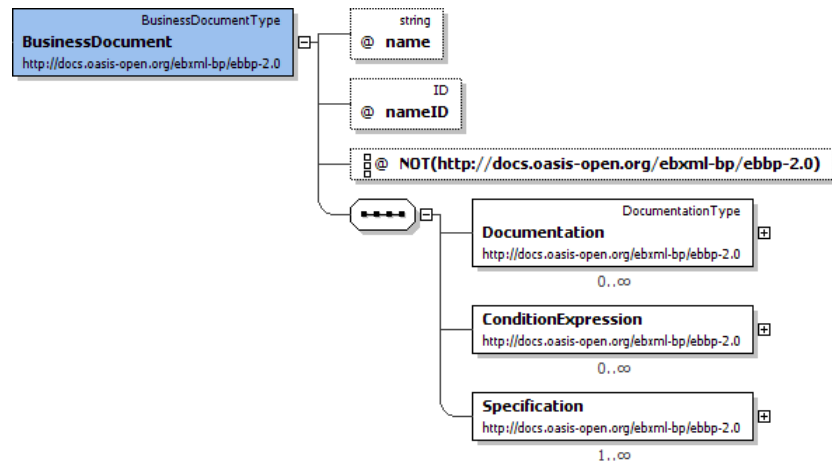


Figure B.2: ebBP definition for logical Business Documents

As it is visible in Figure B.3, a business transaction consists of a Requesting Business Activity, a Responding Business Activity, one or two business document flow between them and several optional business signals.

