



THE HONG KONG
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library

包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

The Hong Kong Polytechnic University

Department of Computing

**An Ontology-based Modeling Methodology for
Service-Oriented Architecture**

LIAO Li

**A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy**

December 2008

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

LIAO Li _____ (Name of Student)

Abstract

With the rapid growth of the software industry, Service-Oriented Architecture (SOA) has been considered as a new paradigm for system development and integration. By using services to encapsulate functionalities of business tasks and providing standard communication between services, SOA provides a design framework for realizing rapid and low-cost system development and improving total system quality. SOA modeling is the initial phase of SOA development lifecycle and the quality of the SOA model will directly affect the quality of SOA application.

Applying ontology techniques to SOA modeling can provide accurate descriptions for models, identify the binding information of business process and service, increase the reusability of existing business processes and services, and accelerate application development. In this thesis, we adopt ontology techniques to assist SOA modeling, developing a core ontology BPO (Business Process Ontology) for business process modeling as well as proposing an ontology-based SOA application modeling and developing framework. BPO can provide accurate definitions of the main components of SOA modeling. Its extension on a specific domain can help to construct a knowledge base for business process modeling, describing processes and services and defining their mappings. As such, we propose four modeling methods for SOA development: TDM (Top-Down Modeling) supports developers to directly create new process models; TDM-RP (Top-Down Modeling based on Reusable Process) supports developers to construct new process models by reusing similar process models already defined in the knowledge base; BUM-RS (Bottom-Up Modeling based on Reusable Services) supports developers to construct

new process models by reusing identified services; and AM-RPRS (Agile Modeling based on Reusable Process and Reusable Services) provides specific queries on both the business process and identified services, and enables developers to efficiently choose suitable models. The modeling methods are verified by extending BPO to the automotive software development domain, constructing a knowledge base AutoPO (Automotive Process Ontology), and applying AutoPO to simulate the execution of the modeling method with a series of case studies. Based on a survey of the quality requirements for models, we also propose a set of quality attributes for SOA models.

Publications Arising From the Thesis

1. Liao, L., Leung, H. K. N. SOA-based Process Modelling for Automotive Software Development. Submitted to Journal of Systems and Software.
2. Liao, L., Leung, H. K. N. 2007. Testing Techniques for SOA Model. Proceedings of the International Conference on Software Engineering and Data Engineering (SEDE-2007), USA, 9-11 Jul. 2007
3. Liao, L., Leung, H. K. N. 2007. An Ontology-based Business Process Modeling Methodology. Proceedings of the IASTED International Conference on Advances in Computer Science and Technology (ACST 2007), Thailand, 02-04 Apr. 2007
4. Leung, H.K.N., Liao, L. and Qu, Y.Z. 2007. Automated Support of Quality Improvement. Journal of Quality and Reliability Management. vol. 24, no. 3, pp.230-243.

Acknowledgements

I wish to express my gratitude to everyone who contributed to making the dissertation a reality.

First and foremost, I would like to express my deepest thanks to my supervisor Dr. Hareton Leung, who gave his full effort to this research and supported me during the years to bring it to fruition. And, I would like to thank all of the teammates in Dr. Leung's group for their continuous support and kind help.

I would like to thank Prof. Keith Chan, Prof. Francois Coallier and Dr. Yuen Tak Yu, for their insightful comments and helpful advice to improve the quality of my thesis. I would like to thank Dr. Yuzhong Qu for his helpful suggestions at the beginning phase of this research. I am grateful to Ms. Yvette Lui, who corrected the wording of this work. I am again grateful to my friends for their encouragement and help.

Finally, I would like to express my deepest appreciation to my family for their constant support and patience during the years to make my dream come true.

Table of Contents

CERTIFICATE OF ORIGINALITY	I
Abstract.....	II
Publications Arising From the Thesis	IV
Acknowledgements	V
Table of Contents	VI
List of Figures.....	IX
List of Tables.....	X
Chapter 1 Introduction.....	1
1.1 Background	1
1.2 Motivation and Objectives	3
1.2.1 Motivation.....	3
1.2.2 Objectives and Methodology	7
1.3 Contributions.....	9
1.4 Organization.....	11
Chapter 2 Literature Review	12
2.1 SOA Overview	12
2.1.1 Definitions of SOA	14
2.1.2 SOA Development Lifecycle	16
2.1.3 Layers of SOA	17
2.1.4 SOA Delivery Strategies	20
2.2 Related Works of SOA Modeling	23
2.2.1 Standards for SOA Modeling.....	24

2.2.2	Methods for SOA Modeling.....	29
2.2.3	SOA Modeling and Developing Frameworks.....	35
2.3	Quality Attributes for Models.....	38
2.3.1	Attributes of Engineering Models.....	38
2.3.2	Quality Attributes Defined in ISO 9126.....	39
2.3.3	Quality Attributes for UML-based Models.....	41
2.3.4	Guidelines to Improve Quality of Information Models.....	42
2.4	Related Works of Automotive Software Modeling.....	43
2.4.1	Overview of Automotive Software.....	43
2.4.2	Features of Automotive Software.....	47
2.4.3	Automotive Software Modeling Methods.....	49
2.5	Summary.....	57
Chapter 3	Ontology-based SOA Modeling Methodology.....	58
3.1	Ontology Definitions.....	59
3.2	Business Process Ontology (BPO).....	67
3.2.1	Concept Set of BPO.....	68
3.2.2	Relation Set of BPO.....	72
3.2.3	The Formal Description of BPO.....	78
3.2.4	Axioms for BPO.....	81
3.2.5	Comparison of BPO and OWL-S.....	84
3.3	Ontology-based Business Process Modeling and Developing Framework.....	86
3.4	Modeling Methods.....	89
3.4.1	Notations.....	89
3.4.2	The Modeling Processes.....	92

3.4.3	Top-Down Modeling (TDM)	97
3.4.4	Top-Down Modeling based on Reusable Processes (TDM-RP) 100	
3.4.5	Bottom-Up Modeling based on Reusable Services (BUM-RS) 105	
3.4.6	Agile Modeling based on Reusable Process and Reusable Services (AM-RPRS).....	106
3.5	Summary	109
Chapter 4	Validation and Verification	110
4.1	Validation of Modeling Methodology.....	110
4.1.1	Extension of BPO for Automotive Software Modeling.....	111
4.1.2	Case Studies	121
4.2	Comparison of the Automotive Software Modeling Methods.....	147
4.3	Verification of SOA Models.....	151
4.3.1	Quality Attributes for SOA Models	151
4.3.2	Verification of Our SOA Models	155
4.4	Summary	161
Chapter 5	Conclusion and Future Work.....	162
Bibliography		169
Appendix A: Glossary		183
Appendix B: Framework of AutoPO		186

List of Figures

Figure 1. SOA development lifecycle (High et al., 2005).....	16
Figure 2. The layers of SOA (Arsanjani, 2004)	18
Figure 3. Development flow of Model Driven Service-Oriented Architecture	36
Figure 4. Steps of ontology based software development.....	37
Figure 5. Layered architecture of AUTOSAR (2006a)	53
Figure 6. Framework of BPO.....	68
Figure 7. Ontology-based Business Process Modeling and Developing Framework	86
Figure 8. Ontology-based modeling flow	95
Figure 9. Extension of BPO for automotive categories	114
Figure 10. Extension of BPO for automotive software components' ports	117
Figure 11. Wiper/washer process model	124
Figure 12. Sequence graph of a wiping process.....	126
Figure 13. Business Process model: WiperWasherSystem_V1	126
Figure 14. Searching for reusable processes	130
Figure 15. A partial process model of WiperWasherSystem_V2.....	132
Figure 16. Searching with service information (I).....	133
Figure 17. Searching with service information (II).....	134
Figure 18. Combined search for the processes implemented by “RainSensingService_V1”	135
Figure 19. Searching the processes for “Washer”	138
Figure 20. LampWasherSystem model	140

List of Tables

Table 1. Comparison of SOA modeling methods	34
Table 2. Quality attributes and sub-attributes in ISO 9126.....	40
Table 3. Comparison of automotive software modeling methods	54
Table 4. Hierarchical relationships between the concepts in BPO	72
Table 5. Main object properties in BPO.....	74
Table 6. Main data type properties in BPO.....	76
Table 7. Relation hierarchies between the properties in BPO	76
Table 8. Definitions of abbreviations.....	91
Table 9. Comparison of different modeling methods	97
Table 10. Comparison of the four modeling methods.....	108
Table 11. New class definitions in AutoPO	114
Table 12. Hierarchical relationships between the concepts in AutoPO	118
Table 13. Main object properties in AutoPO	119
Table 14. A partial list for properties of concrete services.....	128
Table 15. The developed process models in Case 1-4	141
Table 16. Concrete process models.....	142
Table 17. Comparison between logical models	146
Table 18. Comparison of automotive software modeling methods	148
Table 19. The ISO 9126 Quality attributes applicable to SOA model.....	152
Table 20. Mapping of the quality attributes	153
Table 21. Category of possible errors when developing ontologies	156

Chapter 1 Introduction

1.1 Background

In the past decades, different software architectures have been proposed and practiced to deal with the growing software complexity. Using the structured system analysis and design method, developers can solve the early problems of complexity by choosing the right data structures, developing appropriate algorithms, and modularizing various system functions. After the appearance of Object-Oriented analysis and design method and Component-based development, developers can handle more complex problems, and software can be partially reused to solve the code redundancy problem.

In the 1990s, with the maturity of computer networks, most enterprises work with a systemic infrastructure of multiple heterogeneous systems and may need to integrate them. Architectures, such as CORBA (Common Object Requesting Broker Architecture) (OMG, 2004) and DCOM (Distributed Component Object Model) (Microsoft, 2007), can be used to handle the communication problems among software components distributed across networked computers. They are however not widely accepted, because they require every participant in the distributed system to use the same technology (High et al., 2005).

From the viewpoint of system engineering, Enterprise Architecture (EA) is defined as a coherent set of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure (Lankhorst et al., 2005). Enterprise

architecture captures the essentials of the business, IT and its evolution, and provides a holistic view of the enterprise. The Open Group Architecture Framework (TOGAF) (The Open Group, 2009) is one of the most popular enterprise architecture frameworks, which provides architectural framework, architecture development methodology and relevant resources for organizations.

Nowadays, developers begin to use web service technology and Service-Oriented Architecture (SOA) to solve the integration problems of distributed and heterogeneous systems. SOA is also seen as a style of architecture associated with the application architecture of an enterprise architecture. According to W3C's definition (W3C, 2004a), a service is an abstract resource to represent a capability of performing tasks that represents a coherent functionality. A service performs one or more tasks, and the provider and consumer of the service can be different persons or organizations. A service has a service interface and can be accessed over a network. The communication between the services is standard-based (such as Simple Object-based Access Protocol, SOAP). Hence, a service can be considered as a black box that completely hides the underlying implementation and simply offers the execution of a certain behavior.

SOA is an architectural concept for describing distributed systems (W3C, 2004a), and it can also be viewed as a paradigm for organizing and utilizing distributed capabilities that may be provided by different owners (OASIS, 2006). In SOA, service can be considered as a black box for business driven functional units, and can be invoked across networks to provide flexible enterprise application integration (Stojanovic et al., 2004). Generally, SOA considers services in the context of business functions with specific business behaviors rather than as technical software entities.

Compared with the traditional EA (e.g. TOGAF), SOA is a discipline that spans the entire spectrum from business architecture to IT implementation (Bercovici et al., 2008). SOA enables agile businesses through business processes and services (Zhao, 2006). According to IBM, the primary goal of SOA is to align the business world with the IT world in a way that makes both more effective (High et al., 2005). SOA can be applied to the full spectrum of enterprise business and IT, which include business service specification, IT strategic planning, enterprise architecture, solution development, business implementation and business monitoring. SOA can also be considered as a practical modeling approach for enterprise architecture development. It can help to bridge EA with a solution architecture and implementation by layered service descriptions across business modeling, application modeling, and technology implementation; hence it can help bring EA into reality.

Compared with the traditional integration techniques (e.g. CORBA and DCOM), SOA-based solutions can provide many benefits such as simplicity, reusability, standard-based, flexibility, low cost, efficiency and dynamic systems (Bouras et al., 2007).

1.2 Motivation and Objectives

1.2.1 Motivation

SOA modeling is the initial phase of the lifecycle of SOA development (High et al., 2005). The target of SOA modeling is to capture the business requirements, create a specification of business processes, goals and assumptions, and design an encoded model. To support different views of business, SOA modeling includes two

kinds of modeling: one is service modeling, constructing models for service implementation; the other is business process modeling, using services as the basic building blocks to compose an SOA system. The former considers the design of a service from a technical view, pays more attention to the interfaces and the communication of the services; while the latter considers the design from a higher level viewpoint, considering the targets, strategies, and workflow of business processes. In business process modeling, local and remote business behaviors represented by (web) services reach a level of technical abstraction, and the software structure and control-flow are closely related to the business goal. This introduces a business-oriented approach to easily and flexibly designing and structuring software systems. In our study, we focus on the business process modeling. Therefore, SOA modeling is taken to be equivalent to business process modeling in the remainder of the thesis, if not specified otherwise.

The business process modeling is not an invention brought about by SOA. Traditional Business Process Modeling has been investigated for years before the appearance of SOA. Researchers and organizations have developed business process description languages, such as PSL (Process Specification Language) (ISO, 2004) and BPMN (Business Process Modeling Notation) (OMG, 2008), and patterns (Rozman et al., 2004; van Dongen et al., 2006) to describe the business workflow. Originally, the process-oriented modeling was used for re-structuring business applications, integrating new processes and continuously monitoring system performance (Karagiannis et al., 1996), and was mainly used in the context of Business Process Reengineering, Workflow Management and Supply Chain Management (Becker et al., 2000). With the acceptance of service and SOA in software industry, researchers began to use service concepts in new patterns for

business processes (Mahleko et al., 2006; Zdun et al., 2007); however, they only adopted the concept of service in their modeling; they continued to focus on the vision from the business aspect and did not provide a complete view that includes both business processes and services.

After the appearance of SOA, several Model Driven Architecture (MDA)-based methodologies for SOA modeling have also been proposed (Gardner, 2003; Torres et al., 2005; Wada et al., 2006; and Zhou et al., 2008). Unified Modeling Language (UML) diagrams are used to capture business visions and model services as well as their choreography. Applying MDA to SOA modeling can accelerate SOA's formalization and automation. It supports separating conceptual concerns from implementation-specific concerns. However, MDA is not suitable for dynamic application environment, and it cannot be queried nor reasoned about (Tetlow et al., 2006). There is, for example, no way to ask the MDA system whether some configuration is valid or more elements are needed. Therefore, using MDA only is not effective for SOA modeling.

Besides using UML to model services and processes, researchers also use service composition techniques for business processes modeling and implementation. Several standards and languages have been developed for web service composition, such as WSBPEL (Web Services Business Process Execution Language) (OASIS, 2007), WSCL (Web Services Conversation Language) (W3C, 2002a), WSCDL (Web Services Choreography Description Language) (W3C, 2005a) and WSCI (Web Service Choreography Interface) (W3C, 2002b). These standards and languages can be used to construct the high-level specification of a complex business process and represent the interactions between services. However, to construct business models using these standards, developers face a steep learning curve (including these

standards and other related standards, such as WSDL (Web Services Description Language) (W3C, 2001a)). This may cause difficulties because the business process model must be understandable by all the stakeholders, including the end users who will probably not be able to understand these standards and languages.

In recent years, Ontology has been recognized as a useful technique for business process modeling and management by providing formal descriptions, reasoning functions and extensible knowledge base. Ontologies can provide formal description for the models and their relationships, and support the model information query, sharing and reusing of this information. Researchers have proposed different ontologies to support business process and service modeling. For example, domain ontologies can provide knowledge supporting, which may include the concepts and relations of the domain specific terms, for the application modeling (Kuziemsky et al., 2003; Liu et al., 2007). Based on the Web Ontology Language (OWL), upper ontologies have also been proposed, such as the OWL for Processes and Protocols (OWL-P) (Mallya et al., 2005; Desai et al., 2005), Task ontology language (OWL-T) (Tran et al., 2007) and the OWL for Services (OWL-S) (W3C, 2004e). These ontologies can capture the general concepts and relationships in a model. OWL-P and OWL-T can be used for business process description; OWL-S can be used for describing the properties and capabilities of web services. However, most modeling frameworks usually apply these ontologies separately, which may cause inconsistency.

To build high-level and abstract models of the business goals, a modeling methodology should be able to identify the reusable components and integrate different parts of the business (Graham, 2006). To identify the reusable components, the modeling methodology should be able to identify the common services and the

models related to the services from the repository of an organization; to support seamless business operations, business rules for the integration should also be specified.

This can be considered as common functional requirements for the business process modeling. Besides this, business process modeling should enable non-technical domain experts to participate in the SOA model design, which means that the business model should be understood by both the developers in business and IT domains. Therefore, an easy-to-understand formal description language for capturing the business design is needed.

Although most of the process modeling languages and methods are formalized or semi-formalized, they usually can only satisfy part of the above requirements. For example, all the modeling methods introduced previously support the integration of different parts of the business; however, few of them provide the function to identify common services or reusable components. These methods usually need to cooperate with other techniques, such as service discovery techniques, to satisfy the above requirements. Although this provides a way to solve the problem, other difficulties emerge, such as: how to choose a service discovery method; can the modeling method combines seamlessly with the service discovery method; how to ensure the security of the discovered service; etc.

1.2.2 Objectives and Methodology

In view of the limitations and problems of previous business process modeling approaches, the main objective of this research is to provide a new formal modeling methodology to satisfy the flexible requirements of the business process modeling in SOA development.

To achieve this objective, four aspects will be considered:

1) Selecting a suitable formal modeling language for constructing business process models in SOA development.

Capturing business design using a rigorous approach offers the potential to gain better insight into business. Formal modeling languages can describe the business processes precisely, and would not cause confusion to the developers. A suitable formal language should be understandable by both the developers and computer, so that computer-aided tools can help to speed up the system development and provide automatic checking and management. Another requirement for the modeling language is that it should support the maintenance of the business process models, because the models may need to be changed during their usage. Finally, the language should also be easy to learn and easy to use.

In our study, we use ontologies to represent business process models and adopt OWL (W3C, 2004c) as our modeling language. The concept definition of ontology can help to define the business processes and services accurately; the extensibility and inheritance properties of ontology can help to reuse processes; and the reasoning of ontology can help to identify suitable services for a business process. If ontology can be used to represent both business processes and business process model components, we may develop formal descriptions for them, and use reasoning functions to assist in the SOA modeling.

OWL is an ontology language which is more expressive than other ontology languages. Although OWL is Extensible Markup Language (XML)-based, it provides higher machine readability than XML. There are many ontology editors which support OWL, for example, Protégé (2008) provides a visual tool for ontology construction, with which, developers can construct and manage their

knowledge base.

2) Identifying the information and properties of the business process model to form the basis for SOA system development.

In this part, we abstract the common features of business processes and services to construct the business process concept and service concept of the ontology. The relationships between the concepts are also defined. This builds up the basic framework for the knowledge base of business process modeling.

3) Proposing relevant modeling framework and modeling methods.

In practice, an organization may have many different modeling requirements. For new development, the developers may analyze the business targets and processes first, create process models and then implement the system. Alternatively, the organization may have a large asset base of existing services available for reuse through years of development. Then, its developers may build a new system by integrating some existing business processes, or reusing some identified common services. To satisfy the different requirements, different modeling strategies and methods need to be developed.

4) Validating whether the proposed modeling methods can facilitate the requirement analysis and system design.

We will apply our methodology to a specific scenario and use several case studies to validate the methods.

1.3 Contributions

The contributions of this study are listed as follows:

1) A core ontology (Business Process Ontology) for Business Process modeling

is designed to describe business processes and their component services. On the basis of the core ontology BPO, a knowledge base can be constructed to present and manage the information of software modeling. These ontologies can provide sharable and precise description of the process models and services and support the knowledge management of the process model and service asset bases.

2) An Ontology-based Business Process Modeling and Developing Framework (OBPMDF) is presented. This framework illustrates the modeling and assembling phases of SOA development lifecycle. It applies the extension of BPO to work as the knowledge base, providing accurate descriptions for business processes, services and their relationships.

3) On the basis of the extension of BPO and OBPMDF, four modeling methods which address different requirements of modeling are presented. TDM (Top-Down Modeling) allows developers to create new process models directly; TDM-RP (Top-Down Modeling based on Reusable Process) allows developers to construct new process models by reusing similar process models already defined in the knowledge base; BUM-RS (Bottom-Up Modeling based on Reusable Services) allows developers to construct new process models by reusing identified services; and AM-RPRS (Agile Modeling based on Reusable Process and Reusable Services) provides specific queries on both the business process and identified services, and can facilitate developers to efficiently choose suitable models.

4) The modeling methods are verified by extending BPO to the automotive software development domain, constructing a knowledge base AutoPO, and applying AutoPO to simulate the execution of the modeling methods with a series of case studies. In the extension of BPO, domain specific knowledge for automotive software development is added to the ontology, so that the knowledge can be well

structured. The case studies demonstrate how the business process models can be constructed step by step.

5) Based on a survey of different quality models, a set of key quality attributes for SOA models are proposed. We also discuss how our methodology can support the quality attributes for SOA models.

1.4 Organization

The remainder of the thesis is organized as follows: Chapter 2 gives details of background information and related works, including the SOA modeling methods and framework, model quality attributes, and modeling methods for automotive software. Chapter 3 presents the proposed framework of BPO (Business Process Ontology), illustrates an Ontology-based Business Process Modeling and Developing Framework (OBPMDF) and presents four modeling methods that exploit three well known modeling strategies, Top-Down, Bottom-Up, and Agile Modeling, each suited to a different kind of modeling requirements. In Chapter 4, we validate our framework and modeling methods by constructing a knowledge base AutoPO (Automotive Process Ontology) for automotive software modeling on the basis of BPO and using several case studies to demonstrate their usage. We also propose a set of quality attributes for SOA models and discuss how our methodology supports the attributes. The final chapter concludes the thesis and identifies some directions for future research.

Chapter 2 Literature Review

In this chapter, we will introduce the SOA-related concepts first; then we will present the related works on SOA modeling methods and frameworks as well as the quality attributes for models. Since we will validate our modeling methodology by applying it to the automotive software development domain, this chapter also reviews the related works for the automotive software modeling.

2.1 SOA Overview

The concept of Software Architecture as a practice in the field of software engineering has continuously evolved to deal with the increasing complexity of today's software systems. The software architecture reflects the system structure, which is comprised of software components, their externally visible properties and their relationships (Bass et al., 2003). Software Architectures are used to describe how these components interact on a high level and provide a structural and behavioral view of the system (McGovern et al., 2003).

With the introduction of Object-Orientation, the real world business logic can be represented by concept models which are described by classes, objects, attributes and methods. Compared to the older structural programming concept, Object-Oriented Analysis and Programming can partially solve the increasingly complex and quality problems of software by using discrete units of programming logic which greatly enhance software reuse. Objects can provide programming language level abstractions.

Component-based technologies have also been developed to facilitate the creation of complex, high-quality systems, internally managing complexities and dependencies of software components. Components are essentially larger grain abstraction of objects, in which typically a group of objects come together to provide a business functionality that is required by an application (Herzum et al., 2000). However, composing software from software components still requires knowledge about the underlying object models which are programming language dependent.

It is difficult for these technologies to support the development of complex, distributed and heterogeneous systems because of their dependency on the platforms and programming languages. Even architectures, such as CORBA (OMG, 2004) and DCOM (Microsoft, 2007), which are specific for distributed systems communication and integration, have strict syntax and semantics requirements for the participating systems (High et al., 2005). Furthermore, the component-based technologies have difficulty in supporting unrestricted portability and platform independence.

In a Service-Oriented approach, an SOA application consists of self-contained business-oriented software blocks that are accessible over networks and solely need to describe their service interface in a well-understood manner (W3C, 2004a). With well-defined interfaces, services can be described, discovered and used by external users and accessed via a standardized mechanism. With services representing business functionality, SOA strives at easing the design of business processes and provides an architectural solution that enables flexible business execution and business partner integration.