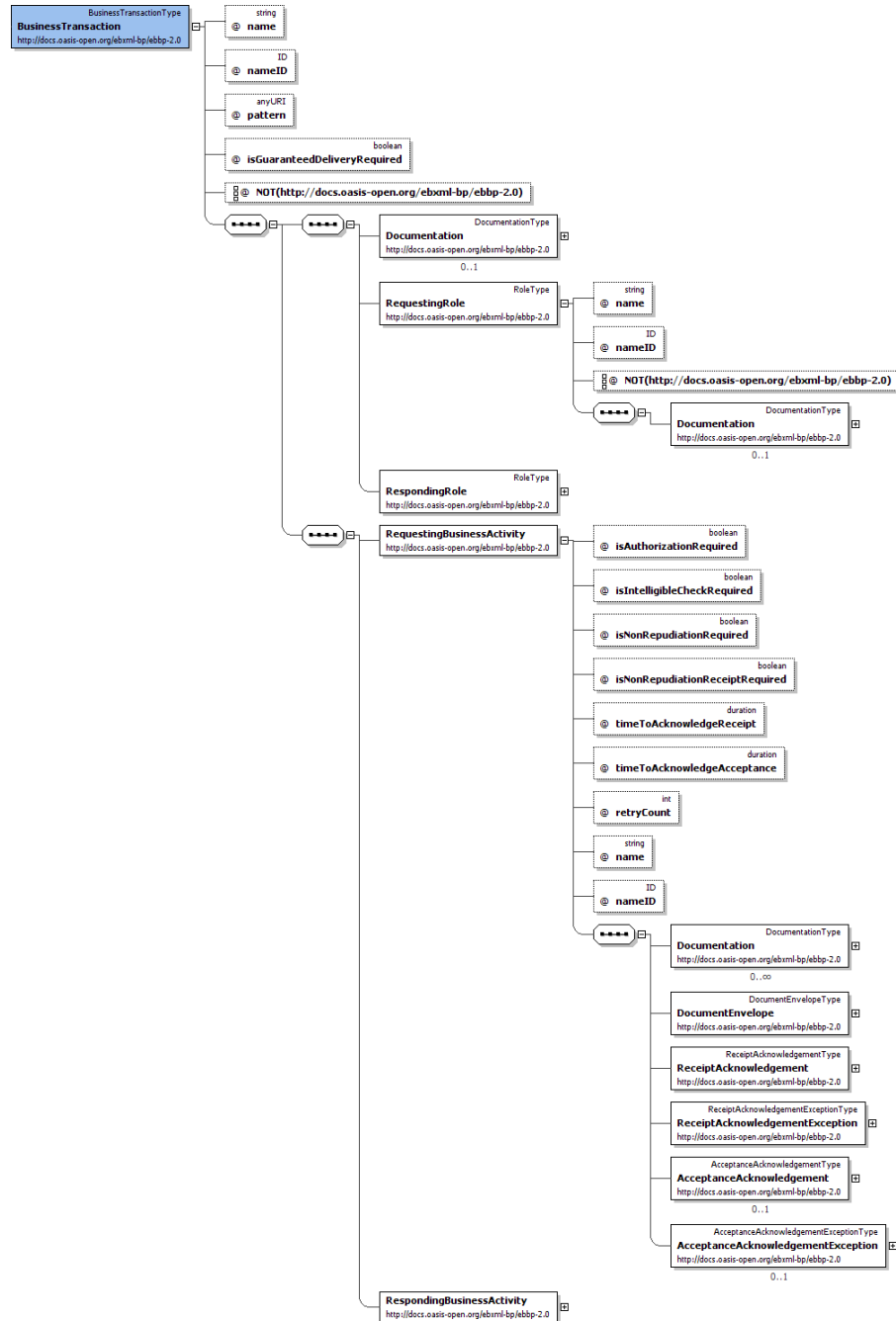


Figure B.3: High level view of the Business Transaction

Schema of the business transaction activity is given in Figure B.4.

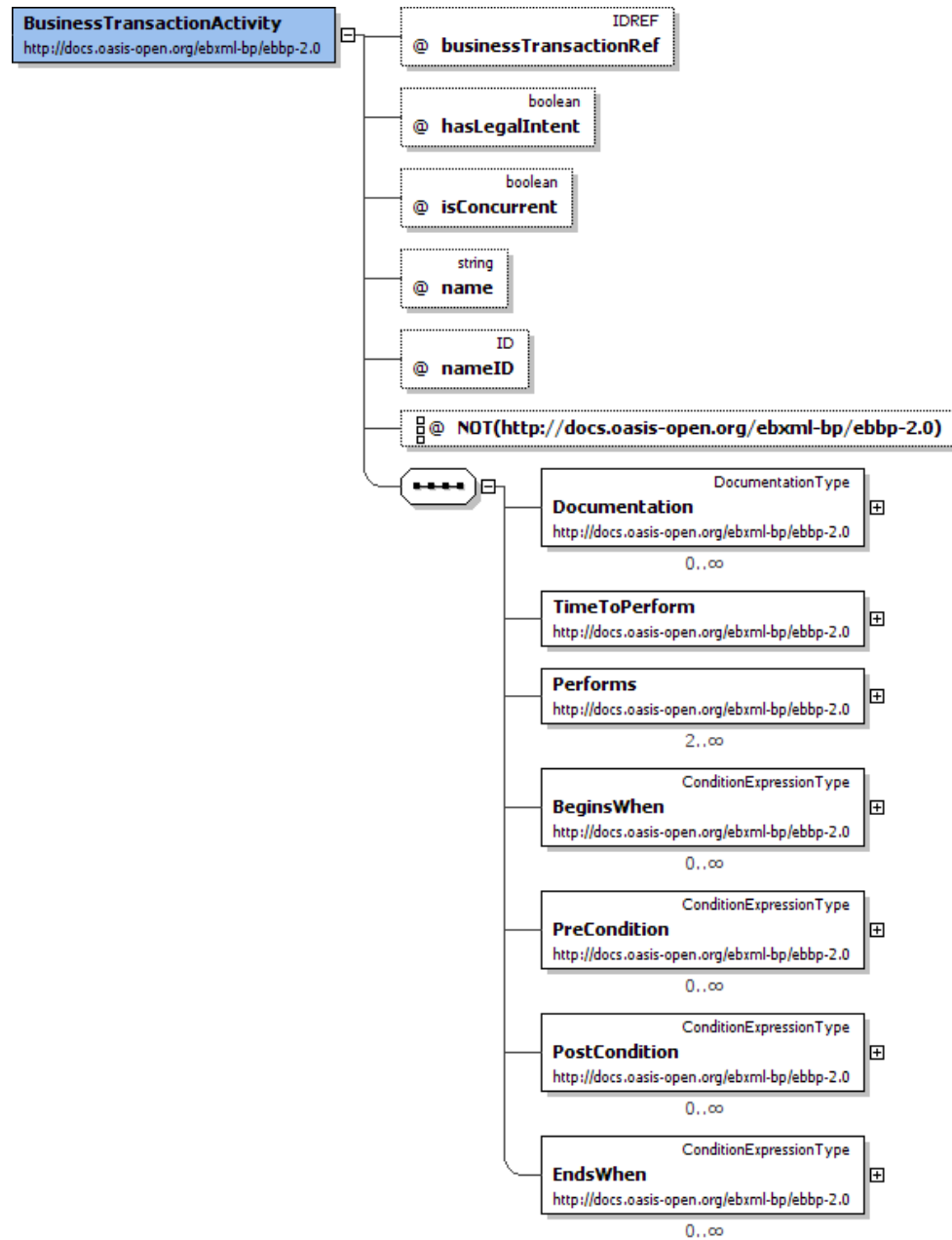


Figure B.4: Graphical view of the schema of the Business Transaction Activity

The ebBP notation for the complex business transaction activities is given in Figure B.5.