In this section, we present detailed background information for our study, such as SOA definitions, SOA development lifecycle, SOA layered structure and delivery strategies for SOA development.

2.1.1 Definitions of SOA

Common definitions of SOA are:

- **OASIS's definition** (OASIS, 2006): SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.
- **W3C's definition** (W3C, 2004a): SOA is a form of distributed systems architecture that is typically characterized by the following properties: logical view, message orientation, description orientation, granularity, network orientation, and platform neutral.

These two definitions present the SOA from different points of view. SOA can be a design style to guide the creation and use of business services throughout their lifecycle (from conception to retirement), and can also be a way to define an IT infrastructure to support different applications to exchange data and participate in business processes, regardless of the underlying operating systems or programming languages (Newcomer et al., 2004).

Service is an important concept for SOA. *Service* normally denotes the provision of a general business activity which provides a certain value to the customer in a business domain (Baida et al., 2004). In computer science, *service* is a software component of distinctive functional meaning that typically encapsulates a high-level business concept (Krafzig et al., 2004). In SOA, *service* is often used synonymously with web service, which can be considered as a self-contained and self-describing business driven functional unit, and can be invoked across networks to provide flexible enterprise application integration (Stojanovic et al., 2004).

Service encapsulates the logic within a distinct context for business which could

be a specific business task, a business entity, or other logical grouping (Erl, 2005). Varying amounts of logic can be encapsulated into services and be considered as independent artifacts accessed in a standardized way. The application developers or system integrators can build applications by composing one or more services, without having to know their underlying implementation.

The main features of SOA can be summarized as follows (Newcomer et al., 2004).

- **Loosely Coupled:** In SOA, services are the mechanisms by which needs and capabilities are brought together. The use of services establishes a loosely coupled environment that runs contrary to many traditional distributed application designs. If properly designed, loosely coupled services can support a composition model, allowing several individual services to be integrated into an SOA application. This introduces continual opportunities for reuse and extensibility.
- **Diversely Owned:** SOA applications may be composed of services which are owned and operated by outside organizations. Diverse ownership implies that the published service interface will be treated as a black box from the standpoint of the programmers since they cannot penetrate the interface and modify its code and behavior.
- **Interoperable:** Standards (such as SOAP, XML, UDDI, etc) ensure that services from differing organizations can use each other's services. SOAP (Simple Object-based Access Protocol) (W3C, 2007) provides the messaging format used by service and service requester. XML (W3C, 2006) supports the data representation of SOA. UDDI (Universal Description, Discovery and Integration) (OASIS, 2004) provides an industry standard

for service registration and discovery.

These features brought some significant benefits: With SOA, software organizations can reuse business processes and services, facilitate the manageable growth of large scale enterprise systems, and reduce development costs (OASIS, 2006).

2.1.2 SOA Development Lifecycle

According to IBM (High et al., 2005), the SOA lifecycle can be divided into four phases: Model, Assemble, Deploy and Manage. The four phases are layered on a backdrop of a set of governance and processes to ensure the compliance, and feedback is cycled to and from phases in iterative steps of refinement in the lifecycle, as shown in Figure 1.

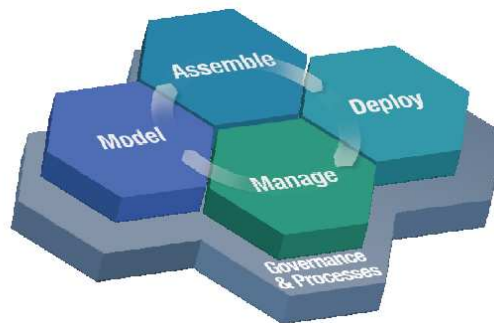


Figure 1. SOA development lifecycle (High et al., 2005)

The *Model* phase is the process of capturing the business design of an organization, translating that into a specification of business processes, goals and assumptions, and finally creating an encoded model of the business (Newcomer et al., 2004). The SOA solutions should ensure that the design can satisfy the organization's business requirements and objectives.

During the *Assemble* phase, software organizations should take actions to

design and implement the modeled business processes and services. These actions include searching the existing asset base inventories to find reusable application components, create or purchase new services.

In the *Deploy* phase, after the deployment of applications, a hosting environment for the applications will be created, which resolves the application's resource dependencies, operational conditions, capacity requirements, and integrity and access constraints. In the *Manage* phase, software organizations need to consider how to maintain the operational environment and the policies expressed in the assembly of the SOA applications deployed to that environment.

The *Model* phase is the initial phase of the lifecycle, and forms the basis of the latter phases. Therefore, suitable modeling methodology and qualified models are very important for SOA development.

2.1.3 Layers of SOA

The importance of modeling in SOA development can also be reflected by the architectural structure of an SOA. Generally, enterprise logic can be divided into two layers from an IT perspective, *business logic layer* and *application logic layer* (Erl, 2005). *Business logic layer* is generally structured into business processes to express the requirements, associated constraints, dependencies, and outside influences. *Application logic layer* is the implementation of the business logic, which can be purchased or custom-developed systems that express the business processes within the confines of an organization's IT infrastructure, security constraints, technical capabilities, and vendor dependencies.

With SOA, a new layer, *service interface layer*, is added to the enterprise logic (Erl, 2005). *Service interface layer* wedges between traditional business and

application layers and establishes a higher form of abstraction which encapsulates the physical application logic and business process logic of the applications. Three layers of abstraction are identified in the service interface layer; they are *application service layer*, *business service layer* and *orchestration service layer*. The *application service layer* contains the foundation level services to express technology-specific functionality. The *business service layer* expresses the business logic through service-orientation and brings the representation of corporate business models into the web services arena. The *application service layer* is platform-concerned and the *business service layer* solely concerns business logic. The *orchestration service layer* concerns the workflow management and composes the business services to provide specific sets of functions.

Besides this layered architecture, IBM (Arsanjani, 2004) also proposes an architectural template for SOA, which mainly contains five layers. The layers are *operational systems layer*, *enterprise components layer*, *services layer*, *business process choreography layer* and *presentation layer*. QoS (Quality of Service), security and monitoring act on each of the five layers. The architecture is shown in Figure 2.

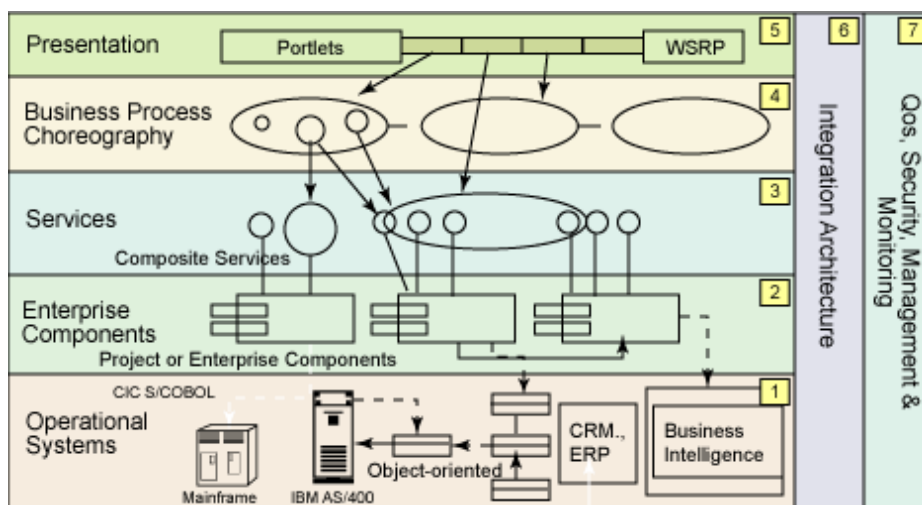


Figure 2. The layers of SOA (Arsanjani, 2004)

The *operational systems layer* consists of existing applications, which may include existing CRM (Customer Relationship Management) and ERP (Enterprise Resource Planning) packaged applications, object-oriented system implementations and business intelligence applications. This layer can be viewed as the implementation of application, and can map to the application logic layer of Erl's enterprise logic (Erl, 2005).

The *enterprise components layer* is responsible for realizing functionality and maintaining the QoS of the exposed services. This layer can map to the application service layer in Erl's enterprise logic (Erl, 2005). The services in this layer compose the enterprise service asset bases.

The *services layer* represents the services that the business chooses. In this layer, services can be discovered or be statically bound and then invoked, and can be choreographed into a composite service. This layer can map to the business service layer in Erl's enterprise logic (Erl, 2005). As a service exposure layer, it provides service descriptions and the mechanism to discover and invoke services.

The *business process choreography layer* defines the compositions and choreographies of services. This layer can map to the orchestration service layer in Erl's enterprise logic (Erl, 2005). In this layer, services can be bundled into a flow through orchestration or choreography, and a single application can be generated to finish a business task.

The *presentation layer* is also called the *access layer* in IBM's definition, which considers the access channel to a service and is usually outside the scope of SOA.

Among the four SOA-related layers, we can see that the operational systems layer and enterprise components layer concern the design and implementation of the services; the service layer concerns the description and discovery of the services; and

the business process choreography layer concerns the composition and orchestration of the services.

Service-oriented modeling provides modeling, analysis, design techniques, and includes activities to define the foundations of an SOA application. It defines the key elements in each of the SOA layers and makes critical architectural decisions at each level (Arsanjani, 2004). Researchers have studied different aspects of modeling. Some researchers focus their work on the modeling of services, which concerns the implementation of services; others focus on business process modeling or the workflow, which is in the business process choreography layer; and some consider both. We will discuss these methods later in Section 2.2.

In our research, modeling is a process to capture the business design and to create an encoded model as the solution for a specific business task. Therefore, our SOA modeling method concerns both the business logic layer and the orchestration service layer, which means that our SOA modeling method needs to capture the requirements of a business task and to create a process model invoking the existing and new services to implement the business task. The implementation and execution of the services in SOA are outside the scope of our study.

2.1.4 SOA Delivery Strategies

The lifecycle stages identified in Section 2.1.2 represent a simple process to build SOA applications. The success of SOA within an enterprise is generally dependent on the extent to which it is standardized when it is phased into the business and application domains. However, the success of a project delivering an SOA solution generally is measured by the extent to which the solution fulfills expected requirements within a given budget and timeline. For different projects, an

organization may have different priorities for the standardization extent and timeline. To satisfy different requirements of organizations, Erl (2005) has proposed three strategies which can also be useful for SOA modeling.

➤ **Top-Down strategy**

The Top-Down strategy is like an “analysis first” approach that not only requires business processes to become service-oriented, but also promotes the creation of an organization’s overall business model. Using this strategy, the business requirements should be collected and defined first. Then an enterprise-wide ontology will be defined to provide a common vocabulary. After that, service-oriented analysis and design will be conducted and the required services will be developed and deployed.

The Top-Down strategy can generate a high quality service architecture with well designed services, maximizing potential reusability and opportunities for streamlined compositions. However, with this strategy, organizations may be required to invest significantly in up-front analysis that can take a great deal of time without showing any immediate results.

In our research, the Top-Down strategy means that the developers will analyze the requirements first, after that a whole model for the target business approach will be created, and then this model will be further refined into sub business processes and service models.

➤ **Bottom-Up strategy**

The Bottom-Up strategy essentially encourages the creation of services as a means of fulfilling application-centric requirements. Web services are built on an “as needed” basis and modeled to encapsulate application logic to best serve the immediate requirements of the solution. Integration is the primary motivator for

Bottom-Up designs. Using this strategy, the business requirements should also be collected and defined first. Then the developers can analyze the required services, model them and develop them directly.

This approach supports quick realization of services. Although the Bottom-Up design allows for the efficient creation of web services as required by applications, this strategy may cause difficulties in later composition and orchestration because the services developed “as needed” may not fit each other well.

We enhanced the Bottom-Up strategy in our research. After the requirement analysis, the Bottom-Up strategy encourages searching and reusing the standardized services in the organization’s asset base. Those services should fulfill the requirements or partially satisfy the functions, and the services also have standardized interfaces for integration. They can be reused and integrated with new services to construct the final application.

The Top-Down strategy emphasizes creating a holistic model first, while the Bottom-Up strategy emphasizes preparing the component services first.

➤ **Agile strategy**

The challenge remains to find an acceptable balance between incorporating service-oriented design principles into business analysis environments and integrating web services technologies into a technical environment. For many organizations it is therefore useful to view the previous two strategies as extremes and to find a suitable middle ground. The Agile strategy allows for the business-level analysis to occur concurrently with service design and development. This strategy is also known as the meet-in-the-middle approach.

In our research, the Agile strategy means that the business-level analysis will be conducted concurrently with the service searching in the organization’s asset base.

The service searching can provide potentially reusable process models, and the business-level analysis can refine the results of the service searching during the modeling procedure.

We will present how these definitions and strategies can be realized in the SOA modeling methods in the next section.

2.2 Related Works of SOA Modeling

Modeling is the activity of developing a representation or simulation of a system as the basis for understanding, planning, developing or modifying the system. Software modeling is an essential part of the software development process that occurs prior to software implementation.

Software models are developed for representing the software requirements. Because the software models are generally used to translate software user's requirements into software developer's specification, the models are often not comprehensible for both sides and usually focus more on technical details.

In SOA, with the recognition of service reuse, two kinds of modeling appear: business process modeling and service modeling. Business process modeling mainly focuses on the business service layer and orchestration service layer, translating business requirements of the business logic layer into the business process models in the orchestration service layer and describing the functions and performance requirements of the services in the business service layer. Service modeling is the modeling for service implementation, which focuses on the application service layer and expresses the technology-specific functionality of services. Our study focuses on the business process modeling.

In the following sub-sections, we present a broad review of the research works relevant to business process modeling, service composition and orchestration, and model-driven design of web applications.

2.2.1 Standards for SOA Modeling

Organizations such as W3C (the World Wide Web Consortium), OMG (the Object Management Group) and OASIS (the Organization for the Advancement of Structured Information Standards) have proposed several standards for software modeling, such as UML (OMG, 2005) and WSBPEL (OASIS, 2007). Because many research works are based on these standards, we will introduce some of them first in this section.

2.2.1.1 Unified Modeling Language (UML)

The UML (OMG, 2005) is a standardized general-purpose modeling language. It provides a set of graphical notation techniques to construct abstract models for systems. It is widely used in the IT industry (Boggs et al., 2003).

The UML contains many different types of diagrams which can provide different perspectives of a system. For example, Use Case diagrams can represent the functional requirements of a system from the user's point of view and provide a functional requirements view for the system; Class diagrams and Composite Structure diagrams can represent the static structure of a system using objects, attributes, operations as well as relationships, and provide a static structural view for a system; Sequence diagrams, Activity diagrams and State Machine diagrams can show the collaborations among objects and the changes to the internal states of objects, and provide a dynamic behavior view for a system (OMG, 2005).

Based on UML and related techniques, researchers have developed different modeling methods for SOA modeling, which will be presented in detail in Section 2.2.2.

2.2.1.2 Business Process Modeling Standards

Traditionally, process-oriented modeling has been used for re-structuring business applications, integrating new processes and continuously monitoring system performance for years (Karagiannis et al., 1996). It is important in the context of Lean Management, Total Quality Management, Business Process Reengineering, Workflow Management and Supply Chain Management (Becker et al., 2000). This kind of modeling is also called business system modeling and organizational context modeling (Chen-Burger et al., 2005). Researchers have proposed several business process modeling methods, such as PSL (Process Specification Language) (ISO, 2004) and IDEF3 (Integration DEFinition Language) (Mayer et al., 1995).

PSL is a language and ontology for the specification of basic manufacturing, engineering and business processes. It was originally developed by the National Institute of Standards and Technology (NIST), and is now an international standard, ISO 18629. It can be used to define business processes and manufacturing engineering processes.

The IDEF3 Process Description Capture Method (Mayer et al., 1995) provides a mechanism for collecting and documenting processes. IDEF3 can be used to build structured descriptions to capture information about what a system actually does or will do, and can also provide different user views of the system. There are two IDEF3 description modes, process flow and object state transition network. The process flow description captures a description of a process and the network of

relations that exists between processes within the context of the overall scenario in which they occur. The object state transition network description summarizes the allowable transitions that an object may undergo throughout a particular process.

Besides these standards, some business process modeling notations have also been proposed in the last decade. BPMN (Business Process Modeling Notation) (OMG, 2008) provides a graphical notation for specifying business processes in a workflow. It was developed by Business Process Management Initiative (BPMI), and is currently maintained by the OMG. BPMN provides a standard notation that can be understood by all business stakeholders, who include business analysts, technical developers and business managers. Because BPMN cannot be handled by the computer, researchers have proposed methods for the mapping between BPMN and WSBPEL.

These business process modeling methods are able to formally express informally practiced business tasks, and the actions and effects of these processes can be demonstrated by using simulation techniques. However, these traditional business process modeling standards do not support SOA and they do not even define and use services. As a result, the research works based on these standards often only focus on the workflow (Yu et al., 2005).

After the appearance of service concept and SOA, researchers began to consider how to unite the business process modeling and SOA modeling, so that the reuse of services can accelerate the implementation of business processes. Although mappings between BPMN graphs and WSBPEL specifications have been proposed (Ouvans et al., 2006), we cannot say that the gaps between traditional business process modeling and SOA modeling have been bridged.

In SOA, a business process is generally implemented by services. There may be

hundreds of candidate services distributed on the net. With UDDI (OASIS, 2004) and SOAP (W3C, 2007), developers can discover and obtain the suitable services on the net. Then, the service composition standards can help them compose the services into an application. We will introduce these composition standards next.

2.2.1.3 Service Composition Standards

The crux of SOA development resides in combining several services into more complex, meaningful functions. There are several service coordination and orchestration standards, such as WSBPEL (Web Services Business Process Execution Language) (OASIS, 2007), WSCL (Web Services Conversation Language) (W3C, 2002a), WSCDL (Web Services Choreography Description Language) (W3C, 2005a) and WSCI (Web Service Choreography Interface) (W3C, 2002b).

WSBPEL was originally named BPEL4WS (Business Process Execution Language for Web Service) (Andrews et al., 2002), which is a specification proposed by IBM, Microsoft and other organizations. It is an OASIS standard now. WSBPEL (OASIS, 2007) provides a language to specify the workflows consisting exclusively of web services. It extends the web services interaction model and enables it to support business transactions. The processes in WSBPEL can be applied in one of two ways: abstract or executable. Abstract business processes are partially specified processes that are not intended to be executed, whereas executable business processes model actual behavior of a participant in a business interaction. An abstract process may hide some of the required concrete operational details.

The purpose of WSCL (W3C, 2002a) is to provide and define the minimal set of concepts necessary to specify conversations. It can be used to define the abstract interfaces of web services, such as the business level conversations or public

processes supported by a web service. WSCL conversation definitions are XML documents which can specify the XML documents being exchanged between web services, and the allowed sequencing of these document exchanges. It provides a formal specification language to separate the conversational logic from the application logic.

WSCDL (W3C, 2005a) is an XML-based language that describes peer-to-peer collaborations between the participating services. It can define the common and complementary observable behavior of the participants from a global viewpoint, and order the message exchanges to accomplish a common business goal. The WSCDL specifications can describe the interoperable, peer-to-peer collaborations between any type of participant regardless of the supporting platform or programming model used by the implementation of the hosting environment.

WSOI (W3C, 2002b) is an XML-based interface description language that describes the flow of messages exchanged between web services. It can work in conjunction with WSDL (W3C, 2001a), and describe the observable behavior of a web service. As it can also describe the collective message exchange among interacting web services, it can provide a global, message-oriented view of the interactions.

WSDL (W3C, 2001a) is a basic language for describing web services standardized by the W3C. A WSDL document contains a service type description, as well as a set of services conforming to this description. It provides the protocol bindings (e.g., SOAP (W3C, 2007)) and message formats (e.g., XML Schema (W3C, 2001c)) required to interact with the web services. The WSDL standard can describe web service properties and syntax; however, it does not directly support business process modeling.

These languages support a high-level specification of complex processes consisting of interactions between individual web services. They do not deal with the modeling of the implementation of services, or the automatic deployment of such implementations (Manolescu et al., 2005). These languages can provide formal specifications for service composition and application integration; however, they need to cooperate with other standards and technologies, such as WSDL and web service discovery technologies, to finish this work. Without good support for the information management of the business process models and services, the reusability of the business processes and services will be limited.

2.2.2 Methods for SOA Modeling

There are a number of related approaches regarding the SOA modeling. Some approaches use UML and related techniques. Gardner (2003) defined a UML profile to specify service orchestration with the goal to map the specification to WSBPEL code. Torres et al. (2005) proposed a model driven method for web application integration, in which they introduced mechanisms allowing developers to specify and integrate external services into an application. Wada et al. (2006) proposed an UML profile to model nonfunctional aspects of SOAs and presented a tool for generating skeleton code from these models. On the basis of UML 2.0, Zhou et al. (2008) provided a coding-free enablement framework to realize service modeling and service choreography. These approaches focused on different application areas of an SOA. They allowed separating conceptual concerns from implementation-specific concerns. However, they usually used a Top-Down strategy to analyze the system requirements, decomposed the system into subsystems and components, and represented these components with static graphs. It was difficult for

them to support the dynamic composition of services in an SOA.

Besides using UML to model services and processes, researchers also use service composition techniques to assist in the implementation of the business processes. With the web service composition standards and languages as presented in Section 2.2.1.3, new business processes, applications or solutions can be built in a relatively rapid and low-cost way through the composition of distributed services even in heterogeneous environments. In this way, UML models provide requirements analysis for the business processes, and web services composition techniques provide a practical foundation for business process management in loosely coupled distributed environments. However, gaps still remain between these two kinds of methods. The developers often cannot simultaneously grasp what is needed from the business world and what is available from the IT world. They usually need to adopt service discovery techniques to fix the gaps; however, to seamlessly use these methods together is also a challenge. To achieve smooth business collaboration, a new method that can unify the business processes and services is still needed.

There are many modeling approaches for business processes. We provide a brief overview of the main approaches here. For example, zur Muehlen et al. (2008) have developed a subset of BPMN, which is constructed for the process modeling in industry domain. Workflow patterns (van der Aalst et al., 2003) described concepts of workflow languages and Yu et al. (2005) proposed a framework to reuse workflow for business process modeling. Petri Net is another common technique for business process modeling. Wang et al. (2007) adopted Petri Net approach for collaborative business process modeling; Zdun et al. (2007) proposed a Petri Net-based pattern language for process-oriented integration of services. Besides Petri Net, Finite State Automaton has also been adopted to describe business processes (Mahleko et al.,

2006). These methods are typical traditional business process modeling methods with additional consideration of services. These methods can help to increase the understandability and reusability of business process models; however, they do not support reuse of services. They can only provide modeling from the business viewpoint.

In recent years, ontology has been recognized as a useful technique to provide precise descriptions of the objects in process modeling. Ontologies can be used for knowledge sharing and reuse. In business process modeling domain, ontologies can provide formal descriptions of the models and support the model information sharing and reusing. Different ontology languages have been developed, such as RDF (Resource Description Framework) (W3C, 2004b), DAML+OIL (DARPA Agent Markup Language and Ontology Interchange Language) (W3C, 2001b) and OWL (Web Ontology Language) (W3C, 2004c). Researchers have adopted these languages as the basic languages to construct their business process modeling languages or frameworks, as presented next.

Both *domain ontologies* and *upper ontologies* have been proposed in recent years. *Domain ontology* can capture the knowledge valid for a particular type of domain (Niles et al., 2001). It can describe the domain specific terms and knowledge formally and precisely and can provide a knowledge base for the application modeling. The defined domain ontologies for SOA modeling usually focus on the e-business domain (Liu et al., 2007; Lim et al., 2007; and Osterwalder, 2002) and healthcare domain (Kuziemy et al., 2003). Researchers have also adopted ontology techniques in the automotive domain (Blomqvista et al., 2008; Angele et al., 2008). Blomqvista et al. (2008) proposed a manual method and an automatic method for ontology construction, and applied them in developing a domain ontology for a

company in the automotive supply industry. Angele et al. (2008) proposed a method for ontology construction and created an ontology to represent and share the knowledge for testing of cars. Although domain ontologies can provide unified concepts and domain knowledge sharing for the software development, they cannot directly provide specification and service reuse.

Upper ontology describes general concepts that are the same across several domains (Niles et al., 2001). In business process modeling, it can capture the general concepts and relationships in a model. OWL-P (Mallya et al., 2005, Desai et al., 2005) is an upper ontology based on OWL. It captures meaningful interactions among different roles as protocols and uses the protocols to create concrete business processes. Researchers have also proposed task-based process ontologies: Therani (2007) proposed a task-based process ontology for capturing process knowledge for information systems development at different levels of abstraction; Tran et al. (2007) also proposed an upper ontology OWL-T, which can be used to express business processes at a high-level abstraction. Both task-based ontologies provided a framework to represent attributes of business processes, the former focused more on information representation, using ontology to formally and precisely show the parameters and conditions of tasks; the latter focused more on process modeling, using ontology to represent the composition of tasks. However, to use OWL-T in modeling, developers need to work with four ontologies representing “Domain”, “Task”, “Process” and “Service” respectively, which are difficult to use and may cause inconsistency between the ontologies.

In SOA development, ontologies are also used for service modeling and service composition. OWL-S (W3C, 2004e) provides a core set of markup language for describing the properties and capabilities of web services. To describe a service,

three concepts, “ServiceProfile”, “ServiceModel” and “ServiceGrounding”, are defined in OWL-S. “ServiceProfile” is for service advertising and discovery on the net. “ServiceModel” exposes how a service is composed. “ServiceGrounding” specifies the details of how to access the service. OWL-S concerns the automatic discovery and composition of web services. Besides OWL-S, several ontologies representing the workflow models of services have been proposed, such as SOF (Fang et al., 2007) and m3po (Haller et al., 2006). SOF (Slight Ontology Framework) can be used for performing semi-automatic change management for software development; while m3po (multi meta-model process ontology) is an intermediate unifying workflow ontology based on workflow reference models, such as WSBPEL and PSL. These ontologies provide meta-models for the description of the internal workflow of (web) services; however, the granularity of the modeling components may not be suitable for the modeling of external workflow of enterprise business processes.

Table 1 compares the above modeling methodologies. From Table 1, we can find that these methods are based on different standards and focus on different aspects of SOA development. Some of the methods are developed from IT viewpoint and deal with the service composition problems; other methods are from business viewpoint and provide workflow design of business processes. There are also some ontology-based methods providing meta-model for service orchestration and process description. Although it might be possible to combine some of these methods to provide better support to SOA development, however, gaps still remain between these methods. Difficulty still exists for the developers to map their business needs to the available services in the IT world.

Table 1. Comparison of SOA modeling methods

	UML-based Methods	Service composition Methods	Traditional business process modeling methods	Ontology-based methods
Fundamental Standards	UML	WSBPEL, WSCL, WSCDL, WSCI	PSL, IDEF3, BPMN	OWL, OWL-P, OWL-T, OWL-S
Modeling Focus	Separating conceptual concerns from implementation specific concerns	Service composition and orchestration	Workflow design of business processes	Providing meta model for business process modeling and service modeling
Example Methods	UML profiles specific for service orchestration	WSBPEL-based service orchestration methods	BPMN-based methods; workflow languages; Petri-net methods	Domain ontologies and upper ontologies such as OWL-P
Limitation	Providing only static graphs	Lacking a complete view of the business	Lacking the concern of services	Developed separately, difficult to use together

To overcome the limitations and problems of previous SOA modeling approaches, and to support the reuse of both models and services, we design a core ontology BPO (Business Process Ontology) for Business Process modeling to describe business processes and their relevant services. On the basis of the core ontology BPO, a knowledge base can be constructed to present and manage the information of SOA modeling. These ontologies can provide sharable and precise description for the models and services and support the knowledge management of the process model and service asset bases. With these ontologies, our modeling methods can provide not only the application reuse (by reusing services), but also the specification reuse (by reusing the concrete process models or the process model templates).

2.2.3 SOA Modeling and Developing Frameworks

As introduced in Section 2.1.2, the general lifecycle of SOA applications includes four phases (High et al., 2005): Model, Assemble, Deploy and Manage, which are layered on a backdrop of a set of governance and processes to ensure the compliance.

An MDA-based framework has been proposed for SOA development (Stojanovic et al., 2004; Karhunen, 2005). MDA (Model Driven Architecture) (OMG, 2003) stresses the importance of the platform-independent models (PIMs) and platform-specific models (PSMs). Using MDA-based methodologies, organizations can separate abstract business logics from the concrete implementation environment, and capture the business design based on the business processes. Although there are several different frameworks for model driven service-oriented architecture, their modeling and developing flow generally follows the outline of MDA shown in Figure 3. The application development naturally starts with the business domain requirement descriptions. From the analysis of business domain, developers can obtain models on a strategic level, which are platform-independent models. In SOA modeling, this can be considered as business process modeling. After that, developers can create platform-specific models to implement the business process models, which can be considered as service modeling and service orchestration. Organizations can construct their own Service/Business process repository to support the modeling, or for some cases the Net can be the repository, which means the developers can discover services on the Net. When all the services are available, a final application can be constructed by assembling or integrating these services together. PIM and PSM are connected by an Up-Down arrow and the same

connection appears between PSM and application, which means that the developers can reconstruct the models and applications when the requirement changes.

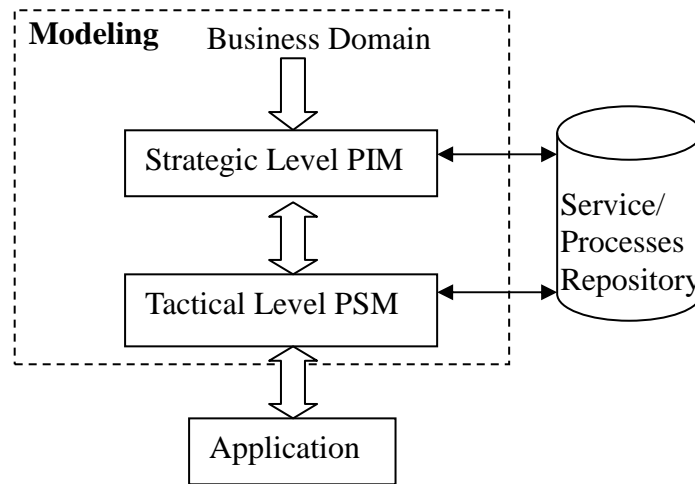


Figure 3. Development flow of Model Driven Service-Oriented Architecture

Using MDA for SOA modeling can accelerate SOA's formalization and automation. MDA allows separating conceptual concerns from implementation-specific concerns. Although MDA can provide static models for the implementation, it cannot represent run-time relevant attributes (Tetlow et al., 2006). Therefore, MDA-based modeling and developing techniques may not fully support a flexible and dynamic SOA development.

Besides MDA-based framework, some ontology-based framework has also been proposed for SOA development (Tran et al., 2007). With the support of a knowledge base, ontologies can be used at several stages of the software lifecycle. Tran et al. (2007) have proposed a system framework, as shown in Figure 4, to illustrate how different ontologies can be used in SOA application development. In this framework, the domain ontologies provide essential concepts for the business domains and can be referred to at different stages of the lifecycle. The task ontologies can help the analysis of business tasks and map the task template to the

process template. The business process ontology written in OWL-T (Tran et al., 2007) can help to define business processes. The service ontology written in OWL-S (W3C, 2004e) or WSDL-S (W3C, 2005b) can help the service discovery and selection. The framework introduces ontologies to the modeling and developing procedure of software, which can provide formal and precise descriptions of business information, process models and services. In this framework, it is assumed that the concepts and relations in the ontologies have been created. These concepts are retrieved from the related ontology knowledge base and customized into working products at each step of the development. However, because of the separation of the maintenance of the ontologies and the software development, there may be a gap between the information in the knowledge base and the real world, which generates inconsistency.

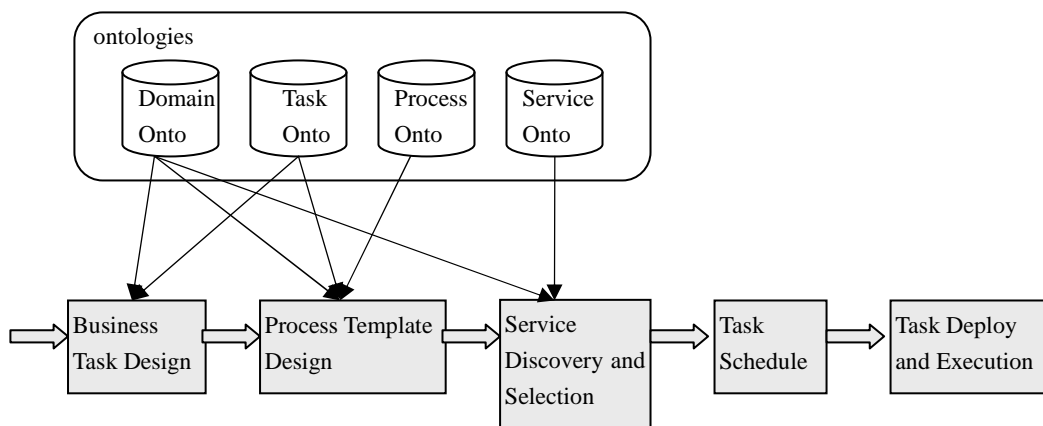


Figure 4. Steps of ontology based software development

In this research, we propose an Ontology-based Business Process Modeling and Developing Framework (OBPMDF), which focuses on illustrating the modeling procedure of SOA application development. Our framework can be used with the core ontology BPO and its extension, which together provide consistent information for business processes, services and their mappings.

2.3 Quality Attributes for Models

The importance of software product quality has been widely recognized by most organizations and stakeholders. SOA modeling is the first stage of SOA development and the quality of SOA models will directly impact on the quality of the final systems. Given an inferior model, the risk of misunderstanding and obtaining a low quality product would increase. To reduce these risks, models should be verified and tested. This section presents the quality attributes proposed by different researchers. Based on these attributes, a set of quality attributes for SOA models will be proposed and analyzed in Section 4.3.

2.3.1 Attributes of Engineering Models

Selic (2003b) has proposed five attributes of engineering models for building complex systems, which are listed below:

- **Abstraction:** A model is always a reduced rendering of the system that it represents. By removing or hiding detail that is irrelevant for a given viewpoint, it lets us understand the essence more easily.
- **Understandability:** Understandability is a direct function of the expressiveness of the modeling form used (expressiveness is the capacity to convey a complex idea with little direct information). A good model provides a shortcut by reducing the amount of intellectual effort required for understanding.
- **Accuracy:** A model must provide a true-to-life representation of the modeled system's features of interest.
- **Predictive:** Using a model, an organization can correctly predict the

modeled system's interesting but non-obvious properties, either through experimentation (such as by executing a model on a computer) or through some sort of formal analysis, which depends greatly on the model's accuracy and modeling form.

- **Inexpensive:** A model must be significantly cheaper to construct and analyze than the modeled system.

These five attributes are high level requirements of an engineering model. More specific requirements are needed for software models.

2.3.2 Quality Attributes Defined in ISO 9126

ISO 9126 standard specifies three kinds of software products quality attributes: internal quality, external quality and quality for use. Internal quality can be used to measure not only the quality of final products but also the quality of interim products.

The six attributes of internal quality include (ISO, 2001):

- **Functionality:** It ensures that the software product can provide functions which meet stated and implied needs when the software is used under specified conditions.
- **Reliability:** It ensures that the software product can maintain a specified level of performance when used under specified conditions.
- **Usability:** It ensures that the software product can be understood, learned, used and is attractive to the user.
- **Efficiency:** It ensures that the software product can provide appropriate performance, relative to the amount of resources used.
- **Maintainability:** It ensures that the software product can be modified.
- **Portability:** It ensures that the software product can be transferred from

one environment to another.

These attributes and their sub-attributes are associated with the final system and the interim products of different lifecycle stages. Some of them are applicable to the SOA model. Table 2 shows the quality attributes and sub-attributes in ISO 9126.

Table 2. Quality attributes and sub-attributes in ISO 9126

Attributes	Sub-Attributes	Description
Functionality	Accuracy	To provide the agreed results with the needed degree of precision
	Compliance	To adhere to standards, conventions or regulations in laws and similar prescriptions related to functionality
	Suitability	To provide an appropriate set of functions for specified tasks and user objectives.
	Interoperability	To interact with one or more specified systems
	Security	To protect data from unauthorized persons
Reliability	Fault Tolerance	To maintain a specified level of performance in cases of software faults or of infringement of its specified interface
	Recoverability	To re-establish a specified level of performance and recover the data directly affected in the case of a failure
	Maturity	To avoid failure as a result of faults in the software
Usability	Understandability	To enable the user to understand whether the software is suitable, and how it can be used
	Learn-ability	To enable the user to learn the application of the software
	Operability	To enable the user to operate and control the software
Efficiency	Resource Utilization	To use appropriate amounts and types of resources when the software performs its function under stated conditions
	Time Behavior	To provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
Maintainability	Stability	To avoid unexpected effects from modifications of the software
	Changeability	To enable a specified modification to be implemented
	Testability	To be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified
Portability	Install-ability	To be installed in a specified environment
	Adaptability	To be adapted for different specified environments without applying actions or means other than those provided for

		this purpose for the software considered
	Compliance	To adhere to standards or conventions related to portability
	Replaceability	To be used in place of another specified software product for the same purpose in the same environment
	Co-existence	To co-exist with other independent software in a common environment sharing common resources

2.3.3 Quality Attributes for UML-based Models

Lange et al. (2005) have proposed a quality model specific for UML-based development. Their model identifies the following attributes for UML models:

- **Complexity:** It measures the effort required to understand a model.
- **Traceability:** It ensures that the relations between design decisions are explicitly described.
- **Completeness:** It ensures that the model's functionality covers all the requirements.
- **Consistency:** It ensures that there is no conflicting information.
- **Self-Descriptiveness:** It ensures that the model contains enough information for a reader to determine its objectives, assumptions, constraints, inputs, outputs, components, and status.
- **Detailedness:** It ensures that the model describes relevant details of the system.
- **Balance:** It ensures that the model satisfies the other nine attributes at an equal level.
- **Conciseness:** It ensures that the system is described to the point and there is no unnecessarily detail.
- **Esthetics:** It ensures that the graphical layout of models enables ease of

understanding of the described system.

- **Correspondence:** It ensures that the system elements, their relations and design decisions are the same between the model and the system.

Some of the above ten attributes can be mapped to the quality attributes of ISO 9126. The *Complexity* and the *Esthetics* attributes can be considered as different views of *Understandability*, as they both consider the understandability of a model. The *Detailedness* and *Conciseness* attributes can be considered as different aspects of *Suitability*. The *Traceability* attribute considers the relationship between different versions of design. As the *Correspondence* attribute is about the consistency of the model and its system, this attribute can be ignored at the modeling stage.

2.3.4 Guidelines to Improve Quality of Information

Models

Becker et al. (2000) have proposed general guidelines of modeling to improve the quality of information models, which include six quality attributes:

- **Correctness:** The models should be consistent and complete in both the syntactic and the semantic facets.
- **Relevance:** The model developers should select a relevant object system, use a relevant modeling technique, and develop a relevant (minimal) model system.
- **Economic Efficiency:** This attribute focuses on improving efficiency by using reference models and appropriate modeling tools, and reusing models.
- **Clarity:** The model can be understood by model users.

- **Comparability:** The models should be interoperable and comparable so that they can be used in cross-company projects.
- **Systematic Design:** The relationship between different views of information models (such as the process model and data model) should be well-defined.

Some of the above attributes are similar to the quality attributes discussed earlier. *Correctness* considers the completeness and consistency of models. *Clarity* is related to the readability and understandability of models. *Systematic Design* considers the relationship between processes and resources. If the resource utilization of a process model can be predicted, the model should have been systematically designed.

2.4 Related Works of Automotive Software

Modeling

As we plan to apply our SOA modeling methods to the automotive software domain for the validation of our proposed methodology, we first present some background of automotive software in this section. Related works of automotive software modeling will also be covered.

2.4.1 Overview of Automotive Software

Automobiles nowadays incorporate substantial amounts of software (Dannenberg et al., 2004; Leen et al., 2002). From the viewpoint of vehicle builders (Original Equipment Manufacturers or OEMs), this increasingly complex software is unlike the traditional parts of mechanical vehicles in that it cannot be simply

assembled but rather must be integrated as a subsystem within a system (Broy et al., 2007b; Emaus, 2005). There are a number of difficulties associated with this. First, if the heterogeneous subsystems sourced from different suppliers do not fit together smoothly, the integration may fail (Broy et al., 2007b). Second, there is a lack of precise specifications and guidelines for architecture development (Broy et al., 2007b; Weber et al., 2003). Besides these difficulties, there are process issues such as abstract requirement specifications, specification reuse and software reuse (Grimm, 2003; Weber et al., 2003). For example, even though 90% of the functions in two sequential generations of a car are often the same (Broy et al., 2007b), there is limited software reuse. This particular problem can be attributed to the close coupling of software and hardware in traditional embedded systems (AUTOSAR, 2006a; Broy et al., 2007b). Clearly, if the interfaces of hardware and software could be standardized, the coupling of hardware and software can be loosened. Then, it would be possible to use more general hardware structures and implement application-level reuse, allowing a more economical automotive software development.

A number of software development standards have been developed for the automotive industry, such as AUTOSAR (AUTomotive Open System ARchitecture) (2006a) and AMI-C (Automotive Multimedia Interface Collaboration) (2003). AUTOSAR (2006a) provides a de-facto open industry standard for electric and electronic automotive architectures. Its particular benefit is that it facilitates the separation of applications from infrastructure by adopting a “Virtual Functional Bus” (VFB), which provides applications with standardized communication mechanisms. The contribution of AMI-C (2003) has been to produce a series of technical specifications for vehicle communication networks so as to promote the

standardization of common automotive information and entertainment system interfaces serving mobile phones, PDAs, navigation, etc. Standards such as these make it possible to create a common service layer in automobiles, and they also allow the reuse of common services on different hardware components without requiring changes in the rest of the system. This can avoid software duplications and redundant hardware.

Several kinds of automotive software development modeling methods and tools have been proposed and applied. MATLAB®, Simulink® and Stateflow® are a series of commercial modeling tools (MATLAB, 2008; Simulink, 2008; and Stateflow, 2008). They provide graphical design functions for modeling and simulating continuous and discrete event-based behavior of a dynamic system. These tools do not however address every aspect of automotive systems development; for example, developers still need UML models or other discrete data flow / state machine models to model business information processing in vehicles (Broy et al., 2007b). To unite these separating engineering cultures, UML-based modeling methods have been developed for automotive systems. Examples of these include AML (Automotive Modeling Language) (von der Beeck et al., 2003) and SysML (Systems Modeling Language) (Rao et al., 2006; OMG, 2007), which tailor the standard UML and define new notations to satisfy the specific features of real-time and embedded systems. Yet another example is MSC-based (Message Sequence Charts) and its related tools (Krüger et al., 2004), with which developers can use the MSC graph to capture system requirements and specify the communications within the system. Although these methods can satisfy the general requirements for constructing real-time system models, only the MSC-based method recognizes the importance of the reuse of common services and supports the modeling and

configuration of services. Other methods, in contrast, focus on component-based modeling, which may limit the reuse of functions and cause problems for heterogeneous system integration (Krüger et al., 2004). Furthermore, none of the methods supports the developers reuse their existing business process specifications.

A further difficulty in automotive software development has been the lack of relevant ontologies that can be directly adapted to business process modeling. As discussed in Section 2.2, there exists domain ontologies and upper ontologies for business process modeling or service composition for specific domains, such as e-business (Osterwalder, 2002; Lim et al., 2007; and Liu et al. 2007) and healthcare (Kuziemyk et al., 2003), and there are upper ontologies that focus on resolving modeling problems such as the description of internal workflow of services (W3C, 2004e; Haller et al., 2006; and Fang et al., 2007) and the message heterogeneities between services (Gouvas et al., 2007). More particularly, some researchers have proposed methods (Angele et al., 2008; Blomqvista et al., 2008) which OEM developers and suppliers can use to construct automotive domain ontologies to provide terminology definitions and sharable knowledge for the entire enterprise. This can help by providing a knowledge base for software development. Nonetheless, there still remains a need for modeling methods to maximize the usage of the knowledge base.

In this thesis we propose to use SOA modeling for automotive software development so that the reuse of existing automotive specifications and services can be improved during the modeling stage. The advantage of applying SOA is that it provides a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains (OASIS, 2006) and therefore is suitable for solving problems associated with the integration of heterogeneous

systems, in particular allowing the sharing of a knowledge base of process models and services.

2.4.2 Features of Automotive Software

Automotive software systems are complex distributed real-time and embedded systems. The modeling of such systems must consider not only the functional requirements such as system operations but also specific non-functional features inherent in the operation of distributed and real-time systems, in particular the following four features:

➤ **Heterogeneity**

Heterogeneity is an inherent attribute of a distributed system (Broy, 2006). Formerly, automotive software operated in stand-alone embedded systems. Nowadays such software operates in complex distributed systems composed of subsystems deployed by different suppliers (Broy, 2005). Such subsystems are often heterogeneous, and developers from OEMs need to integrate the heterogeneous software into a whole system.

➤ **Interactivity**

Today's automotive software can support many different kinds of processes, from chassis to infotainment functions. The software is distributed in the automobile and is closely related and interdependent. There is frequent communication between different subsystems and between the machine and the external world (such as the Internet or human beings). This interaction may be supported with buses or by wireless technology (Broy, 2005).

➤ **Timeliness**

Timeliness is a basic feature of real-time systems (Gerard et al., 2003; Selic,

2003a; Krüger et al., 2004; and Tsai et al., 2006). Timing properties, such as minimum and maximum response times and deadlines, are critical and may influence the safety of the automobile.

➤ **Concurrency**

Many features in distributed and real-time systems are inherently concurrent; therefore concurrency specification is an important issue in automotive software modeling. This feature should be considered in the initial phase of software development, not only because the real world is fundamentally concurrent (Gerard et al., 2003), but also because the conflict of resources caused by concurrency should be detected as early as possible.

These four non-functional features make the modeling of automotive software systems more complex than general business software modeling. In particular, it is not possible to directly apply traditional modeling methods because the representation of these features may require new notations.

The reusability of automotive software is another important consideration during modeling. As noted earlier, over two sequential generations of a car, 90% of the functions will typically remain unchanged (Broy et al., 2007a and 2007b). Moreover, most of the few changes between two consecutive generations of a car are not evolutions but simple updates not affecting the structure of the system. Clearly, there is an opportunity here to avoid or reduce the redevelopment of functions by identifying the reusable services in the modeling stage and reusing them in the new system. Reusing the systematic requirements and using system model templates can offer significant gains in developmental productivity and in the quality of the resulting software product (Cybulski et al., 2000; Lam et al., 1997). While the reuse of model specifications is still not common today (Weber et al., 2003), the benefits

of reusing existing services and business process specifications are clear:

- Reduction of costs associated with re-development.
- Reduction of errors, resulting in safer systems. One of the benefits of reuse is that existing services have been tested by the developing team, by the organization, and even by users.
- Provision of clear specifications for software assembling or integration. The fact that the configuration and performance of the services are known in the modeling stage means that it is possible both to obtain a clearer integration specification and to make an earlier identification of potential problems in service assembly or integration.
- Improvement in the accuracy of the model simulation. Provided with performance information of the component services, designers can produce a more realistic simulation of the model.