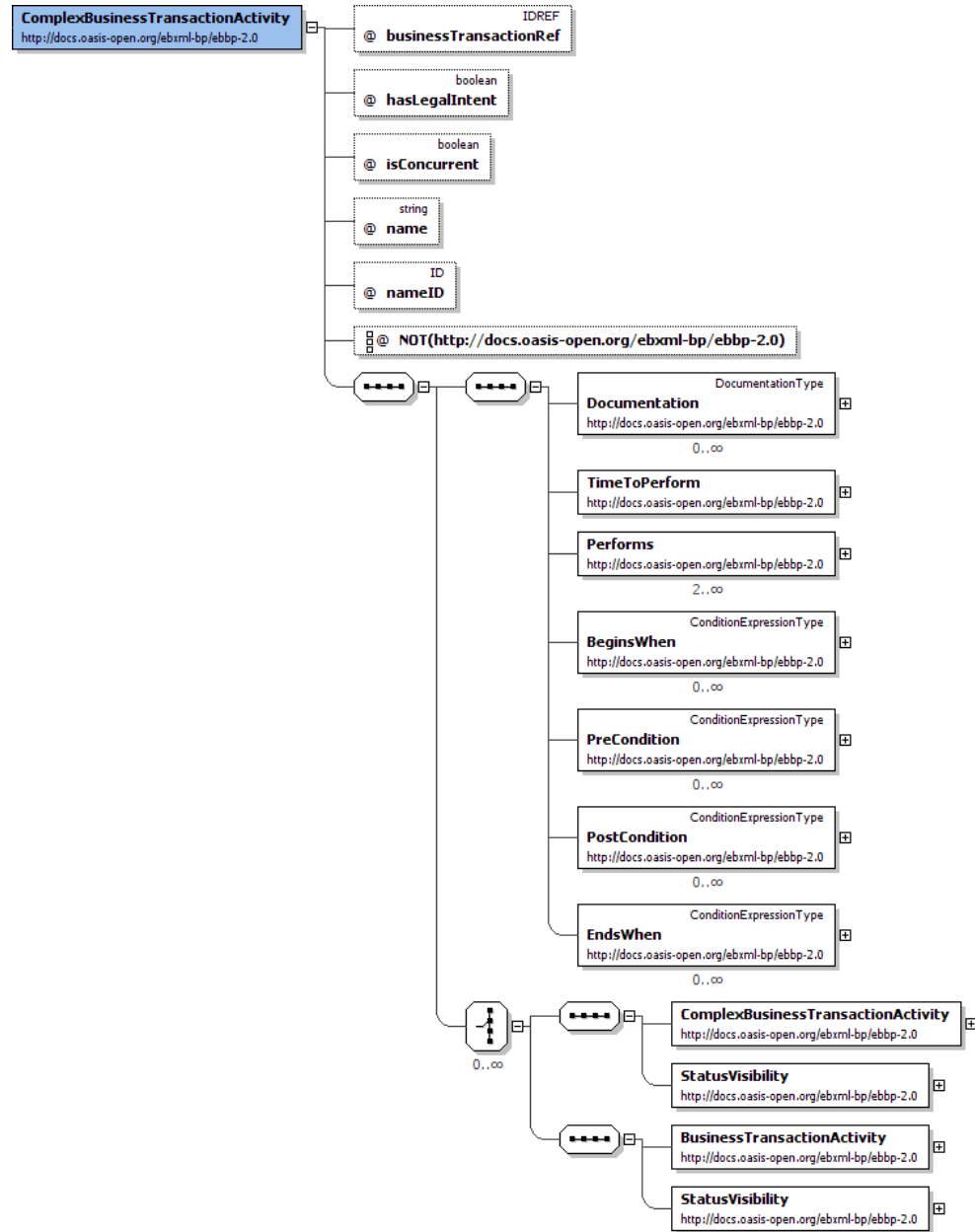


Figure B.5: Schema definition of the Complex Business Transaction Activity

High level view of the Receipt Acknowledgement signal can be found in Figure B.6.



Figure B.6: Model view of the Receipt Acknowledgement signal

The defined schema for exceptions is shown in Figure B.7.