2.4.3 Automotive Software Modeling Methods

2.4.3.1 Traditional Automotive Software Modeling Methods

Three different kinds of automotive software modeling methods have been proposed in the last decade, MATLAB®/Simulink®/Stateflow® (MATLAB, 2008; Simulink, 2008; and Stateflow, 2008), UML-based methods (von der Beeck et al., 2003; OMG, 2007; Rao et al., 2006; Gerard et al., 2003; Selic, 2003a; and Karsai, 2004) and MSC-based methods (Broy, 2006; Harel et al., 2003).

The MATLAB®/Simulink®/Stateflow® (MATLAB, 2008; Simulink, 2008; and Stateflow, 2008) methods have been widely used by both OEMs and suppliers. They are control theory-based modeling tools and even today automotive software

developers continue to require an understanding of control theory because automotive software systems are connected to sensors and actuators, and must respond according to classical control theory (Broy et al., 2007b). The MATLAB®/Simulink®/Stateflow® are a series of graphical design tools for the modeling, simulation and development of system/software components. MATLAB® provides a high-level technical computing language, Simulink® provides an interactive graphical environment for the modeling and simulation of dynamic and embedded systems, and Stateflow® extends Simulink® with a design environment for state machines and flow charts development and can describe more complex logic (MATLAB, 2008; Simulink, 2008; and Stateflow, 2008). This set of tools is able to satisfy the requirements in event-based systems. However, when the task is business information processing in vehicles, UML-like model is better for representing the discrete models of data processing (Broy et al., 2007b).

2.4.3.2 UML-based Automotive Software Modeling Methods

The original use of UML was as a tool for object-oriented analysis and design; Section 2.2 presented the researches on using UML to SOA development; here we will review its usage in real-time and embedded system modeling. A standard UML profile for real-time based applications has been proposed and can be used for modeling parallelism, behavior, and communication (Gerard et al., 2003). UML has also been adopted as a meta-language for defining specific domain modeling languages (Neema et al., 2004). These languages have also been proposed as the basis of MIC (Model-Integrated Computing), a model-integrated approach to embedded software development (Neema et al., 2004; Karsai et al., 2003). Proposed improvements to UML include AML (Automotive Modeling Language) (von der

Beeck et al., 2003) and SysML (Systems Modeling Language) (OMG, 2007). AML adopts a subset of UML to represent automotive systems and defines a meta-model to represent modeling concepts such as functions, ports, and connectors (von der Beeck et al., 2003). SysML is also based on a subset of UML, and improves and extends some diagrams, providing a graphical modeling language for both hardware and software systems. SysML was proposed as an open source specification for system modeling in 2003, and has now been issued by OMG as an “available specification”. A driver information system has also now been developed based on SysML (Rao et al., 2006).

Both the MATLAB® family tools and the UML-based methods can be used to model real-time, embedded systems but they support modeling at different levels of abstraction: UML-based methods can support higher levels modeling such as the structure of systems and the composition of components, and MATLAB® family tools can support lower levels modeling such as the behavior of a component and signal processing. UML provides formal notations and diagrams to represent models; Simulink® also provides a block library for modeling components, for example, “inport” identifies the links from outside a system entering into the system and “outport” identifies the links from a system exiting to an outside destination (MATLAB, 2008; Simulink, 2008; and Stateflow, 2008).

Although the two types of methods are based on different theories, (the MATLAB® family is based on control theory and UML-based methods are object-oriented), both support hierarchical modeling. They usually use a Top-Down strategy to analyze the system requirements and decompose the system into subsystems and components. Although these methods and tools can accelerate the modeling and developing process, they do have some limitations. For example,

because of the system's heterogeneity and the lack of standard communication interfaces, re-development of similar functions cannot be avoided, and the portability and reusability of the functions are limited.

2.4.3.3 Service-based Methodology in Automotive Domain

The appearance of standard AUTOSAR (AUTomotive Open System ARchitecture) (2006a) also makes it possible to employ a service-based methodology in automotive software development. The AUTOSAR development partnership was founded in 2003 by major OEMs and Tier 1 suppliers (software suppliers). AUTOSAR aims at facilitating the reuse of software and hardware components between different vehicle platforms, OEMs and suppliers (AUTOSAR, 2006a; Fennel et al., 2006). It uses a layered architecture to separate AUTOSAR software (applications) and basic software (related to hardware), as shown in Figure 5. The software architecture is structured into five layers, besides the Complex Drivers layer which cannot be mapped into a single layer. The AUTOSAR Application Layer consists of AUTOSAR software components, which are "Atomic Software Components" that can provide reusable pieces of functionality of the application and each AUTOSAR software component can only be conducted on one AUTOSAR ECU (Electronic Control Unit). The other four layers enable the AUTOSAR software components to be independent from the specific hardware/ECUs. Runtime Environment (RTE) is also a hardware independent layer and can provide communication service for the AUTOSAR software components in the Application Layer.

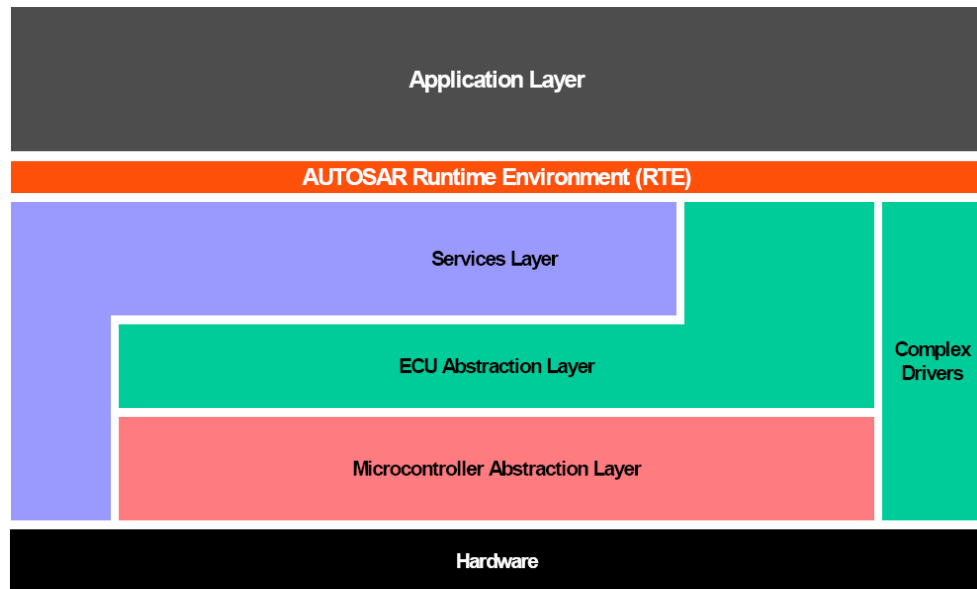


Figure 5. Layered architecture of AUTOSAR (2006a)

Besides using this layered structure to separate the applications and hardware, AUTOSAR provides standardized APIs (Application Programming Interfaces) for the integration of the AUTOSAR software components. This can help the manufacturers and suppliers to freely exchange hardware and software components.

AUTOSAR has also been the basis of a proposal to reuse common services that uses an MSC-based method and related tools for service-oriented automotive software (Broy, 2006; Krüger et al., 2004). Using this method, developers can define the target services and map these services to a specific component configuration. MSCs (ITU-T, 2004; Harel et al., 2003) are adopted to capture service requirements, especially communication between the services (Krüger et al., 2004). Validation, simulation, and executable specifications can be generated after the modeling. Similar to the above two kinds of methods, the MSC-based method also applies the Top-Down modeling strategy. Researchers have adopted the “service” concept in their methodology and it has been shown that service-based software development can improve the reusability of software (Krüger et al., 2004). However, how the

services can be discovered in the organization’s asset base is an open problem.

Comparing these three kinds of automotive software modeling methods (Traditional modeling methods, UML-based modeling methods and service-based modeling methods), we can obtain the following table:

Table 3. Comparison of automotive software modeling methods

	Traditional methods	UML-based methods	Service-based Method
Fundamental Theory	Control theory	Object-Oriented	Service-Oriented
Modeling Focus	Structural modeling; Behavior modeling; Event-based modeling	Component modeling	Service communication modeling
Unit of Modeling	Component	Component	Service
Reusable Unit	Component	Component	Service
Hardware-Coherent	Yes	Yes	No

From Table 3, we can see that the modeling methods are based on different fundamental theories and deal with different modeling issues. Although these methods support both the structural modeling for a new system and the detail modeling for new components, none of them identifies how to reuse the existing models and services during the modeling stage. Thus, to complement these methods, our ontology-based SOA modeling methodology can be used. Our method can treat the AUTOSAR software components as services, and focuses on business process modeling for the automotive software development. Supported by ontologies, our method can provide formal descriptions of the models and services and improve the reusability of existing models and services in the construction of new models.

We are not the first one to apply an ontology-based approach. An RTSOA (Real Time SOA) framework (Tsai et al., 2006) has been proposed which uses ontologies to represent the concepts and relations in real-time SOA application development.

Different ontologies are used to describe different aspects of real-time SOA software, and the software construction can be represented by cross-referencing between the ontologies. However, more investigation is needed to integrate those ontologies within an SOA development platform and maintain the consistency of the ontologies. Recently, a Semantically-enabled Service-Oriented Device Architecture (SeSODA) (Gouvas et al., 2007) was proposed to extend Service-Oriented Device Architecture (SODA) (de Deugd et al., 2006). SODA provides a way of adapting SOA to devices (sensors and actuators) and lets programmers deal with these devices just as business services. SeSODA introduces semantics into the SODA environment, using ontologies to resolve data and message heterogeneities between services in runtime. These methods can partially resolve the problems of service integration; however, they cannot improve service reuse.

Comparing with these ontologies, our core ontology BPO can solve the problem of reuse of both models and services. On the basis of the core ontology BPO, a knowledge base AutoPO (Automotive Process Ontology) is constructed to present and manage information associated with automotive software modeling. These ontologies can provide precise, sharable descriptions for the models and services and support the knowledge management of the process model and service asset base. Using these ontologies, our modeling methods can provide not only the application reuse (by reusing services) but also the specification reuse (by reusing the concrete process models or the process model templates).

2.4.3.4 Event-Driven Architecture and SOA

Automotive software, being an embedded system, also suits to be represented by event-driven architecture. Event-driven architecture (EDA)¹ is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. Service-oriented architecture (SOA) provides methods for systems development and integration where systems package functionality as interoperable services. SOA can be used to design composite applications and implement workflow. The service invocation can be driven by user requests or by events (Michelson, 2006). With the development of SOA, researchers have been investigating the interaction of SOA and EDA.

Laliwala et al. (2008) proposed an Event-driven SOA (EDSOA) as an extension of SOA to model the event-driven process chains and to achieve event-driven automation of business process. In the EDSOA, an event manager is used to control the components of the system. The Event Manager is designed to communicate dynamically with the distributed heterogeneous services and other components. Komoda (2006) proposed a SOA framework for industrial systems. This framework uses a layered architecture to separate the hardware, controller and management system. It can support real-time and embedded system. Besides these research works, industry organizations (such as Oracle) also propose Event-Driven SOA, which is also called SOA 2.0². In SOA 2.0, Event-driven architecture can complement service-oriented architecture by activating services with incoming events (Hanson 2005).

¹ http://en.wikipedia.org/wiki/Event_Driven_Architecture

² http://en.wikipedia.org/wiki/Event-driven_SOA

2.5 Summary

In this chapter, we introduced the SOA-related concepts, such as SOA development lifecycle, layers of SOA structure and SOA delivery strategies. On the basis of this introduction, we explained the importance of SOA modeling and different modeling strategies.

We also presented different kinds of related works for SOA modeling, such as modeling standards, modeling methods and frameworks, and analyzed the benefits and limitations of these methods. As we will validate our modeling methodology by applying it to the automotive software development domain, which will be presented in Chapter 4, this chapter also reviewed the related works for the automotive software modeling. For the verification of SOA models, we reviewed quality attributes for models proposed by different researchers. This would help to identify quality attributes suitable for SOA models.

Chapter 3 Ontology-based SOA Modeling

Methodology

In this study, we adopt knowledge management techniques to accelerate the modeling process and improve the accuracy and reusability of software models. This chapter will describe the core of our methodology.

We will first define ontology. After that we will present a core ontology for Business Process modeling, called BPO (Business Process Ontology) (Liao et al., 2007a), which can define business processes formally, provide information of existing process models and services in an organization's asset base, and record the mapping between the processes and services. BPO can be applied to different domains. A knowledge base AutoPO is then constructed in Chapter 4 by extending the core ontology BPO to the automotive business process domain.

An Ontology-based Business Process Modeling and Developing Framework (OBPMDF) is also proposed in this chapter to illustrate the main processes and products in the Model and Assemble phases of SOA development lifecycle. The framework applies the extension of BPO to work as the knowledge base, which can give accurate definitions of business processes, services, and their relationships.

Based on BPO and OBPMDF, we will present relevant modeling methods at last, which support the three strategies for SOA modeling (Top-Down, Bottom-Up and Agile) as well as satisfy the reuse of model specifications and services.

3.1 Ontology Definitions

Originally derived from philosophy, in modern computer science the concept of ontology is defined as a formal, explicit specification of a shared conceptualization (Gruber, 1995). Generally, ontologies can be used for knowledge sharing and reuse, and, according to Uschold et al. (1996), are in particular useful in three areas: communication, inter-operability and systems engineering. With regard to communication, adopting a shared ontology allows all participants to use a standardized terminology for all objects and relations in their domains. Ontologies improve inter-operability because they can be used to support translation between different languages and representations. Furthermore ontologies support the design and development of the software systems by providing declarative specifications, which improves reliability and reusability. In this study, we adopt ontology to help in the construction of business process models, which is an important phase of system engineering.

From the different ontology languages, such as RDF (W3C, 2004b), RDFS (RDF Schema) (W3C, 2004d), DAML+OIL (W3C, 2001b), OWL (W3C, 2004c), we choose OWL to describe our ontologies because it is a machine readable language for defining ontologies, and provides higher machine readability than RDF and RDFS, and has more vocabulary than RDF and DAML+OIL (Salam et al., 2007). The main elements of OWL are classes, individuals and properties. Classes represent the abstraction of objects and can be considered as a set containing elements. Individuals are instances of a class or can be considered as members of a class. Properties represent the relations between individuals (which are called object properties) or from individuals to data values (which are called data type properties).

A core ontology is one of the key building blocks necessary to enable the scalable assimilation of information from diverse sources (Hunter, 2003). In this study, a core ontology is considered as an initial domain ontology, a kernel that can be extended by adding Axioms, Lexicon and Instances. For better understanding of the mathematical definitions for these concepts, we first list the notations which will be used in those definitions.

- Italic uppercase letters, such as C and R , represent sets.
- \leq_X : a partial order over set X .
- Greek letters, such as σ and ι , represents functions.
- $+$: a Kleene plus on a set. For example, the Kleene plus on set X is

$$X^+ = \bigcup_{i \in \mathbb{N}^*} X_i = X_1 \cup X_2 \cup X_3 \cup \dots$$
- $|X|$: represents the size of set X .
- \times : the notation for a Cartesian product. The Cartesian product of two sets X and Y , denoted $X \times Y$, is the set of all possible ordered pairs whose first component is a member of X and whose second component is a member of Y : $X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$.
- $\pi_i X$, $1 \leq i \leq |X|$: the Cartesian product of an arbitrary family of sets for set X .
- 2^I : the power set (or powerset) of I , which is the set of all subsets of I .
- $\llbracket t \rrbracket$: the values of t .

According to Karlsruhe's group (Stumme et al., 2003), a core ontology is defined as a structure

$$O := (C, \leq_C, R, \sigma, \leq_R)$$

where

- C and R are two disjoint sets whose elements are concept identifiers and relation identifiers respectively,
- a partial order \leq_C on C represents the hierarchical relationship between concepts,
- $\sigma : R \rightarrow C^+$ is a signature that describes the non-logical symbols of the formal language,
- a partial order \leq_R on R represents the hierarchical relationship between properties. $r_1 \leq_R r_2$ implies $|\sigma(r_1)| = |\sigma(r_2)|$ and $\pi_i(\sigma(r_1)) \leq_C \pi_i(\sigma(r_2))$, $1 \leq i \leq |\sigma(r_1)|$.

To better represent the attributes of ontologies, Cimiano (2006) proposes to refine this definition and instead defining an ontology as a structure

$$O := (C', \leq_{C'}, R', \sigma_{R'}, \leq_{R'}, A, \sigma_A, T)$$

where

- C' , R' , A , and T are four disjoint sets whose elements are respectively concept identifiers, relation identifiers, attribute identifiers and data types,
- a partial order $\leq_{C'}$ on C' represents the hierarchical relationship between concepts,
- a function $\sigma_{R'} : R' \rightarrow C'^+$ is the relation signature,

- a partial order $\leq_{R'}$ on R' represents the relation hierarchy. $r_1 \leq_{R'} r_2$
implies $|\sigma(r_1)| = |\sigma(r_2)|$ and $\pi_i(\sigma(r_1)) \leq_C \pi_i(\sigma(r_2))$, $1 \leq i \leq |\sigma(r_1)|$.
- a function $\sigma_A : A \rightarrow C \times T$ is the attribute signature.

In this definition, the concept set is divided into two sets: C' represents the concepts abstracted from the real world and T represents the data type set specifically. This separation of concepts can better reflect the elements in ontology languages. For example: C' can represent the classes which reflect the abstraction of objects and T represents the set of RDF literals and XML Schema data types in OWL. Correspondingly, the relation set R and function σ are also separated: R' and $\sigma_{R'}$ represent the relations between concepts in C' ; A and σ_A represent the relations between concepts and data types. However, this definition ignores the function $\sigma_{R''} : R'' \rightarrow T \times T$, which may limit the usage of ontology. For example, with this ontology definition users cannot use this ontology to create user-defined data types.

Apart from ignoring the function $\sigma_{R''}$, Cimiano's definition also ignores two other important definitions. First, it only defined hierarchical relationships for the elements of R' , denoted by $\leq_{R'}$. However, the elements of A also have hierarchical relationships because attributes can be considered specific kinds of properties whose ranges are data types. Therefore, a new partial order \leq_A which represents the hierarchical relationship between attributes is needed. Second, in both Cimiano and Karlsruhe's definitions, the function $\sigma_{R'}$ is defined as $\sigma_{R'} : R' \rightarrow C^{1+}$. In fact, the relations in an ontology are generally binary relations, which means that for a relation $r \in R'$, $|\sigma(r)|$ is 2. Therefore, to be more explicit, the function $\sigma_{R'}$

can be defined as $\sigma_{R'} : R' \rightarrow C \times C'$.

Given the above observation, we adopt and refine Karlsruhe's definition of core ontology (Stumme et al., 2003) according to some of the concepts in Cimiano's definition (Cimiano, 2006). Our definition of core ontology is as follows:

Definition (Core Ontology): *A core ontology is defined as a structure*

$$O := (C, \leq_C, R, \sigma, \leq_R)$$

consisting of

- *two disjoint sets C and R whose elements are respectively called concept identifiers and relation identifiers.*
 - *Data type can be considered as a specific type of concept, which can be represented by a set DT , $DT \subseteq C$. In OWL, DT can represent the set of XML Schema or RDF data types.*
 - *A set DR can also be defined, which contains the relation identifiers representing the relationship between a concept and a data type, $DR \subseteq R$.*
- *a partial order \leq_C on C , called concept hierarchy or taxonomy, represents the hierarchical relationship between concepts.*
 - *If $c_1 <_C c_2$, for $c_1, c_2 \in C$, then c_1 is a **sub-concept** of c_2 , and c_2 is a **super-concept** of c_1 . In OWL, \leq_C can map to the relationship between classes and their sub-classes.*
 - *If $c_1 <_C c_2$ and there is no $c_3 \in C$ satisfying $c_1 <_C c_3 <_C c_2$, then c_2 is a **direct super-concept** of c_1 and c_1 is a **direct sub-concept** of c_2 . This can be denoted by $c_1 \prec c_2$.*

- a function $\sigma : R \rightarrow C \times C$, called *relation signature*, where for a relation $r \in R$, $\sigma(r) = \langle \text{dom}(r), \text{ran}(r) \rangle$, $\text{dom}(r)$ is the **domain** of relation r and $\text{ran}(r)$ is the **range** of relation r .
- For the function σ , we have $DR \subseteq R$ and $DT \subseteq C$, then we can define $\sigma|_{DR} : DR \rightarrow C \times DT$, which can represent the data type property in OWL. For a relation $r \in DR$, the **domain** and **range** of r are limited as: $\text{dom}(r) \in C$ and $\text{ran}(r) \in DT$.
- For a relation $r \in R$, if $\text{dom}(r) \in C$, $\text{ran}(r) \in C$ and $\text{ran}(r) \notin DT$, the relation r represents the object property in OWL.
- a partial order \leq_R on R , called *relation hierarchy*, where $r_1 \leq_R r_2$ implies $\text{dom}(r_1) \leq_C \text{dom}(r_2)$ and $\text{ran}(r_1) \leq_C \text{ran}(r_2)$.
- If $r_1 <_R r_2$, for $r_1, r_2 \in R$, then r_1 is a **sub-relation** of r_2 , and r_2 is a **super-relation** of r_1 . In OWL, \leq_R can map to the relationship between properties and their sub-properties.
- If $r_1 <_R r_2$ and there is no $r_3 \in R$ satisfying $r_1 <_R r_3 <_R r_2$, then r_2 is a **direct super-relation** of r_1 and r_1 is a **direct sub-relation** of r_2 . This can be denoted by $r_1 \prec r_2$.

Axioms constrain the interpretation and well-formed use of the concepts (Gruber, 1995). The definition of axioms in ontology is as follows:

Definition (Axiom): Let L be a logical language. An L -axiom system for an ontology $O := (C, \leq_C, R, \sigma, \leq_R)$ is a triple

$$A := (AI, \alpha, L)$$

where

- AI is a set whose elements are called axiom identifiers,
- $\alpha : AI \rightarrow L$ is a mapping from AI to L .

The elements of $A := \alpha(AI)$ are called axioms.

An ontology with L -axioms is a pair

$$(O, A)$$

where O is an ontology and A is an L -axiom system for O .

A lexicon for an ontology can provide mapping between concepts or relations (Hirst, 2004). It can assist the communication between users who use different terms and it can also be useful for system integration. For example, the subsidiaries of an enterprise in different countries may use different languages to describe the same objects, and the lexicon can provide a bridge between the languages and the knowledge defined in the ontology so that the system definition can be unified. Although we are not directly concerned with lexicons in this study, the lexicon is still needed, especially for the construction of a unified ontology for a specific domain. A lexicon is defined as follows (Stumme et al., 2003):

Definition (Lexicon): A lexicon for an ontology $O := (C, \leq_C, R, \sigma, \leq_R)$ is a structure

$$Lex := (S_C, S_R, Ref_C, Ref_R)$$

consisting of

- two sets S_C and S_R whose elements are respectively called signs for concepts and signs for relations,
- a relation $Ref_C \subseteq S_C \times C$ called lexical reference for concepts, where $(c, c) \in Ref_C$ holds for all $c \in C \cap S_C$.
- a relation $Ref_R \subseteq S_R \times R$ called lexical reference for relations, where $(r, r) \in Ref_R$ holds for all $r \in R \cap S_R$.
- based on Ref_C , for $s \in S_C$, $Ref_C(s) := \{ c \in C \mid (s, c) \in Ref_C \}$, and for $c \in C$, $Ref_C^{-1}(c) := \{ s \in S_C \mid (s, c) \in Ref_C \}$
- Ref_R and Ref_R^{-1} can be defined analogously: for $s' \in S_R$, $Ref_R(s') := \{ r \in R \mid (s', r) \in Ref_R \}$, and for $r \in R$, $Ref_R^{-1}(r) := \{ s' \in S_R \mid (s', r) \in Ref_R \}$

An ontology with a lexicon is a pair

$$(O, Lex)$$

where O is an ontology and Lex is a lexicon for O .

A knowledge base can provide the means for the computerized collection, organization, and retrieval of knowledge. It can provide the extensional aspects such as assertions about instances of the concepts and relations. A knowledge base can be defined as (Stumme et al., 2003):

Definition (Knowledge Base): A Knowledge base for an ontology

$O := (C, \leq_C, R, \sigma, \leq_R)$ is a structure

$$KB := (C, R, I, \iota_C, \iota_R)$$

consisting of

- two disjoint sets C and R as defined in ontology O ,
- a set I whose elements are called instance identifiers, which are individuals in OWL,
- a function $\iota_C : C \rightarrow 2^I$ called concept instantiation, especially
 - a function $\iota_{DT} : DT \rightarrow \bigcup_{t \in DT} \llbracket t \rrbracket$ is the instantiation of the data types, where $\llbracket t \rrbracket$ are the values of data type $t \in DT$.
- a function $\iota_R : R \rightarrow 2^{I^+}$ with $\iota_R(r) \subseteq \iota_C(\text{dom}(r)) \times \iota_C(\text{ran}(r))$, $\forall r \in R$. This function is called a relation instantiation, especially
 - a function $\iota_{DR} : DR \rightarrow I \times \bigcup_{t \in DT} \llbracket t \rrbracket$ with $\iota_{DR}(a) \subseteq \iota_C(\text{dom}(a)) \times \llbracket \text{ran}(a) \rrbracket$, $\forall a \in DR$. This function can represent the instantiation of data type property in OWL.

3.2 Business Process Ontology (BPO)

BPO is a core ontology for business domains which can be used for modeling business processes and for representing business processes and services information and business process system integration. BPO is designed with OWL (W3C, 2004c) and its framework is represented with a UML graph, as shown in Figure 6.

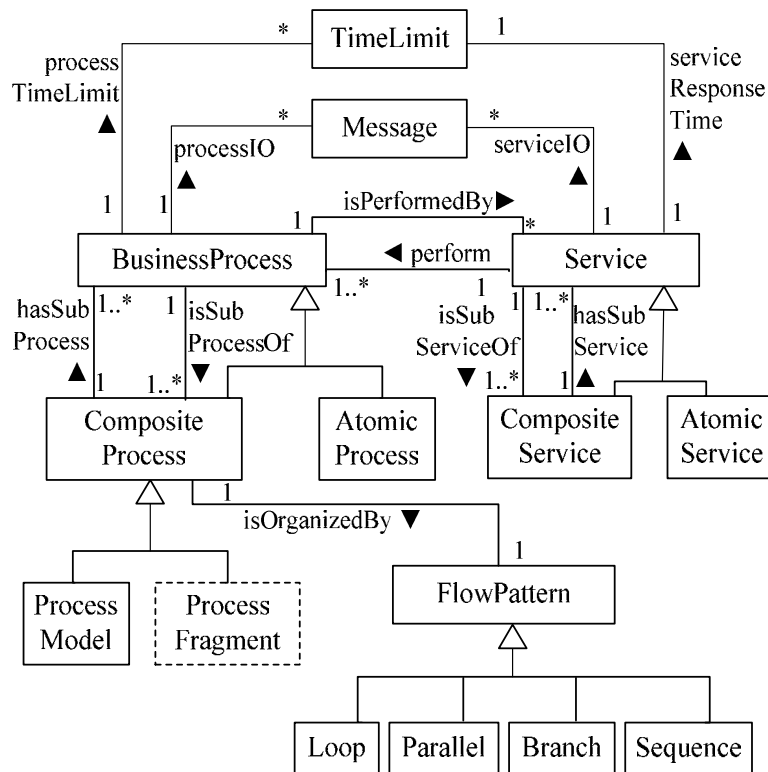


Figure 6. Framework of BPO

3.2.1 Concept Set of BPO

As BPO is a core ontology for business domains, its main concepts are the most abstract ones for business process modeling. These can be classified into three categories: business process-related, service-related, and others. The business process-related concepts represent five kinds of business processes in a business domain or in an organization:

- **BusinessProcess** represents the abstraction of all kinds of business processes of an organization.
- **AtomicProcess** represents the processes whose functions should be conducted as a whole and could not be subdivided.
- **CompositeProcess** represents processes that are composed of one or more

business processes, which can be atomic processes and other composite processes.

- **ProcessModel** represents all the reusable business process models in an organization which model a whole workflow of a business process model.
- **ProcessFragment** represents an encapsulated step in a composite process model which is organized by a kind of “FlowPattern”. It is drawn in a dashed box because it is an auxiliary concept for constructing the workflow of a business process model.

The relationships between AtomicProcess, CompositeProcess and BusinessProcess are $AtomicProcess <_C BusinessProcess$ and $CompositeProcess <_C BusinessProcess$, which means that AtomicProcess and CompositeProcess are **sub-concepts** of BusinessProcess, and BusinessProcess is a **super-concept** of both of them.

Because there is no $c_1 \in C$ satisfying $AtomicProcess <_C c_1 <_C BusinessProcess$ or $CompositeProcess <_C c_1 <_C BusinessProcess$, BusinessProcess is a **direct super-concept** of AtomicProcess and CompositeProcess and the latter two are **direct sub-concepts** of the former concept. This can be denoted as $AtomicProcess \prec BusinessProcess$ and $CompositeProcess \prec BusinessProcess$. In BPO, these concepts are defined as classes. The AtomicProcess and CompositeProcess classes are defined as sub-classes of BusinessProcess class with OWL statements.

Analogously, the ProcessModel and ProcessFragment concepts are defined as sub-classes of the CompositeProcess class and can be denoted as $ProcessModel \prec CompositeProcess$ and $ProcessFragment \prec CompositeProcess$. Because of the transitivity of \leq_C , BusinessProcess is another super-concept of

ProcessModel and ProcessFragment. The transitivity is a natural attribute of “rdfs:subClassOf” relation in OWL.

BPO can be extended to form a knowledge base for all the business process models in an organization. However, the organizations need to reuse not only the business process models but also the related services. If BPO includes only process-related information, its reusability will be limited. Therefore, BPO also defines service related classes and identifies the relationships between business processes and services. The service related classes are:

- **Service** represents the super-class of all the available services in an organization’s asset bases.
- **AtomicService** represents a service that can fully implement a business process and is considered as a black box during the modeling.
- **CompositeService** represents a service that is composed of several other services, which can be atomic services and other composite services.

A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description (OASIS, 2006). For example, in the automotive software domain, a service can represent a piece of functionality in an automobile. The Service class and its sub-classes are used to provide information associated with the services, such as the functional description, the location, and performance information. These classes can be used to build the mapping between business processes and services, models, and the implementation.

The modeling of processes also requires other concepts such as the concepts for communication and workflow description. These additional classes include:

- **TimeLimit** representing the time limitation of the execution of processes

and services.

- **Message** representing the data exchange between processes or services. If necessary, users can define sub-classes of the Message class to represent different kinds of messages for a specific business domain.
- **FlowPattern** representing the abstraction of the descriptions of control flow dependencies between the component processes within a composite process. Four basic patterns are defined in BPO:
 - **Sequence** executes a list of composite or atomic processes in order.
 - **Branch** chooses an execution path from several alternatives.
 - **Loop** repeatedly executes a business process or a set of business processes.
 - **Parallel** allows the business processes to be executed concurrently by different processors.

In summary, the concept set of BPO is

$$C := \text{DomainConcept} \cup \text{DataType}$$

$\text{DomainConcept} := \{ \text{AtomicProcess}, \text{AtomicService}, \text{Branch}, \text{BusinessProcess}, \text{CompositeProcess}, \text{CompositeService}, \text{FlowPattern}, \text{Loop}, \text{Message}, \text{Parallel}, \text{ProcessFragment}, \text{ProcessModel}, \text{Sequence}, \text{Service}, \text{TimeLimit} \}$

where

- The elements of *DomainConcept* represent the main objects in business process modeling.
- The elements of *DataType* are the *XML Schema* or *RDF data types*.

The hierarchical relationships \leq_C between the concepts are summarized in Table 4. “OWL: Thing” is the super-concept for all the concepts in BPO. For simplicity, we identify this only when it is the direct super-concept of other concepts.

Table 4. Hierarchical relationships between the concepts in BPO

	Direct Super-Concept	Super-Concepts³	Direct Sub-Concept	Sub-Concepts
AtomicProcess	BusinessProcess	BusinessProcess		
AtomicService	Service	Service		
Branch	FlowPattern	FlowPattern		
BusinessProcess	OWL: Thing		AtomicProcess, CompositeProcess	AtomicProcess, CompositeProcess, ProcessModel, ProcessFragment
CompositeProcess	BusinessProcess	BusinessProcess	ProcessModel, ProcessFragment	ProcessModel, ProcessFragment
CompositeService	Service	Service		
FlowPattern	OWL: Thing		Sequence, Branch, Loop, Parallel	Sequence, Branch, Loop, Parallel
Loop	FlowPattern	FlowPattern		
Message	OWL: Thing			
Parallel	FlowPattern	FlowPattern		
ProcessFragment	CompositeProcess	CompositeProcess, BusinessProcess		
ProcessModel	CompositeProcess	CompositeProcess, BusinessProcess		
Sequence	FlowPattern	FlowPattern		
Service	OWL: Thing		AtomicService, CompositeService	AtomicService, CompositeService
TimeLimit	OWL: Thing			

3.2.2 Relation Set of BPO

According to Zachman Framework (The Open Group 2006), six categories of information are necessary to be identified in a system architecture:

- The data description — What

³ This column only shows the super-concepts other than “OWL:Thing”.

- The function description — How
- The Network description — Where
- The people description — Who
- The time description — When
- The motivation description — Why

Therefore, in the framework of BPO, we should be able to describe the functionality of the processes, the data transmission between the processes, the performance requirements of the processes and relevant services and the actor of the processes.

On this basis, the relation set of BPO is defined as:

$$R := \textit{ObjectProperty} \cup \textit{DataTypeProperty}$$

ObjectProperty := {hasSubProcess, isSubProcessOf, hasSubService, isSubServiceOf, isPerformedBy, perform, isOrganizedBy, processIO, processInputs, processOutputs, serviceIO, serviceInputs, serviceOutputs, processTimeLimit, serviceResponseTime}

where

- The elements of *ObjectProperty* are the relation identifiers which represent the main relations between the instances of the concepts in the *DomainConcept* set.
- The elements of *DataTypeProperty* are the relation identifiers which identify the relation between instances of concepts and RDF literals and XML Schema data types.

The main object properties are represented by the connections between the classes in Figure 6 and are illustrated in detail in Table 5. An object property is a binary relation which is defined as a relation $r \in R$ with $|\sigma(r)| = 2$, representing a

relation between instances of two classes. The domain of an object property r is defined as $dom(r) := \pi_1(\sigma(r))$, and its range is defined as $range(r) := \pi_2(\sigma(r))$. In Table 5, the domain of an object property is the first element of σ_R and the range is the second element of σ_R .

Table 5. Main object properties in BPO

Relation Identifier	σ_R	Explanation
hasSubProcess	<CompositeProcess, BusinessProcess>	A transitive property. States the sub-processes of a composite business process. Its sub-processes can be both composite and atomic business processes. This is the inverse property of isSubProcessOf.
isSubProcessOf	<BusinessProcess, CompositeProcess>	A transitive property. Identifies a business process that is a component of another business process. The former can be either atomic or composite processes and the latter should be a composite process. This is the inverse property of hasSubProcess.
hasSubService	<CompositeService, Service>	A transitive property. States the components of a composite service. A composite service can be constructed by both composite and atomic services. This is the inverse property of isSubServiceOf.
isSubServiceOf	<Service, CompositeService>	A transitive property. Identifies a service that is a component of another service. The former can be either atomic or composite service and the latter should be a composite service. This is the inverse property of hasSubService.
Perform	<Service, BusinessProcess>	Identifies the binding information of services and business processes. A service can be engaged in one or more business processes. This property is the inverse property of isPerformedBy.
isPerformedBy	<BusinessProcess, Service>	Identifies the binding information of business processes and services. A business process can be implemented by one or more services. If the process is conducted manually, no service will support it. This property is the inverse property of perform.
isOrganizedBy	<CompositeProcess, FlowPattern>	States the flow pattern of the components in a composite process.

processIO	<BusinessProcess, Message>	Refers to the types of the information transfer between different business processes. This has two sub-properties: processInputs and processOutputs.
processInputs	<BusinessProcess, Message>	Specifies the type of messages that a process requires for its execution. This is a sub-property of processIO.
processOutputs	<BusinessProcess, Message>	Specifies the type of results of a process. This is a sub-property of processIO.
serviceIO	<Service, Message>	Refers to the information transfer between different services. This has two sub-properties: serviceInputs and serviceOutputs.
serviceInputs	<Service, Message>	Specifies the inputs that a service requires for its execution. This is a sub-property of serviceIO.
serviceOutputs	<Service, Message>	Specifies the outputs of a service. This is a sub-property of serviceIO.
process-TimeLimit	<BusinessProcess, TimeLimit>	Specifies the acceptable range of execution time for a process.
service-ResponseTime	<Service, TimeLimit>	Specifies the actual response time of a service.

As shown in Table 5, some properties of Business Process and Service have similar names, such as processIO and serviceIO properties and their sub-properties. Nonetheless, they represent very different concepts. The business process ontology is developed from the users' perspective. In contrast, the service ontology provides information about concrete services. Therefore, the processIO property just describes what kind of communication occurs between the processes whereas the serviceIO property provides the concrete constraints (such as data types) for the messages which are transferred between services. The business process ontology can be used in constructing process models. The service ontology provides information of the implementation.

Data type properties are also needed to describe classes and individuals. We list the essential data type properties of our framework in Table 6.

Table 6. Main data type properties in BPO

Relation Identifier	$\sigma_{Data\ Type\ Property}$	Explanation
processName	<BusinessProcess, xsd:string>	Refers to the name of the offered business process.
processDescription	<BusinessProcess, xsd:string>	Provides a brief description of business processes: summarizing the purpose of a business process, functions of a business process, the requirements of the business process, the results of the business process, and the execution mode of the business process (e.g. manual or computer aided).
preCondition	<BusinessProcess, xsd:string>	States the conditions under which a business process can be conducted.
postCondition	<BusinessProcess, xsd:string>	States the situation after a business process is conducted. This has two sub-properties: successGuarantee and minimalGuarantee.
role	<BusinessProcess, xsd:string>	Describes the information of roles or actors of the business process.
successGuarantee	<BusinessProcess, xsd:string>	States what must be satisfied after the business process is successfully executed.
minimalGuarantee	<BusinessProcess, xsd:string>	States what must be satisfied after execution of a business process, no matter the execution was successful or not.
serviceName	<Service, xsd:string>	Refers to the name of the offered service.
serviceDescription	<Service, xsd:string>	Provides a brief description of a service: function, resource needed, type of inputs and outputs, and any additional information that the users need to know.
serviceLocation	<Service, xsd:string>	Identifies the location of a service.

The relation hierarchies between some of the properties are summarized in Table 7.

Table 7. Relation hierarchies between the properties in BPO

	Direct Super-Relation	Super-Relation	Direct Sub-Relation	Sub-Relation
processIO			processInputs processOutputs	processInputs processOutputs
processInputs	processIO	processIO		
processOutputs	processIO	processIO		
serviceIO			serviceInputs	serviceInputs

			serviceOutputs	serviceOutputs
serviceInputs	serviceIO	serviceIO		
serviceOutputs	serviceIO	serviceIO		
postCondition			successGuarantee, minimalGuarantee	successGuarantee, minimalGuarantee
successGuarantee	postCondition	postCondition		
minimalGuarantee	postCondition	postCondition		

Considering the six categories of information in Zachman Framework, we can find that most of the information is described by BPO's classes and properties. The class Message represents the data exchange between processes or services. Developers can describe the motivation and function of a business process in its property processDescription. The mapping between the business process and services identifies which services can conduct this business process. The property preCondition states the conditions under which a business process can be conducted or when the business process can start, and the role property describes the information of roles or actors of a business process.

Additional classes and properties may be needed to describe the objects in a specific business domain. For example, a Unit class can define the units of values, such as "Micro-Seconds" and "Seconds" to represent the unit of time limitation. More performance features can also be added according to the performance requirements of a specific domain. For example, sub-properties throughput and maxNumberOfCustomers can be added to represent the non-functional requirements of the business processes of a web site.

Once we have these definitions of the concepts and relations, we can construct the framework for a business process model and identify the mapping between the process model and services. A business process can be performed by one or more services (identified by the isPerformedBy property); a service may "perform" one or

more processes. Except for the manual business processes (which would be identified in the `processDescription` property), the business processes may be mapped to existing services. Generally, atomic processes may be implemented by atomic services and composite services are often mapped to composite processes. Suitable services can be adopted according to the performance requirements of a business process. By separating the business processes and services, organizations could produce applications with more flexibility. For example, a business process can map to several services which have different performance and cost; the developers can choose the most suitable service according to their specific requirements.

3.2.3 The Formal Description of BPO

To conclude the definitions for the concepts and relations, the core ontology BPO can be defined as a structure:

$$BPO := (C, \leq_C, R, \sigma, \leq_R)$$

consisting of

- $C := \text{DomainConcept} \cup \text{DataType}$
 - $\text{DomainConcept} := \{ \text{AtomicProcess}, \text{AtomicService}, \text{Branch}, \text{BusinessProcess}, \text{CompositeProcess}, \text{CompositeService}, \text{FlowPattern}, \text{Loop}, \text{Message}, \text{Parallel}, \text{ProcessFragment}, \text{ProcessModel}, \text{Sequence}, \text{Service}, \text{TimeLimit} \}$
 - $\text{DataType} := \text{The set of Data Types in XML Schema}$
- $R := \text{ObjectProperty} \cup \text{DataTypeProperty}$

- *ObjectProperty* := { *hasSubProcess*, *isSubProcessOf*, *hasSubService*, *isSubServiceOf*, *isPerformedBy*, *perform*, *isOrganizedBy*, *processIO*, *processInputs*, *processOutputs*, *serviceIO*, *serviceInputs*, *serviceOutputs*, *processTimeLimit*, *serviceResponseTime* }
- *DataTypeProperty* := The set of Data Type Attributes in XML Schema
- The partial order \leq_c over C represents the concept hierarchy in BPO. The direct super-class relation is shown as:
 - $R_{ClassHierarchy} := (\langle AtomicProcess, BusinessProcess \rangle, \langle AtomicService, Service \rangle, \langle Branch, FlowPattern \rangle, \langle CompositeProcess, BusinessProcess \rangle, \langle CompositeService, Service \rangle, \langle Loop, FlowPattern \rangle, \langle Parallel, FlowPattern \rangle, \langle ProcessFragment, CompositeProcess \rangle, \langle ProcessModel, CompositeProcess \rangle, \langle Sequence, FlowPattern \rangle)$
- For $r \in R$, $\sigma(r) = \langle dom(r), ran(r) \rangle$. The function $\sigma(r)$ is defined as:
 - For object properties:
 - ◆ $\sigma(hasSubProcess) = \langle CompositeProcess, BusinessProcess \rangle$
 - ◆ $\sigma(isSubProcessOf) = \langle BusinessProcess, CompositeProcess \rangle$
 - ◆ $\sigma(hasSubService) = \langle CompositeService, Service \rangle$
 - ◆ $\sigma(isSubServiceOf) = \langle Service, CompositeService \rangle$
 - ◆ $\sigma(Perform) = \langle Service, BusinessProcess \rangle$
 - ◆ $\sigma(isPerformedBy) = \langle BusinessProcess, Service \rangle$
 - ◆ $\sigma(isOrganizedBy) = \langle CompositeProcess, FlowPattern \rangle$
 - ◆ $\sigma(processIO) = \langle BusinessProcess, Message \rangle$
 - ◆ $\sigma(processInputs) = \langle BusinessProcess, Message \rangle$
 - ◆ $\sigma(processOutputs) = \langle BusinessProcess, Message \rangle$

- ◆ $\sigma(\text{serviceIO}) = \langle \text{Service}, \text{Message} \rangle$
- ◆ $\sigma(\text{serviceInputs}) = \langle \text{Service}, \text{Message} \rangle$
- ◆ $\sigma(\text{serviceOutputs}) = \langle \text{Service}, \text{Message} \rangle$
- ◆ $\sigma(\text{processTimeLimit}) = \langle \text{BusinessProcess}, \text{TimeLimit} \rangle$
- ◆ $\sigma(\text{serviceResponseTime}) = \langle \text{Service}, \text{TimeLimit} \rangle$
- For data type properties:
 - ◆ $\sigma(\text{processName}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{processDescription}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{precondition}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{postCondition}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{role}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{successGuarantee}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{minimalGuarantee}) = \langle \text{BusinessProcess}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{serviceName}) = \langle \text{Service}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{serviceDescription}) = \langle \text{Service}, \text{xsd:string} \rangle$
 - ◆ $\sigma(\text{serviceLocation}) = \langle \text{Service}, \text{xsd:string} \rangle$
- The partial order \leq_R over R represents the relation hierarchy in BPO. The direct super-relation is shown as:
 - $R_{\text{RelationHierarchy}} = (\langle \text{processInputs}, \text{processIO} \rangle, \langle \text{processOutputs}, \text{processIO} \rangle, \langle \text{serviceInputs}, \text{serviceIO} \rangle, \langle \text{serviceOutputs}, \text{serviceIO} \rangle, \langle \text{successGuarantee}, \text{postCondition} \rangle, \langle \text{minimalGuarantee}, \text{postCondition} \rangle)$

3.2.4 Axioms for BPO

Different types of axioms can be stated in OWL for ontologies such as class axioms, object property axioms, data property axioms, and fact axioms (W3C, 2008a).

Two types of class axioms are adopted in constructing BPO: *SubClassOf* and *DisjointClasses*.

- *SubClassOf* axiom represents the hierarchical relationships between concepts. *SubClassOf (B A)*, in which B is a sub-class of A, means “B implies A”.
 - For example, *SubClassOf (AtomicProcess Process)* means that atomic processes are a type of processes. More examples can be found in Table 4.
- *DisjointClasses* axiom takes a set of descriptions and states that all descriptions from the set are pair-wise disjoint.
 - Five sets of disjoint classes are defined in BPO:
 - ◆ *DisjointClasses (BusinessProcess Service Message TimeLimit FlowPattern)*
 - ◆ *DisjointClasses (AtomicProcess CompositeProcess)*
 - ◆ *DisjointClasses (AtomicService CompositeService)*
 - ◆ *DisjointClasses (ProcessModel ProcessFragment)*
 - ◆ *DisjointClasses (Loop Parallel Branch Sequence)*

Five types of object property axioms are adopted in BPO. They are *SubObjectPropertyOf*, *ObjectPropertyDomain*, *ObjectPropertyRange*, *InverseObjectProperties* and *TransitiveObjectProperties*.

- *SubObjectPropertyOf* axiom represents the hierarchical relationships between object properties.
 - For example, *SubObjectPropertyOf (processInputs processIO)* means that processInputs are a type of processIO. More examples can be found in Table 7.
- *ObjectPropertyDomain* and *ObjectPropertyRange* axioms define the domain and range constraints in OWL which can be used in reasoning.
 - For example, *ObjectPropertyDomain (isPerformedBy BusinessProcess)* and *ObjectPropertyRange (isPerformedBy Service)* define a relation r , where $\sigma(r) = \langle BusinessProcess, Service \rangle$. The domain and range definitions for other objectProperties of BPO are presented in Table 5.
- *InverseObjectProperties* axiom states that two properties are the inverse of each other.
 - Three pairs of properties are inverse properties:
 - ◆ *InverseObjectProperties (isPerformedBy perform)*, which means $isPerformedBy(A, B) \leftrightarrow perform(B, A)$;
 - ◆ *InverseObjectProperties (hasSubProcess isSubProcessOf)*, which means $hasSubProcess(A, B) \leftrightarrow isSubProcessOf(B, A)$;
 - ◆ *InverseObjectProperties (hasSubService isSubServiceOf)*, which means $hasSubService(A, B) \leftrightarrow isSubServiceOf(B, A)$;
- *TransitiveObjectProperties* axiom: If a property P is specified as transitive object property, then for any x, y and z : $P(x, y)$ and $P(y, z)$ imply $P(x, z)$.
 - Four properties are defined as transitive properties:
 - ◆ *TransitiveObjectProperties (hasSubProcess)*

- ◆ *TransitiveObjectProperties (isSubProcessOf)*
- ◆ *TransitiveObjectProperties (hasSubService)*
- ◆ *TransitiveObjectProperties (isSubServiceOf)*

Three types of data property axioms, *SubDataPropertyOf*, *DataPropertyDomain* and *DataPropertyRange*, are used in BPO.