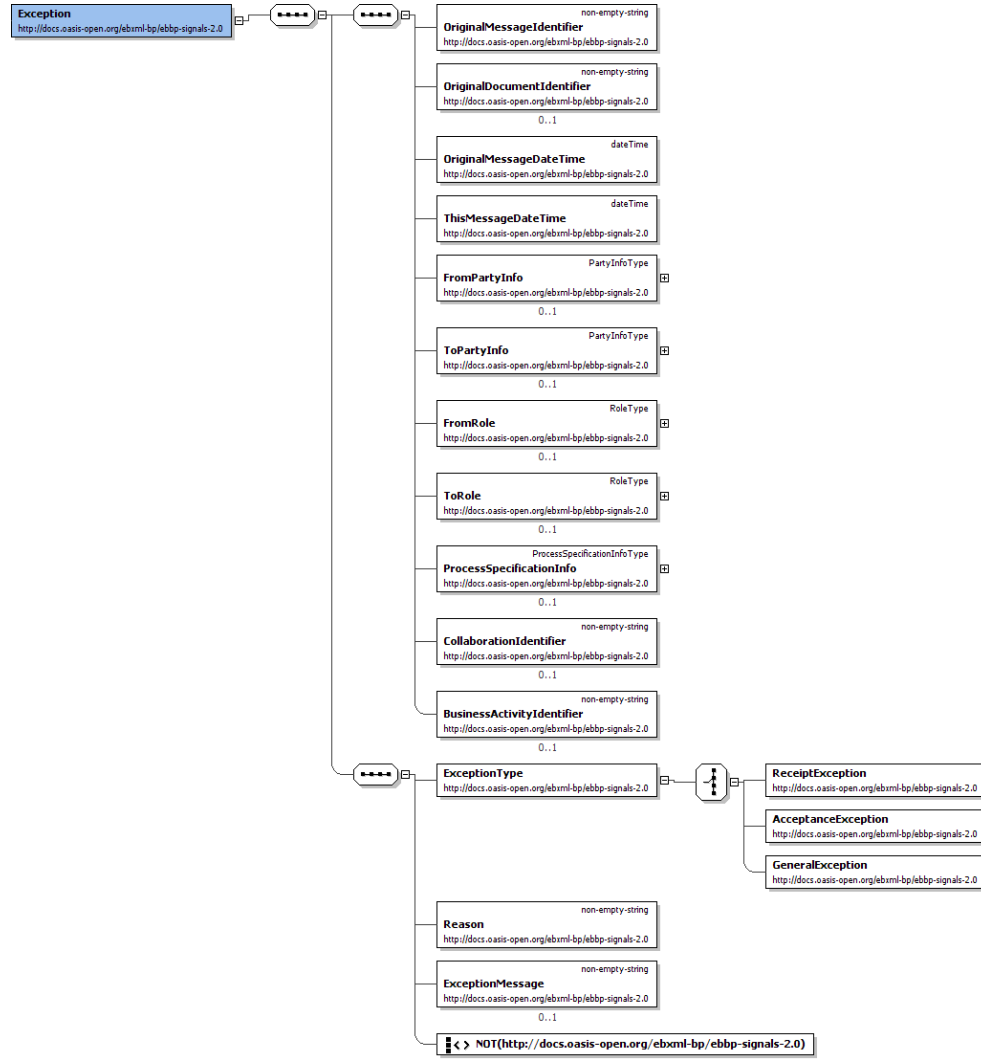


Figure B.7: Schema of the exception elements found in ebBP documents

Schema for business collaborations is given in Figure B.8.

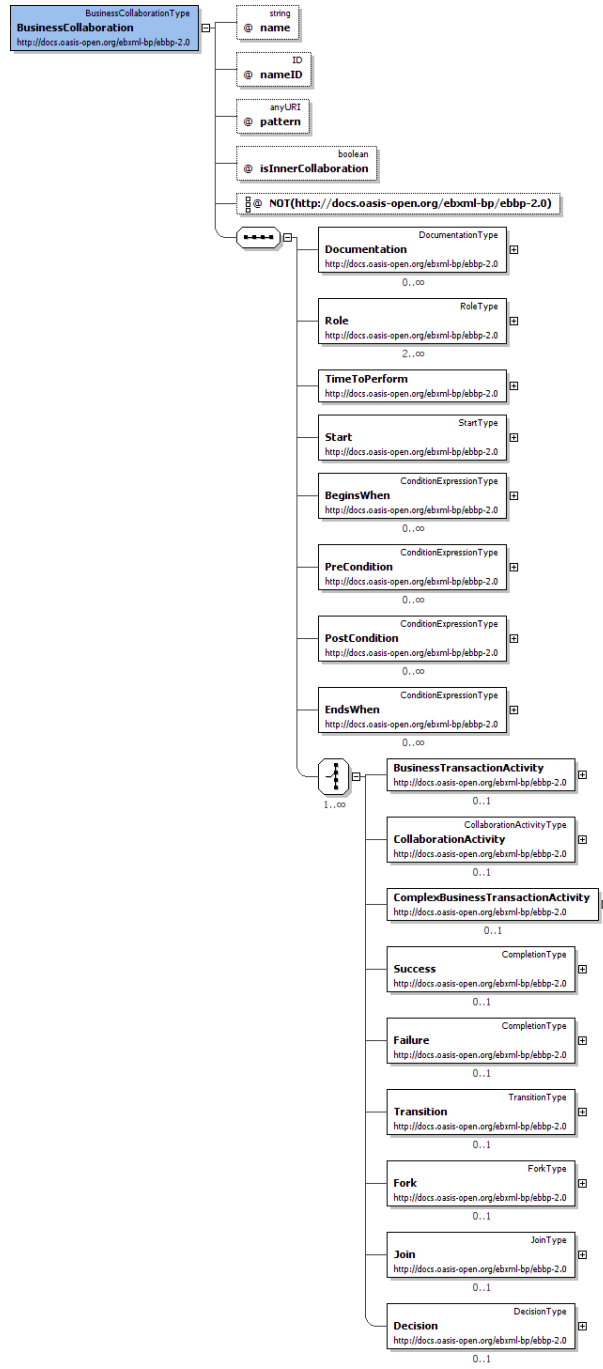


Figure B.8: Schema definition of the Business Collaboration