- Similar to the *SubObjectPropertyOf* axiom, *SubDataPropertyOf* axiom represents the hierarchical relationships between data properties.
 - *SubDataPropertyOf (successGuarantee postCondition)* means that *successGuarantee* is a kind of *postCondition*.
 - *SubDataPropertyOf (minimalGuarantee postCondition)* means that *minimalGuarantee* is also a kind of *postCondition*.
- *DataPropertyDomain* and *DataPropertyRange* axioms define the domain and range constraints of data properties.
 - For example, *DataPropertyDomain (processName BusinessProcess)* and *DataPropertyRange (processName xsd: string)* define a data type relation r , where $\sigma(r) = \langle \text{BusinessProcess}, \text{xsd} : \text{string} \rangle$. The domain and range definitions for other data type properties of BPO are presented in Table 6.

OWL adopts Open World Reasoning, which treats something stated as false only if it can be proved to contradict other information in the ontology (Rector et al., 2004). These axioms for the concepts and relations allow us to restrict the objects defined in the ontology and ensure the consistency of the definitions. For example, if *isPerformedBy (A, B)* is defined for process *A*, the attribute *perform (B, A)* must be defined correspondingly for service *B* because *isPerformedBy* and *perform* properties are a pair of *InverseObjectProperties*.

3.2.5 Comparison of BPO and OWL-S

As described in Section 2.2.2, OWL-S (W3C, 2004e) is an upper ontology which concerns the automatic discovery and composition of web services. As the service composition is closely related to business process modeling, this section will compare OWL-S with BPO, highlight their differences, and explain their relation.

OWL-S is an ontology of service proposed by W3C. It has an OWL-based framework which can be used for describing various aspects of web services. OWL-S defines a top-level concept “Service” and three sub-classes: “ServiceProfile”, “ServiceModel” and “ServiceGrounding”. “ServiceProfile” is for service advertising and discovery on the net, while “ServiceModel” exposes how a service is composed. “ServiceGrounding” specifies the details of how to access the service.

OWL-S also adopts “Process” to represent service model. However, this “Process” is not the same as the BusinessProcess in BPO. The “Process” class in OWL-S represents a process in a web service and it gives a detailed perspective on how a client may interact with a service. In BPO, BusinessProcess represents any actions or procedures to handle business tasks in an organization. No matter the business processes are handled manually or automatically, they will be faithfully described in BPO.

Although both BPO and OWL-S are upper ontologies for SOA, they focus on different aspects.

- They are from different levels of viewpoint. BPO is from the viewpoint of system engineering which is a higher level concerning more about the workflow and user’s requirement specification; on the other hand, OWL-S is more from the technical viewpoint, concerning about the automatic web

service discovery and composition, which is at a lower level.

- They provide different kinds of information. BPO can provide accurate descriptions of reusable business process models, information of available services in an organization's asset base and previous experiences of the implementation of the business process models. OWL-S provides two kinds of information for a service, a brief introduction and implementation details. The introduction is described by "ServiceProfile" class, and the implementation details are represented by "ServiceModel" and "Service Grounding" classes.
- They support different users. BPO is targeted for internal use; business people can use it to construct the business process model and technical people can use it to maintain the information of business processes and services. With OWL-S, service providers can advertise their services on the net; and services users and software agents can discover, invoke, compose, and monitor the web resources.

From this discussion, we can conclude that BPO and OWL-S can be used at different phases of SOA application development. BPO can be used in the modeling phase to construct a model of the whole system and OWL-S can be used in the assemble phase to compose related services.

3.3 Ontology-based Business Process Modeling and Developing Framework

We propose an Ontology-based Business Process Modeling and Developing Framework (OBPMDF) to illustrate the main processes and products in the Model and Assemble phases of the SOA development lifecycle. The framework applied the extension of BPO as the knowledge base in a specific business domain, providing accurate definitions of business process models and the information of their relevant services. The framework is shown in Figure 7.

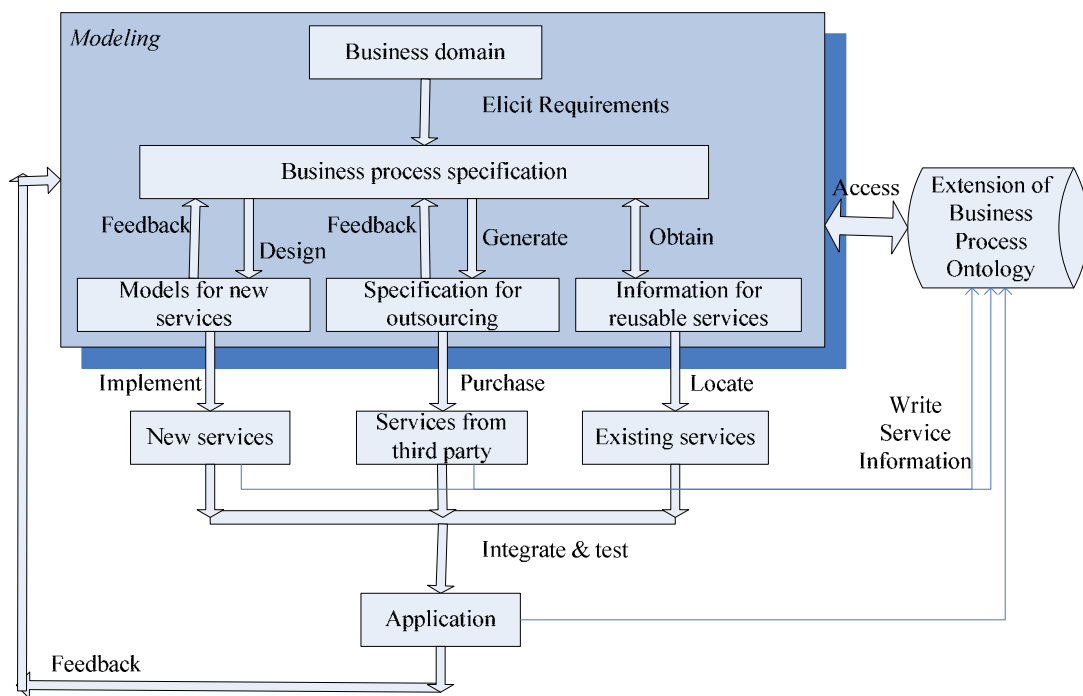


Figure 7. Ontology-based Business Process Modeling and Developing Framework

Our OBPMDF begins at the modeling stage, which can be divided into four steps:

- 1) Create business process specification

Within OBPMDF, the first step of modeling is still requirement elicitation, which produces an initial requirement specification. By analyzing the requirements, the developers can divide the target business process into a set of composite activities and atomic activities. Then the developers can access the knowledge base to determine whether there are any reusable business process models or services that can satisfy the target business or any of its sub-activities. Parts of the business process may have been designed and implemented and other parts may not. By accessing the knowledge base, developers can obtain the formal specifications of the reusable part and its implementation information. The developers can reuse this part directly. For the other part, they need to create new process models. After the modeling of the sub-activities and orchestrating them, it is possible to generate a formal business process specification for integration. The business process specification can reflect the user needs at a high level of abstraction.

2) Obtain reusable services

Because the knowledge base contains information about the business processes and services, especially their mutual relationships, it is easy to query the knowledge base to discover services which can implement a specific business process model. If the business process has been implemented, at least one service for that business process should be found from the knowledge base. The attributes of the services can show their functions, performance and locations. The developers can then determine whether the services can be reused or not.

3) Create models for new services

After the business process modeling, the target business process model is composed of two kinds of business processes: reusable business processes and new business processes. The services for the reusable business processes can be obtained

from the knowledge base. The developers need to implement the other new business processes. Because the business processes are defined from a system viewpoint and at a high level of abstraction, related service models should be constructed in more detail, satisfying not only functional requirements but also technical requirements.

4) Generate specification for outsourcing

Sometimes, some of the services may be obtained by outsourcing. In such cases, specifications for these outsourced services should be constructed and the main performance requirements should also be defined.

The modeling methods in the Model stage, which will be illustrated in Section 3.4, focus mainly on the first step, which concerns business process modeling. After this, the target business process can be formally described using OWL. As OWL is a formal language for knowledge description, the process models written in OWL have two kinds of flexibility. First, they can be translated into other process modeling languages, such as WSBPEL. Second, they can be recognized and handled by computer, supporting automatic simulation of the models. If any conflicts are found in the model simulation, the model will be refined.

After the Model stage, the developers should have a complete model of the target business process which can be used for integration, reusable services which can satisfy part of the functions of the business, and specifications for new service implementation and outsourcing. At the development stage, new services will be developed first, and then all the related services will be integrated as defined in the business process model. After testing, the application can be deployed. Information about the new services and business processes can then be added into the knowledge base so that they can be reused in the future.

In this framework, BPO and its extension can provide an explicit conceptual

model with formal logic-based semantics and can represent properties of, relationships between, and behaviors of business processes and services. The inherent features of ontology bring benefits to the business process model, making the model semantically rich and model components reusable. The business process models are easy to maintain because of the inherent extensibility of ontology. As the extension of BPO is created based on the organization's asset base, which stores the software artifact of the organization, the quality of extracted services should be acceptable for the organization because they have been operating normally in some existing systems.

3.4 Modeling Methods

Given the many offerings of standard services from various software suppliers, organizations can adopt different modeling strategies to satisfy their specific requirements. For example, they might create a new system using a Top-Down strategy which firstly analyzes the business targets and processes, creates process models, and then implement the system. However, some organizations may have a large asset base of existing services available for reuse through years of development. If these services can be fully reused in the creation of the new generation of system, development costs will be reduced. We propose four modeling methods to improve the reuse of specifications and services.

3.4.1 Notations

For more accurate description of the modeling methods, some notations are defined as follows:

- Italic lowercase letters, such as *p* and *x*, are adopted to represent concrete activities, business processes or services.
- An *activity* refers to a real world business task where some sort of business function is carried out. There are two kinds of activities in a business domain:
 - Composite Activity: This represents a business task which contains a series of activities coordinated by some workflow logic.
 - Atomic Activity: This consists of the business logic of a self-contained business task. It can be executed independently and can also be a step in a composite activity.
- A *business process* is an abstraction of an activity, which can be represented by an instance of the `BusinessProcess` class in the extension of BPO. It is a building block for the business process modeling. To represent different kinds of activities, two kinds of business processes will be obtained from the initial requirement analysis:
 - Composite Process: This consists of a collection of logically related business processes that are coordinated according to some workflow logic. It is the abstraction of the relevant composite activity to represent a business task.
 - Atomic Process: This refers to a basic building block for the business process modeling. It is the abstraction of the relevant atomic activity to represent a granular business task.
- A *service* is the abstraction of the software implementing an activity, which can be represented by an instance of `Service` class in the extension of BPO. It encapsulates the logic required to execute one or more self-contained

business functions. There are also two kinds of services:

- **Composite Service:** This refers to an aggregation of services that collectively implement a business task which is composed by a series of activities.
 - **Atomic Service:** This refers to a granular service that implements a specific business function. It does not encompass other services, and can be used independently or participate as a part of a composite service.
- Italic capital letters or abbreviations are adopted to represent the sets of activities, process models and services generated during the modeling procedure. The notations are defined in detail in Table 8.

Table 8. Definitions of abbreviations

Set	Name	Elements
<i>A</i>	Activity	Activities abstracted from the initial specification $A = CA \cup AA$ $A = NA \cup RA$
<i>AA</i>	Atomic Activity	Atomic activities abstracted from the initial specification
<i>APT</i>	Atomic Process Template	Class definitions for the atomic processes abstracted from atomic activities, $APT \subseteq PT$
<i>AS</i>	Atomic Service	Atomic services implementing an atomic business process model
<i>CA</i>	Composite Activity	Composite activities abstracted from the initial specification
<i>CPM</i>	Concrete Process Model	Individual definitions for the concrete process models
<i>CPT</i>	Composite Process Template	Class definitions for the composite processes abstracted from composite activities, $CPT \subseteq PT$
<i>CRP</i>	Class Reusable Process	Process templates that can be reused
<i>CS</i>	Composite Service	Composite services implementing a composite business process model

<i>DRP</i>	Directly Reusable Process	Concrete process models which can be reused without making any changes
<i>NA</i>	New Activity	Activities which have no existing process model
<i>PT</i>	Process Template	Class definitions for the business processes
<i>RA</i>	Reusable Activity	Activities which have relevant process models
<i>RP</i>	Reusable Process	Potential reusable process models (including both process templates and concrete processes) obtained from the knowledge base, $RP = DRP \cup CRP$
<i>S</i>	Service	Services for a business logic

- R_A represents the hierarchical relationship between activities. The relationship can be denoted by the function $\sigma|_{R_A} : R_A \rightarrow CA \times A$, which means an activity can be a component of a composite activity. Similarly, R_P represents the hierarchical relationship between processes and the function $\sigma|_{R_P} : R_P \rightarrow CPT \times PT$ means that a process is a component of a composite process; R_S represents the hierarchical relationship between services and the function $\sigma|_{R_S} : R_S \rightarrow CS \times S$ means that a service is a component of a composite service.

3.4.2 The Modeling Processes

A valid system model can be obtained by following three steps: requirements elicitation, requirements analysis, and requirements validation (Sommerville, 2007). Our methodology focuses mainly on the requirements analysis process, constructing the system models and generating formal specifications. Formal description allows the system models to be evaluated and validated more accurately, either manually or automatically (Liao et al., 2007b).

When using our approach, software developers should start from the

requirements elicitation phase and identify from the application domain the services the system should provide, the required performance of the system, and hardware constraints, etc. They then draft an initial requirement specification using natural languages. This specification should identify the functional and non-functional requirements of the system, organize the unstructured collection of requirements into coherent clusters, and prioritize the requirements.

The knowledge base (the extension of BPO) can be used either by experts or automatically. The experts can use ontology editing tools, such as Protégé (Gennari et al., 2003; Protégé, 2008), to manually maintain the ontology and check the consistency of the definitions. Ontology learning methodologies (Cimiano, 2006) can construct the knowledge base by abstracting concepts and relations from text documents, which can help to extend the knowledge base automatically or semi-automatically. In this study, manual methods will be discussed, and we adopt Protégé to construct the ontologies.

Protégé is a set of open-source ontology design software developed at Stanford Medical Informatics, which can provide knowledge solutions for different areas. Protégé provides a visualized tool for defining OWL classes, properties and individuals, supports reasoners such as RacerPro (the commercial name for a Renamed ABox and Concept Expression Reasoner), and executes queries of SPARQL (Simple Protocol And RDF Query Language). By reasoning with RacerPro, the experts can check the consistency of an OWL ontology and related data descriptions, and identify implicit sub-class relationships induced by the declaration of the ontology. Protégé also provides two kinds of queries. One is its own visual query tool, which supports simple or complex queries of individuals and properties; the other is a SPARQL query panel. As recommended by W3C (2008b), SPARQL

can be used to represent the RDF graph — a set of triples that consists of a subject, a predicate and an object as the basic expression of data stored in OWL-based knowledge base. Compared with the Protégé’s visual query, SPARQL can provide more functions to the query, such as filters, ordering results, handling duplicate solutions, etc. (W3C, 2008b); therefore, we adopt SPARQL as the query language in our study.

From the initial specification, activities and the hierarchical relationship between the activities should be abstracted first. If the target process model and all its activities are new for the organization, the software developers can directly create classes and individuals to represent these new activities abstracted from the requirement analysis. If the organization has developed similar processes and their information have been defined in the extension of BPO (the knowledge base), the software developers can first query the knowledge base to identify these reusable processes, and then design the others as needed. The processing flow is shown in Figure 8.

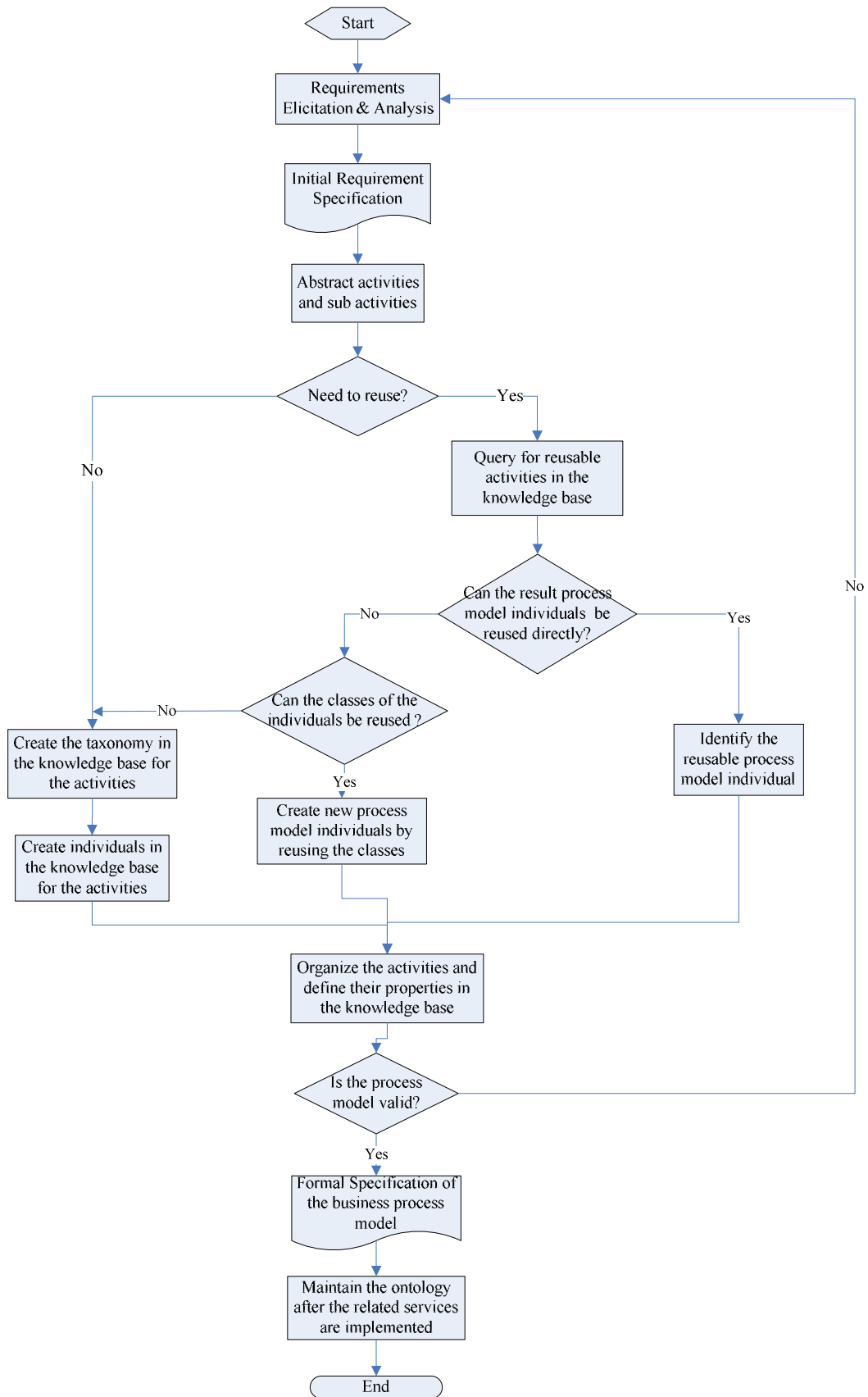


Figure 8. Ontology-based modeling flow

As presented in Section 2.1.4, Erl (2005) proposes three strategies for SOA development: the Top-Down strategy, the Bottom-Up strategy, and the Agile strategy. Based on his strategies, we defined three modeling strategies which have the same names but different contents. The Top-Down strategy focuses on creating an overall business model on the basis of a thorough requirements analysis. The Bottom-Up strategy encourages creating services to best serve the immediate requirements first. The Agile strategy allows for the business-level analysis to occur concurrently with service design and development. After the searching or modeling for the sub-processes, the developers can implement the new services and integrate the reusable services and new services into the final application. Our proposed modeling methods are summarized below:

- **Top-Down Modeling (TDM):** This supports the organizations in creating new process models directly and uses the knowledge base to help to generate the formal requirement specification.
- **Top-Down Modeling based on Reusable Process (TDM-RP):** This supports the organizations in constructing new process models by reusing similar process models already defined in the knowledge base.
- **Bottom-Up Modeling based on Reusable Services (BUM-RS):** This supports the organizations in constructing new process models by reusing some specific services.
- **Agile Modeling based on Reusable Process and Reusable Services (AM-RPRS):** If the specific services are reused by many process models, an analysis of the business target would be helpful to the process of selecting a suitable model. AM-RPRS provides specific queries in the knowledge base and can help the organizations efficiently choose suitable

models.

These four methods can deal with different requirements of organizations, and have their own benefits and weaknesses. Table 9 provides a comparison, with “BP” representing Business Process.

Table 9. Comparison of different modeling methods

	TDM	TDM-RP	BUM-RS	AM-RPRS
Situation	No reusable BP available	Specifies the reusable BP	Specifies the reusable services	Specifies the reusable BP & services
Effort	Creates new BPs directly	Searches for reusable BPs; Creates new BPs	Searches for reusable services; Identifies reusable BPs; Creates new BPs	Searches for reusable BP and services; Identifies reusable BPs; Creates new BPs
Benefit	Formal steps for creating BPs	Reuses existing BP specifications or the logical model of the processes	Reuses existing concrete services and related BPs	More efficient searching of reusable BPs than BUM-RS
Weakness	It is difficult to construct the hierarchy of concepts in a reasonable way at the beginning of development	The organizations need to know what BPs are to be reuse at the beginning of development	The result of searching may involve several BPs	More information about the reusable things is needed

3.4.3 Top-Down Modeling (TDM)

When the goal is to construct a new Business Process Model that does not involve any reusable processes and services, the Top-Down Modeling (TDM) can be used. TDM involves the following steps:

1. **AnalyzeRequirements:** Collect system requirements, organize the

unstructured collection of requirements into coherent clusters, prioritize the requirements and produce an initial requirement specification.

2. **AbstractActivities:** Abstract the activities and the relationships between the activities according to the initial specification.

1) Name the target process model p ;

2) Abstract the activity set A of process p , $A = \{\text{activities} \mid \text{the activities which are described in the initial specification of process } p\}$;

3) Abstract the relationship between the activities in A .

➤ If $x \in A$, $y \in A$, and y is one of the steps in x , then $\langle x, y \rangle \in R_A$, which means x has a sub-activity of y . The activity x is a composite activity, $x \in CA$.

➤ If $x \in A$ and x cannot be divided from the viewpoint of business operation, then x is an atomic activity, denoted as $x \in AA$.

➤ $A = CA \cup AA$.

➤ In TDM, $NA = A$, because $A = NA \cup RA$ and $RA = \emptyset$.

3. **CreateProcessTaxonomy:** Maintain the knowledge base, which includes defining classes to represent the activities in set NA and defining the properties of the classes to represent the relationships between the activities.

1) Define the target process model p as a class in the knowledge base, $p <_c \text{ProcessModel}$;

2) Define the activities in NA as classes in the knowledge base, ensure that

➤ $\forall x \in CA$, define class x' to represent the process model template for activity x , $x' <_c \text{ProcessModel}$, $x' \in CPT$.

➤ $\forall y \in AA$, define class y' to represent the process model template of related

atomic activity $y, y' <_c \text{AtomicProcess}, y' \in \text{APT}$.

➤ $PT = CPT \cup APT$.

3) Define the object properties of the process model classes according to the relationships between the activities.

➤ $\forall \langle x, y \rangle \in R_A$, there are $\langle x', y' \rangle \in R_p$, where x' and y' are the process models representing the activity x and y . The R_p relationship is defined as `hasSubProcess` property in the knowledge base, which means that $\forall \langle x', y' \rangle \in R_p$, there are $\sigma_{(\text{hasSubProcess})} = \langle x', y' \rangle$.

➤ The relationship `isSubProcessOf` in the knowledge base can be defined automatically because it is the inverse property of `hasSubProcess`.

➤ If a composite process x is composed of sub-processes x_i ($i = 1, \dots, n$), the workflow of the sub-processes can be identified by constructing `ProcessFragment` with one of the `FlowPattern`.

4) Configure the properties of all the elements of PT .

4. **CreateProcessIndividuals**: Maintain the knowledge base, which includes instantiating the classes and properties for all the elements in PT ; placing the individuals into the CPM , which means that the individuals can represent the concrete business processes models for the activities; ensuring that all the activities in NA have been modeled and placed in CPM .

5. **OrganizeProcess**: Maintain the knowledge base, which includes identifying the workflow of the concrete process models (the elements in CPM and DRP , $DRP = \emptyset$ in TDM) by defining the flow pattern.

6. **CheckConsistency**: Check the consistency of the knowledge base.

7. **EvaluateModel**: Evaluate and validate the process model; ensure that all the

activities in A have been modeled as process models in PT .

8. **Implementation:** Design or purchase services to implement the concrete business processes models in CPM . Place the atomic services into the set AS and the composite services into the set CS . $S = CS \cup AS$. The relationships between services are defined in set R_S .

9. **CreateServiceTaxonomy:** Maintain the knowledge base; define classes to represent the services.

- $\forall s \in CS, s <_C \text{CompositeService};$
- $\forall s \in AS, s <_C \text{AtomicService}.$
- If $R_S \neq \emptyset$, define the relationships `hasSubService` and `isSubServiceOf` according to the elements in R_S .
- Configure the properties of all the services in S .

10. **CreateServiceIndividuals:** Maintain the knowledge base; instantiate the classes and properties for all the services in S .

11. **MappingProcess:** Maintain the knowledge base; build up the properties between the process models and the related services.

3.4.4 Top-Down Modeling based on Reusable Processes

(TDM-RP)

By reusing an existing business process model, the developers can create a new business process and reuse the related services. The steps include:

1. **AnalyzeRequirements;**
2. **AbstractActivities;**
3. **SearchForProcesses:** Query the knowledge base for the reusable process

models for the activities in A . Place the results, a group of OWL individuals representing potential reusable processes, into the reusable process set RP , and move the relevant activities and sub-activities from A to the reusable activity set RA . $NA = A - RA$. If a composite process belongs to RP , its component processes also belong to RP .

According to the attributes of `BusinessProcess` and related classes (shown earlier in Table 5 and 6), the following queries can be done with SPARQL:

- Assume that the activities and process models are named according to a naming convention of the organization, which means that software developers can obtain the name of the process model on the basis of the name of the activity. Then the query can be conducted with the `processName` attribute of process models. The searching with SPARQL can be as follows:

SearchByProcessName:

```
select ?process
where { ?process BPO:processName ?processName.
      Filter regex (?processName, "PROCESSNAME"). }
```

- If one of the component processes of a composite process model is known, the query can be as follows:

SearchByComponentProcesses:

```
select ?process
where { ?process BPO: hasSubProcess ?componentProcess.
      ?componentProcess BPO:processName ?componentProcessName.
      Filter regex (?componentProcessName,
      "COMPONENTPROCESSNAME"). }
```

- The software developers can also list all the process models which are

performed by a specific category of services as follows:

SearchByCategory:

```
select ?process
where {?process BPO:isPerformedBy ?service.
      ?service AutoPO:serviceCategory ?category.
      ?category AutoPO:categoryName ?name
      Filter regex (?name, "CATEGORYNAME").
      }
```

Because SPARQL provides flexible descriptions for integrated queries, these queries can be combined to improve the accuracy of the query results.

4. **ReviewSearchingResult:** Identify reusable processes by reviewing the query results in *RP*, which is a group of OWL individuals representing potential reusable concrete process models. This review will detect whether the performance (such as time limits) of the result processes can satisfy the requirements. This step can also be conducted by SPARQL queries. For example, the query for process time requirement can be:

SearchByTimeLimit:

```
select ?process
where {?process BPO:processTimeLimit ?timelimit.
      Filter regex (?timelimit < SPECIFIC TIMELIMITVALUE).
      }
```

- If an element of *RP* (a concrete process model) can be reused without making any changes, move that element from *RP* to *DRP* for future usage;
- If an element of *RP* cannot be reused directly, determine whether the class or the super-class of the individual can be reused, then move the reusable classes (process model template) to *CRP*;

- If neither the element x nor its class can be reused, determine whether a super-class can be inserted to generate a more abstract template. If it works, move the class of the element x to CPT and move the relevant activity of element x from RA to NA .
- If neither the class for the element x nor its super-classes can be reused, delete it from RP and move the relevant activities of element x from RA to NA .
- $RP = DRP \cup CRP$.
- $A = NA \cup RA$.
- The processes in RP should be consistent with the activities in RA .

5. Step 3 and step 4 can be repeated to obtain all the reusable processes and services.

6. **MaintainProcessTaxonomy:** Maintain the knowledge base, which includes defining new classes to represent the logical model of the business and defining the properties of the classes to represent their relationships and the workflow of the business logic.

1) If $CPT \neq \emptyset$, which means that the current process template needs to be modified so that it can be more common, define classes to construct the abstract model template.

- $\forall x \in CPT$, define a class y as its direct super-concept to represent the kind of activities of the element x from a more abstract level.
- $y <_c \text{ProcessModel}$
- If $x < z$, the new relationship between the concepts is $x < y < z$.
- Configure the `isOrganizedBy` property of the class y according to the

relevant property definitions in x .

- After creating the classes, set CPT as \emptyset so that it can be used in the next step.

2) Define the activities in NA as classes in the knowledge base, ensure that

- $\forall x \in CA$, define a class x' to represent the process model template for the activity x , $x' <_C \text{ProcessModel}$, $x' \in CPT$.
- $\forall y \in AA$, define a class y' to represent the process model template of related atomic activity y , $y' <_C \text{AtomicProcess}$, $y' \in APT$.
- $PT = CPT \cup APT \cup CRP$.

3) According to the relationships between the activities, define new object properties for the process model classes in PT .

- $\forall \langle x, y \rangle \in R_A$, there are $\langle x', y' \rangle \in R_p$, where x' and y' are the process models representing the activity x and y . The R_p relationship is defined as `hasSubProcess` property in the knowledge base, which means that $\forall \langle x', y' \rangle \in R_p$, there are $\sigma_{(\text{hasSubProcess})} = \langle x', y' \rangle$.
- The relationship `isSubProcessOf` in the knowledge base can be defined automatically because it is the inverse property of `hasSubProcess`.
- If a composite process x is composed of sub-processes x_i ($i = 1, \dots, n$), the workflow of the sub-processes can be identified by using one of the `FlowPatterns` to construct `ProcessFragment`. Some of these relationships may be inherited from the relations of their super-concepts.

4) Configure the properties of all the elements of PT and the classes of the elements in DRP .

7. Follow steps 4 to 11 of TDM.

3.4.5 Bottom-Up Modeling based on Reusable Services

(BUM-RS)

By reusing one or more existing services, developers can identify the reusable business process, reuse those related services, and create required new business processes. The Bottom-Up strategy follows these steps:

1. **AnalyzeRequirements;**

2. **AbstractActivities;**

3. **IdentifyReusableServices:** Identify the services that need to be reused in the new system according to the initial specification; obtain the information of the services such as service name and service category.

4. **SearchForServices:** query the knowledge base for the reusable processes which were implemented by the identified services. Place the query results, a group of process individuals which is performed by the specific services, into the reusable process set RP . If a composite process belongs to RP , its component process models also belong to RP . Move the relevant activities and sub-activities of the processes in RP from the activity set A to RA . $NA = A - RA$. The query with SPARQL can be:

SearchByServiceName:

```
select ?process
where {?process BPO:isPerformedBy ?service.
      ?service BPO:serviceName ?serviceName.
      Filter regex (?serviceName, "SERVICENAME").
      # If the name of the reusable service is known }
```

Then further searching can help to identify the composite processes which indirectly use the identified services.

```

SearchCPByServiceName:

select ?compositeProcess ?process
where { ?process BPO:isPerformedBy ?service.
      ?service BPO:serviceName ?serviceName.
      Filter regex (?serviceName, "SERVICENAME").
      # If the name of the reusable service is known
      ?process BPO:isSubProcessOf ?compositeProcess
}

```

5. ReviewSearchingResult;

6. Step 4 and step 5 can be repeated to obtain all the reusable processes and services.

7. MaintainProcessTaxonomy;

8. Follow steps 4 to 11 of TDM.

3.4.6 Agile Modeling based on Reusable Process and Reusable Services (AM-RPRS)

Sometimes, a common service may be used by several business processes. For example, “Voice” service can be used in both “Phone” and “Navigation” applications (Krüger et al., 2004). Improvements in software reusability means there will be more business processes sharing common services. Under this situation, the BUM-RS method may not be as efficient as hoped. However, if the developers have more information about the target process model, the searching process could be more manageable and more accurate. The steps of AM-RPRS are:

1. AnalyzeRequirements;

2. AbstractActivities;

3. IdentifyReusableServices;

4. **SearchProcessServices:** Search the knowledge base for the reusable processes which are implemented by the identified services and the information for the activities. Place the searching results, a group of process individuals which is performed by the specific services, into *RP*, and move the relevant activities and sub-activities from set *A* to reusable activity set *RA*. $NA = A - RA$. If a composite process belongs to *RP*, its component processes also belong to *RP*.

For example, if the developers know the name of the common service to be reused and the category of the other services of the target process model, then the searching with SPARQL can be:

```

SearchByServiceAndProcess:

select ?process
where { ?process BPO:isPerformedBy ?service.
      ?service BPO:serviceName ?serviceName.
      Filter regex (?serviceName, "SERVICENAME").
      # If the name of the common service is known
      [ ?process BPO:PROCESS-PROPERTY ?knownProcessProperty.
        Filter regex(?knowProcessProperty, "VALUE").
      ]
      # If any of the process properties have been identified
}

```

5. **ReviewSearchingResult;**

6. Step 4 and step 5 can be conducted repeatedly to obtain the reusable processes and services.

7. **MaintainProcessTaxonomy;**

8. Follow steps 4 to 11 of TDM.

This method simultaneously supports both the service searching and process model searching. From service searching, the developers can obtain several process

models that reuse the required services. From process model searching, the developers can identify a suitable reusable process model. This method is a meet-in-the-middle approach.

The steps of the four methods are summarized in Table 10:

Table 10. Comparison of the four modeling methods

TDM	TDM-RP	BUM-RS	AM-RPRS
1. Analyze-Requirements	1. Analyze-Requirements	1. Analyze-Requirements	1. Analyze-Requirements
2. AbstractActivities	2. AbstractActivities	2. AbstractActivities	2. AbstractActivities
		3. Identify-ReusableServices	3. Identify-Reusable-Services
	3. SearchForProcesses	4. SearchForServices	4. SearchProcess-Services
	4. Review-SearchingResult	5. Review-SearchingResult	5. Review-SearchingResult
3. Create-ProcessTaxonomy	5. Maintain-ProcessTaxonomy	6. Maintain-ProcessTaxonomy	6. Maintain-ProcessTaxonomy
4. Create-ProcessIndividuals	6. Create-ProcessIndividuals	7. Create-ProcessIndividuals	7. Create-ProcessIndividuals
5. OrganizeProcess	7. OrganizeProcess	8. OrganizeProcess	8. OrganizeProcess
6. CheckConsistency	8. CheckConsistency	9. CheckConsistency	9. CheckConsistency
7. EvaluateModel	9. EvaluateModel	10. EvaluateModel	10. EvaluateModel
8. Implementation	10. Implementation	11. Implementation	11. Implementation
9. Create-ServiceTaxonomy	11. Create-ServiceTaxonomy	12. Create-ServiceTaxonomy	12. Create-ServiceTaxonomy
10. Create-ServiceIndividuals	12. Create-ServiceIndividuals	13. Create-ServiceIndividuals	13. Create-ServiceIndividuals
11. MappingProcess	13. MappingProcess	14. MappingProcess	14. MappingProcess

From Table 10, we can see that TDM offers the basic steps for model creation, and the other three methods apply different kinds of searching methods. The searching methods can help to identify different reusable items from the knowledge base, which can accelerate the modeling procedure.

3.5 Summary

In this chapter, we presented the mathematical definitions for ontology and its relevant concepts: core ontology, axiom, lexicon and knowledge base. On the basis of these definitions, we proposed the framework of BPO (Business Process Ontology), and the four methods for SOA modeling: TDM, TDM-RP, BUM-RS and AM-RPRS. These four methods illustrate not only the SOA modeling process but also the extension and usage of the knowledge base. We also proposed an Ontology-based Business Process Modeling and Developing Framework (OBPMDF) to illustrate the developing framework of SOA applications with BPO.

During the construction and use of the knowledge base, the developers need to evaluate and validate it, since its contents should be consistent and satisfy real-world applications. Different evaluation approaches can be used for different purposes.

Chapter 4 Validation and Verification

In this chapter, we apply our ontology-based SOA modeling methodology to the automotive software development to validate our framework and modeling methods. A knowledge base AutoPO is constructed by extending the core ontology BPO, and a series of case studies are used to simulate the implementation of the modeling methods. After that, a comparison of our modeling method and other automotive software modeling methods is presented. Besides the validation of our methodology, this chapter also proposes a set of quality attributes for SOA models and shows how our methodology can support these quality attributes.

4.1 Validation of Modeling Methodology

Validation concerns that the right product was built⁴. In software development, it is the demonstration that the software implements each of the software requirements correctly and completely. Here it is a quality process to evaluate whether the methodology complies with our objectives. This validation will be conducted by constructing a knowledge base for automotive software modeling and using the knowledge base to build different SOA models.

Case study was adopted in our research for this validation. Case Study is one of the most common methods for the research in business and SE; it is associated with process evaluation and can satisfy the three tenets of the qualitative method:

⁴ <http://www.critech.com/vv.htm>

describing, understanding, and explaining (Yin, 2003). Therefore we adopt case studies to demonstrate the usage of our modeling methodology here. Five cases for different requirements and modeling methods are presented in this chapter.

4.1.1 Extension of BPO for Automotive Software

Modeling

The core ontology BPO described in Section 3.2 defines the abstract concepts and relationships in business process modeling. It can be used in different domains. When applied to a specific domain, for example the automotive domain, the core ontology BPO needs to be extended. This extension adds domain specific knowledge to the ontology, making the knowledge well structured.

The automotive process modeling ontology that we constructed in this work extends BPO based on AUTOSAR, and is therefore named “AutoPO”. AutoPO is created by importing the core ontology BPO and adding new classes to represent automotive software features. After importing, the concepts in BPO become the basic concepts in AutoPO, for example:

- **BusinessProcess** represents the abstraction of all kinds of business processes of an organization. In the automotive domain, it can represent the super-class of all the processes in a car.
- **AtomicProcess** is the process whose functions should be conducted as a whole and cannot be subdivided. In the automotive domain, it can represent the super-class of a single task in a car, for example, monitoring a specific sensor.
- **CompositeProcess** refers to processes that are composed of one or more

business processes, which may be atomic processes or other composite processes. In the automotive domain, it can represent the super-class of a task, for example, wiping the window.

- **Service** represents the super-class of all the available services in an organization's asset base. If used in the automotive domain, it represents any available service in cars.
- **AtomicService** is a service that can fully implement a business process and during modeling is regarded as a black box. In the automotive domain, it can represent an AUTOSAR Software Component, which is a service that can use a single AUTOSAR ECU to fully implement a business process. An AUTOSAR Software Component can be a small, reusable function (such as a filter) or a large, encapsulated automotive function (such as a sensor/actuator Software Component) (AUTOSAR, 2006a).
- **CompositeService** represents a service that is composed of several other services, which can be atomic services and other composite services. In the automotive domain, it can represent the automotive system and subsystems consisting of connected AUTOSAR Software Components.

In extending BPO to the automotive domain, we have also added classes representing service categories and service communication features. This is a matter of some complexity because different kinds of automotive software may have totally different requirements and features. For example, the wiper/washer system and the multimedia control panel are different systems, with the former being a body comfort control system and the latter being a multimedia system. This means that the former connects to sensors and actuators and the latter connects to multimedia drivers. Their formats for transferring messages and data format also differ. Bearing

in mind that such variations are multiplied many times and in many ways throughout an automobile, to precisely identify and describe services in terms of categories and communication features would provide a great deal of useful information for use in software integration.

The extension for category classification and service ports definition is defined in the following sections.

4.1.1.1 Automotive Software Categories

There are seven domains of software in a car: safety, power train, chassis, multimedia/telematics, body comfort, man-machine interface, and infrastructure software (Broy et al., 2007a; Pretschner et al., 2007). Some of these domains are vehicle-centric, such as chassis and power train, and some are passenger-centric, such as body comfort. The safety domain is concerned with the safety features in automobile which may include vehicle condition recognition, belt pretensioners, and airbags (Pelz et al., 2005). The power train domain contains the motor control-related functions, such as the brains of the gearbox or the automatic transmission. The chassis domain is concerned with functions such as brake assistance, distance control, and drive by wire. The multimedia/ telematics domain contains the infotainment functions, GPS navigation, Internet connection, and information-related functions. The body comfort domain concerns the functions of body electronics, such as the power-adjustable external mirrors, climate control, and keyless entry. The human-machine interface in an automobile provides a solution for piloting the vast number of functions without a large number of buttons and controls. Infrastructure software concerns the management of the IT systems in automobiles, such as software for diagnostics and application updates.

To classify the automotive services and business processes, new classes are added to the core ontology for representing the categories. The AutomotiveCategory is the root class for the category ontology, and the seven automotive software domains are defined as the sub-classes of the AutomotiveCategory.

The hierarchy of the AutomotiveCategory and its sub-classes is shown in Figure 9. This extension is related to the classes in BPO by the property serviceCategory. Eight classes are defined to represent the categories, as described in Table 11.

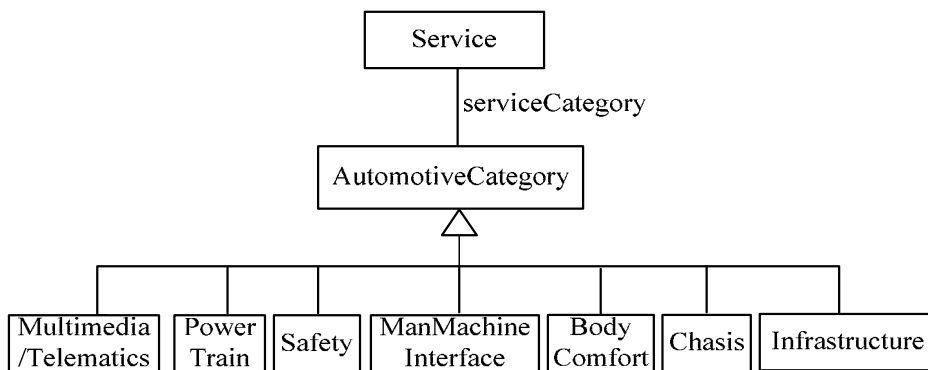


Figure 9. Extension of BPO for automotive categories

Table 11. New class definitions in AutoPO

Concepts	Super-Concepts	Comments
AutomotiveCategory	OWL:Thing	Represents the abstraction of all the categories of the automotive software.
BodyComfort	AutoPO:AutomotiveCategory	Represents the category of body comfort software
Chasis	AutoPO:AutomotiveCategory	Represents the category of chassis software
Infrastructure	AutoPO:AutomotiveCategory	Represents the category of IT system management software
ManMachineInterface	AutoPO:AutomotiveCategory	Represents the category of human-machine interfaces
Multimedia/ Telematics	AutoPO:AutomotiveCategory	Represents the category of multimedia or telematic software
PowerTrain	AutoPO:AutomotiveCategory	Represents the category of power train software
Safety	AutoPO:AutomotiveCategory	Represents the category of safety software

In this extension, a new property `serviceCategory` is defined in the `Service` class. The domain of the property `serviceCategory` is `BPO:Service` and the range is `AutoPO:AutomotiveCategory`. This property represents the category of the services. Generally, an atomic service should belong to one of the categories because it represents an AUTOSAR Software Component which should belong to a specific category. A composite service may belong to a single category or a union of categories because it may cover the functions in multiple domains. For example, besides a lock manager (body comfort category), a central locking system may also include a seat adjustment system (body comfort category), lighting system (body comfort category), crash sensing system (safety category), and radio tuner (multimedia category).

Adding the categories into the knowledge base can make the knowledge management more efficient. For example, the developers query all the existing body-comfort services or process models that relate to those services; they can add new axioms to constrain the features of services in a specific category. This category classification is an initial taxonomy of the automotive software. It can be further extended with detailed categories (sub-classes of the seven categories) in practical usage.

There is no property defined to relate the `BusinessProcess` class and the `AutomotiveCategory` class. The reasons are as follow:

- A composite process model represents a workflow, which usually involves multiple services of different categories. Therefore, it is difficult to classify the composite process models into one specific category. If most of the models are identified as belonging to a union of categories, the query result for a specific category of process models would be complex.

- Generally a process model and its related services should belong to the same category. If both the Process class and the Service class have their own category properties, inconsistencies may occur.
- Because services are the implementation of related process models, we can obtain a process model's category information by querying the category of the services which perform the process model.

4.1.1.2 Ports

In this study, we adopt the AUTOSAR software components as the minimal units of service. In BPO's definition, the AtomicService class represents an AUTOSAR software component and the CompositeService class represents the composition of multiple AUTOSAR software components.

As described in Section 2.4, the AUTOSAR software components and their compositions belong to the highest (most abstract) description level in AUTOSAR, the application layer. This layer is also called Virtual Functional bus level and it is here that the AUTOSAR software components are treated as independent software units communicating with each other by ports (AUTOSAR, 2006b). Because the service-related classes and instances in BPO are used to provide information about the existing services for the business process modeling, the implementation of the services will not be considered in our modeling method. Therefore, in the extension of BPO, the AUTOSAR software components will be treated as a black box service with ports for communication.

AUTOSAR software components' ports are well defined interaction points which define the possible kinds of communication between the components (AUTOSAR, 2006b). There are two kinds of ports for the components, R-port

(Require-port) and P-port (Provide-port). An R-port requires certain services or data, while a P-port provides those services or data. There are two kinds of interfaces for the ports and they define the information exchanged between the ports; one is sender/receiver interface and the other is client/server interface. A sender/receiver interface can describe data elements or model groups to be sent and received, and a client/server interface declares operations that a client can invoke on a server. Because ports are specific to the automotive software components, we define classes to represent ports in AutoPO.

Classes and relations are defined to represent the ports and port interfaces. The framework of this extension and the relation with the class in BPO is shown in Figure 10.

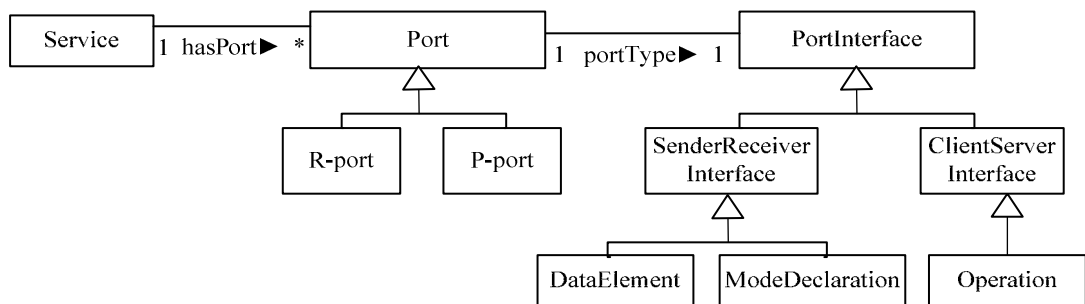


Figure 10. Extension of BPO for automotive software components' ports

The classes in this extension include:

- **Port** representing the abstraction of all kinds of well defined connection points of the services. Ports can represent the dependencies between the services.
- **R-port** representing the require-port, which describes what a service requires.
- **P-port** representing the provide-port, which describes what a service can provide.

- **PortInterface** representing the abstraction of all types of content that are required or provided by the respective ports.
- **SenderReceiverInterface** representing the kinds of data to be sent and received. There are two kinds of data defined in AUTOSAR, data elements and the mode of switches in a car, which are defined as sub-classes of SenderReceiverInterface.
- **ClientServerInterface** representing the abstraction of operations that a client can invoke on a server.
- **DataElement** representing the types of data elements to be sent and received.
- **ModeDeclaration** representing the mode of switches that can be sent and received.
- **Operation** representing the abstraction of operations.

The hierarchical relationships \leq_C between the concepts are summarized in

Table 12.

Table 12. Hierarchical relationships between the concepts in AutoPO

Concepts	Direct Super-Concept	Super-Concepts	Direct Sub-Concept	Sub-Concepts
ClientServer-Interface	PortInterface	PortInterface	Operation	Operation
DataElement	SenderReceiver-Interface	PortInterface, SenderReceiver-Interface		
Mode-Declaration	SenderReceiver-Interface	PortInterface, SenderReceiver-Interface		
Operation	ClientServer-Interface	PortInterface, ClientServer-Interface		

Port	OWL: Thing		R-port, P-port	R-port, P-port
PortInterface	OWL: Thing		SenderReceiver- Interface, ClientServer- Interface	SenderReceiver- Interface, ClientServer- Interface, DataElement, ModeDeclaration, Operation
P-port	Port	Port		
R-port	Port	Port		
SenderReceiver- Interface	PortInterface	PortInterface	DataElement, ModeDeclaration	DataElement, ModeDeclaration

Because the ports will be defined according to the implementation of the services, only the essential object properties are defined to represent the relationship between the concepts in the extension of BPO. These main object properties are presented in Table 13.

Table 13. Main object properties in AutoPO

Relation Identifier	σ_R	Explanation
hasPort	< BPO:Service, AutoPO:Port >	States the ports of a service. This property also connects the classes in BPO and its extension. Each service can have ports. Some of them may be P-ports and the others are R-ports. For composite services, the ports of inner components will not be considered because the composite service should be used as a whole.
portType	< AutoPO:Port, AutoPO:PortInterface>	States the communication type of the ports. Identifies that the communication paradigm is data driven (send/receiver) or operation oriented (client/server).
service- Category	< BPO:Service, AutoPO:Category>	Identifies the category that a service belongs to. This property connects the classes in BPO and its extension. An atomic service should belong to a category; a composite service can belong to a category or a union of categories.

The extension of ports has some overlap with the property serviceIO in BPO.

This is because this extension is specific to automotive software and the serviceIO property is for all services. To resolve this overlap and to avoid any inconsistency, in the usage of AutoPO, the developer can just use hasPort property to describe the inputs and outputs of services, and ignore the property serviceIO.

These extensions allow the main structure of AutoPO to be developed. It then becomes possible to add new classes and individuals for the concrete process models and services to the AutoPO during software development and to use AutoPO as the knowledge base for automotive process modeling. The complete framework of AutoPO is shown in Appendix B. The properties in BPO and its extension can be used to describe business processes and services in the automotive domain. For example, the processTimeLimit and processIO properties and related classes can support the description of the timeliness and communication requirements of processes; the preCondition and postCondition properties can help to describe the resource and features that the environment should satisfy. The properties can be instantiated in the concrete business process individual, and this can help the developers to identify the requirements and conflicts of the business processes in the initial phase of development. If the isOrganizedBy property of a concrete composite business process is defined as an individual of Parallel class, the component processes of that business process should be conducted concurrently. The related services should also be able to run synchronously. Applying this rule allows us to select from all the services the most suitable service to implement a specific business process. In practice, the developers can instantiate the properties of a business process or service individual according to the requirements. The contents of the properties can be identified as needed, which make the modeling procedure more flexible.

4.1.2 Case Studies

In this section we provide a series of case studies to illustrate the application of AutoPO and explain how to reuse the logical and concrete models for the business processes.

There are five case studies about the procedure for modeling the automobile wiper/washer systems. The logic of the case studies is described here. In the first case study, we assume that the developers need to design and develop an automatic wiper/washer system to control the work of the windshield wipers and washers. The key function of an automatic wiper/washer system is to receive driver's commands, sense the environment, and control the action of wipers and washers. The environmental information can be detected with a rain sensor and analyzed with a wiping evaluator. The sensor and evaluator constitute a rain sensing system. The rain sensing system will pass the requirement for wiping the front windshield to the core function of the wiper/washer system, WWManager (Wiper/Washer Manager). The WWManager will determine whether to send a start command to the wiper service according to the current status of the car and the wipers. We can use a service EnableDisableWiperWasher to detect the status of the car, such as the engine hood state. The washers of the car are connected to a Washer Fluid Tank, which contains the washer fluid and reports the level of fluid to the WWManager. In this case study, we will use TDM to create a "WiperWasherSystem" ontology and related ontologies.

In the automotive domain, systems are usually built in increments, rather than from scratch. A new car series inherits most functionality from existing cars, with various adaptations, extensions, or innovations. Therefore, in the second case study, we will adopt TDM-RP to create a new wiper/washer system for a new generation of

car by reusing the models generated in Case 1. The process model generated in Case 1 provided wiper and washer functions only for the windshield. The new system will provide new functions for the rear window. The requirements of the windshield wiper/washer are the same as Case 1, and the driver can start and stop the rear window's wiper and washer with a switch.

Cases 1 and 2 illustrate the steps that are common to all four modeling methods. In Cases 3 and 4, we will focus on the additional steps in BUM-RS and AM-RPRS. Case 3 will assume that the developers need to reuse service "RainSensingService_V1" in a new type of car and will use BUM-RS to construct a new system on the basis of the process model that relates to that particular service. This case will raise a problem associated with using BUM-RS when a common service may be a component service for several processes, as it is not unusual with OEMs with large service and business process asset bases to store their existing work. If the identified service is in fact a common service, BUM-RS may find several candidate process models, and if there are too many potential reusable processes, developers may find it difficult to determine which process is suitable for reuse. In Case 4, then, we simulate this scenario by assuming that "RainSensingService_V1" is a common service for several processes and then use AM-RPRS to obtain a more precise search result.

Finally, in Case 5 we assume that the developers plan to design a new car which can clean the front lamps automatically. We will use this case study to simulate how to select a suitable modeling method to create a new model.

4.1.2.1 Case 1: Creating A New Model for Wiper Washer System with TDM

As described above, Case 1 focuses on developing an automatic wiper/washer system. The system can start and stop the wiper/washer for the front window by driver's commands or sensing the environment. A rain sensor is adopted for this target. The environment information is analyzed with a wiping evaluator and the central of the system is a WWManager (Wiper/Washer Manager). We use a service EnableDisableWiperWasher to detect the status of the car, which include the status of the wiper/washer switch.

First, the activities of a wiper/washer process can be abstracted as follows:

- The target process $p = \text{WiperWasherSystem}$;
- The process activity set $A = \{ \text{AutoWipingEvaluator}, \text{EnableDisableWiperWasher}, \text{RainSensing}, \text{RainSensor}, \text{Washer}, \text{WasherTank}, \text{Wiper}, \text{WiperWasherSystem}, \text{WWManager} \}$, where
 - The composite activity set $CA = \{ \text{RainSensing}, \text{WiperWasherSystem} \}$, and
 - The atomic activity set $AA = \{ \text{AutoWipingEvaluator}, \text{EnableDisableWiperWasher}, \text{RainSensor}, \text{Washer}, \text{WasherTank}, \text{Wiper}, \text{WWManager} \}$;

Next, we create the process taxonomy in AutoPO. The activities in CA can be defined as the sub-classes of ProcessModel and the activities in AA can be defined as the sub-classes of AtomicProcess. The main taxonomy of the wiper/washer ontology is shown in Figure 11.

These classes and their relationships create a reusable template for the process

models. This template can be regarded as a logical model which describes the functionalities and components of a wiper/washer system. This logical model can be refined with sub-classes and individuals. The sub-classes can represent a more detailed logical model and the individuals can represent the concrete process models.

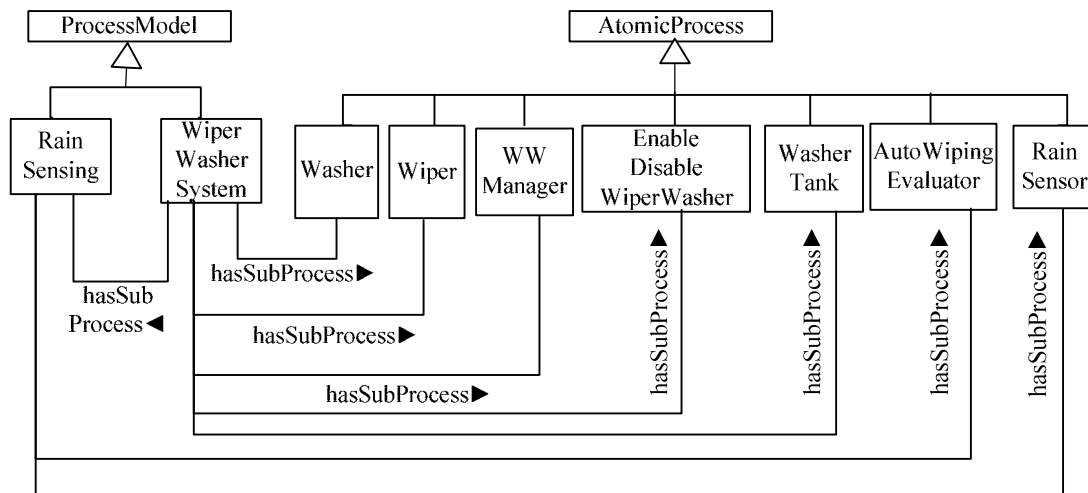


Figure 11. Wiper/washer process model

As shown in Figure 11, the logical model “WiperWasherSystem” consists of:

- **WiperWasherSystem:** The target model representing the general logic of a vehicle wiper/washer system.
- **RainSensing:** A sensing system to monitor the environment and pass the wiping requirement to the WWManager.
- **RainSensor:** A sensor to detect rain. The rain information will be evaluated by the AutoWipingEvaluator.
- **AutoWipingEvaluator:** An analyzer to calculate the requirement for windshield wiping.
- **WWManager:** The kernel of a WiperWasherSystem, controlling wipers or washers according to the drivers’ demands and the environmental signals.
- **EnableDisableWiperWasher:** A component that monitors the vehicle

status and manages the enabling of wiper/washer functionality. It will pass a “disable” signal to the WWManager if the vehicle status is not suitable for wiping/washing action.

- **WasherTank:** A component that monitors the level of the washer fluid containers. It will pass the status of the washer fluid container to the WWManager.
- **Wiper:** An actuator system representing all the functionality which is needed for wiping.
- **Washer:** An actuator system representing all the functionality which is needed for washing.

These classes are organized with the sequence and concurrent flow patterns. Figure 12 shows the sequence graph of the workflow to start and stop the wipers automatically. The EnableDisableWiperWasher and WasherTank processes work concurrently with the WWManager and report the status of the devices to the WWManager. The washer can work concurrently with wiper and the washing process starts when the WWManager receives the drivers’ demands for washing.

The logical model of the wiper/washer system (as shown in Figure 11) is instantiated and shown in Figure 13. This instantiation is an implementation of the logical model for a specific environment. We call it the concrete process model. Detailed requirements, such as the wipe speed limit and wash time-out, will be identified in the concrete process model so that it can reflect the specific design of a specific type of cars. The instantiation of the object properties and data type properties of the individuals can reflect these requirements. For simplicity, Figure 13 shows only part of the individuals of the concrete wiper/washer system process model.

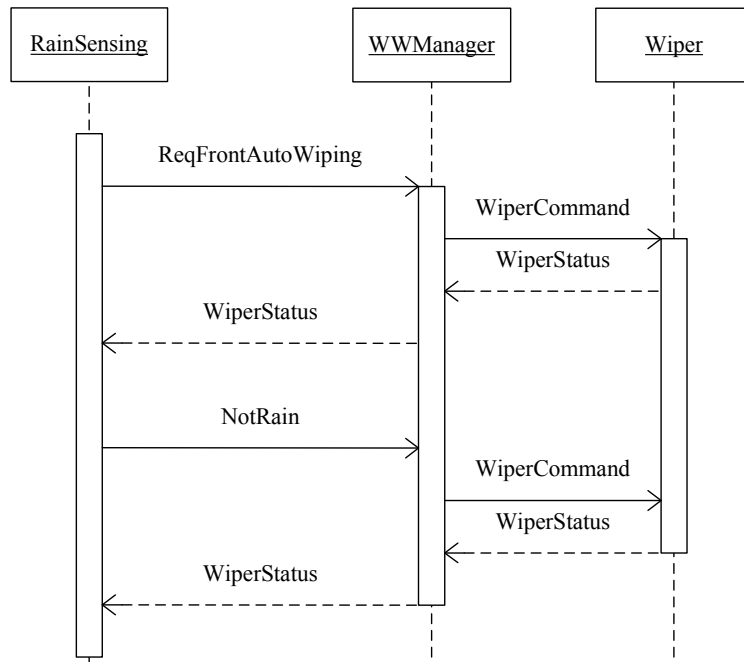


Figure 12. Sequence graph of a wiping process

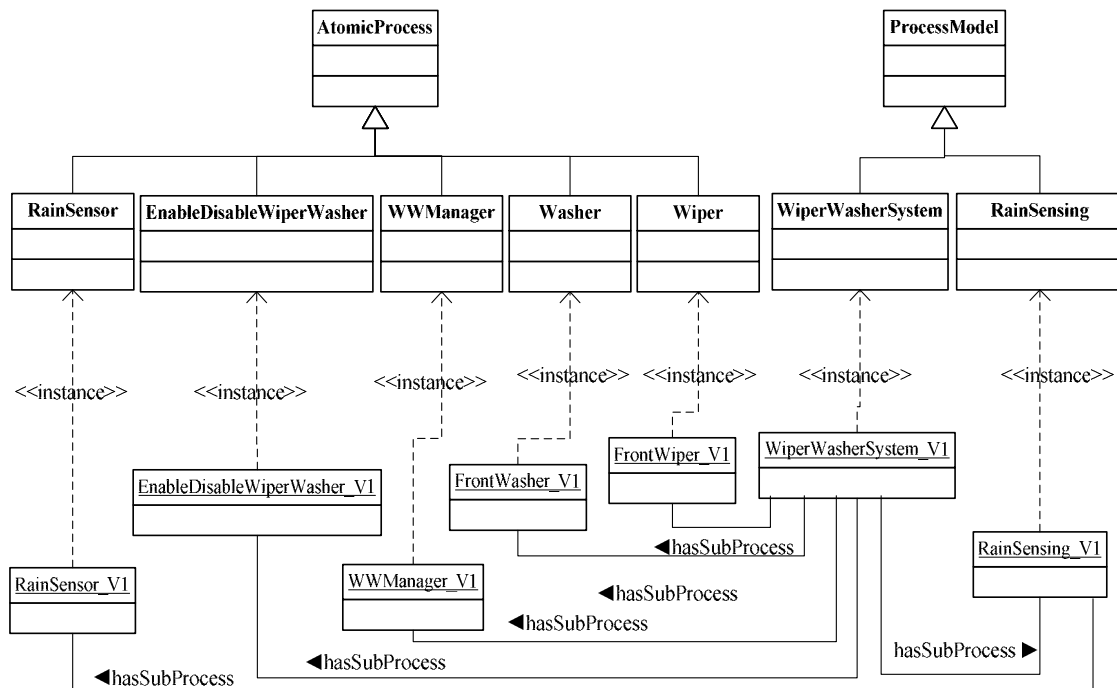


Figure 13. Business Process model: WiperWasherSystem_V1

Our method focuses on the business process modeling and reuse. Using the step CreateProcessTaxonomy, developers can create logical models, as shown in Figure 11. Using the step CreateProcessIndividuals, developers can refine the logical models into concrete process models, as shown in Figure 13. Logical models (classes) can serve as templates for the specifications of the business process models. The individuals describe concrete process models in detail. One class can be instantiated into several individuals. Each individual represents a concrete process model with specific performance requirements. Both the logical models and the concrete models can be reused in the future.

To design and implement the concrete atomic process models in detail, developers can adopt MATLAB® family tools and UML-based methods. These tools allow the implementation of the atomic process models to be described with a series of graphs which represent the internal workflow of the models. Then, services can be developed according to the detailed design. The implementation should be consistent with the AUTOSAR standard. After the implementation of the process models for “WiperWasherSystem_V1”, the information of the related services can be added into the AutoPO. Composite services will be defined as the individuals of CompositeService class, and the atomic services will be defined as the individuals of AtomicService class. The category and the port information of the services should be consistent with the AUTOSAR standards and can also be assigned in AutoPO. For example, Table 14 shows some of the services and their properties.

Table 14. A partial list for properties of concrete services

serviceName	perform	has-SubService	service-Category	R-Port	P-Port
RainSensing-Service_V1	RainSensing_V1	RainSensor-Service_V1, AutoWiping-Evaluator-Service_V1	Body-Comfort	StaFront-WiperFor-RainSensor, Req-RainSensing	ReqFront-AutoWiping, NotRain
WiperService_V1	Wiper_V1	NA	Body-Comfort	Command, Positioning	Status
WWManager-Service_V1	WWManager_V1	NA	Body-Comfort	ReqFront-AutoWiping, NotRain, FrontWiper-Status	FrontWiper-Command; FrontWiper-Positioning

4.1.2.2 Case 2: Creating A New Model by Reusing

Wiper/WasherSystem_V1 with TDM-RP

In Section 4.1.2.1, we created a logical process model WiperWasherSystem and a concrete process model “WiperWasherSystem_V1”. Case 2 demonstrates how to reuse it to construct a new model. In Case 2, we will create a new wiper/washer system which provide wiper/washer functions for both the front and rear windows. This new system will reuse the wiper/washer functions for the front window in “WiperWasherSystem_V1” model, and provide new functions for the rear window.

We will adopt TDM-RP to create a new wiper/washer system in this section. This new system can be considered as an update of the “WiperWasherSystem_V1”. Thus we call the new system “WiperWasherSystem_V2”.

After analyzing the requirements for Case 2 as described at the beginning of Section 4.1.2, we can abstract the following activities:

- The target process $p = \text{WiperWasherSystem}$;
- The process activity set $A = \{ \text{AutoWipingEvaluator}, \text{EnableDisableWiperWasher}, \text{FrontWiper}, \text{FrontWasher}, \text{RainSensing}, \text{RainSensor}, \text{RearWiper}, \text{RearWasher}, \text{WasherTank}, \text{WiperWasherSystem}, \text{WWManager} \}$, where
 - The composite activity set $CA = \{ \text{RainSensing}, \text{WiperWasherSystem} \}$, and
 - The atomic activity set $AA = \{ \text{AutoWipingEvaluator}, \text{EnableDisableWiperWasher}, \text{FrontWiper}, \text{FrontWasher}, \text{RainSensor}, \text{RearWiper}, \text{RearWasher}, \text{WasherTank}, \text{WWManager} \}$;

We use the SPARQL query shown in Figure 14 to find the process model named “WiperWasherSystem_V1”; its sub-processes and related services are also shown in Figure 14.

This query can be repeated to identify all the sub-processes in “WiperWasherSystem_V1”. The results after the SearchForProcesses step consist of

- $RP = \{ \text{AutoWipingEvaluator_V1}, \text{EnableDisableWiperWasher_V1}, \text{FrontWiper_V1}, \text{FrontWasher_V1}, \text{RainSensing_V1}, \text{RainSensor_V1}, \text{WasherTank_V1}, \text{WiperWasherSystem_V1}, \text{WWManager_V1} \}$
- $RA = \{ \text{AutoWipingEvaluator}, \text{EnableDisableWiperWasher}, \text{FrontWiper}, \text{FrontWasher}, \text{RainSensing}, \text{RainSensor}, \text{WasherTank}, \text{WiperWasherSystem}, \text{WWManager} \}$
- $NA = \{ \text{RearWiper}, \text{RearWasher} \}$

```

prefix j.0:<http://www.owl-ontologies.com/BPO#>
prefix auto:<http://www.owl-ontologies.com/AutoPO.owl#>
select ?process ?subProcesses ?services ?location
where { ?process j.0:processName ?name.
        Filter regex (?name, "WiperWasherSystem_V1").
        ?process j.0:hasSubProcess ?subProcesses.
        ?subProcesses j.0:isPerformedBy ?services.
        ?services j.0:serviceLocation ?location.
      }

```

Results			
process	subProcesses	services	location
◆ WiperWasherSystem_V1	◆ RainSensing_V1	◆ RainSensingService_V1	URI for RainSensingService_V1
◆ WiperWasherSystem_V1	◆ EnableDisableWiperWasher_V1	◆ EnableDisableWiperWasherService_V1	URI for EnableDisableWiperWasherService_V1
◆ WiperWasherSystem_V1	◆ WWManager_V1	◆ WWManagerService_V1	URI for WWManagerService_V1
◆ WiperWasherSystem_V1	◆ FrontWasher_V1	◆ FrontWasherService_V1	URI for FrontWasherService_V1
◆ WiperWasherSystem_V1	◆ FrontWiper_V1	◆ FrontWiperService_V1	URI for FrontWiperService_V1
◆ WiperWasherSystem_V1	◆ FrontWasherTank_V1	◆ FrontWasherTankService_V1	URI for FrontWasherTankService_V1

Figure 14. Searching for reusable processes

In the new system, new processes need to be added to handle the wiper and washer for the rear window. This means updating the processes such as WWManager, EnableDisableWiperWasher and WasherTank, so that they can control the wiper/washer of both the front and rear windows. The wiper and washer of the rear window are controlled with a switch. Therefore the wiper and washer for the windshield and the rain sensing system can be reused. After the ReviewSearchingResult step, the reusable process set RP can be defined as

- $DRP = \{ AutoWipingEvaluator_V1, FrontWiper_V1, FrontWasher_V1, RainSensing_V1, RainSensor_V1 \}$
- $CRP = \{ EnableDisableWiperWasher_V1, WasherTank_V1, WiperWasherSystem_V1, WWManager_V1, \}$
- $NA = \{ RearWiper, RearWasher \}$

DRP contains reusable individuals, which are the concrete business processes that can satisfy part of the requirements of the new system. The process models in

DRP and their related services can be reused without making any changes. CRP contains the individuals of the reusable classes. The classes are the logical business processes which can be used as templates for the concrete business process design. The classes of the process models in CRP can be identified and reused to create new individuals to satisfy the new requirements.

For the new activities in NA, if no class can be reused, new taxonomy and individuals will be created and added into AutoPO. Figure 15 shows a partial process model of “WiperWasherSystem_V2”. We can see that a new process model “WiperWasherSystem_V2” is defined and its sub-processes are composed of some reused processes of V1 (such as “FrontWiper_V1” and “FrontWasher_V1”) as well as of some new processes of V2 (such as “RearWiper_V2” and “RearWasher_V2”). New concrete models for EnableDisableWiperWasher, WWManager and WasherTank are constructed by reusing the classes to satisfy the new control of wiper/washer of rear window. The new concrete models are “EnableDisableWiperWasher_V2”, “WWManager_V2” and “WasherTank_V2”. Thus, the construction of “WiperWasherSystem_V2” is based on the reuse of existing business process models. The reuse of specifications and services can help to reduce the workload of new system development and reduce the associated cost.

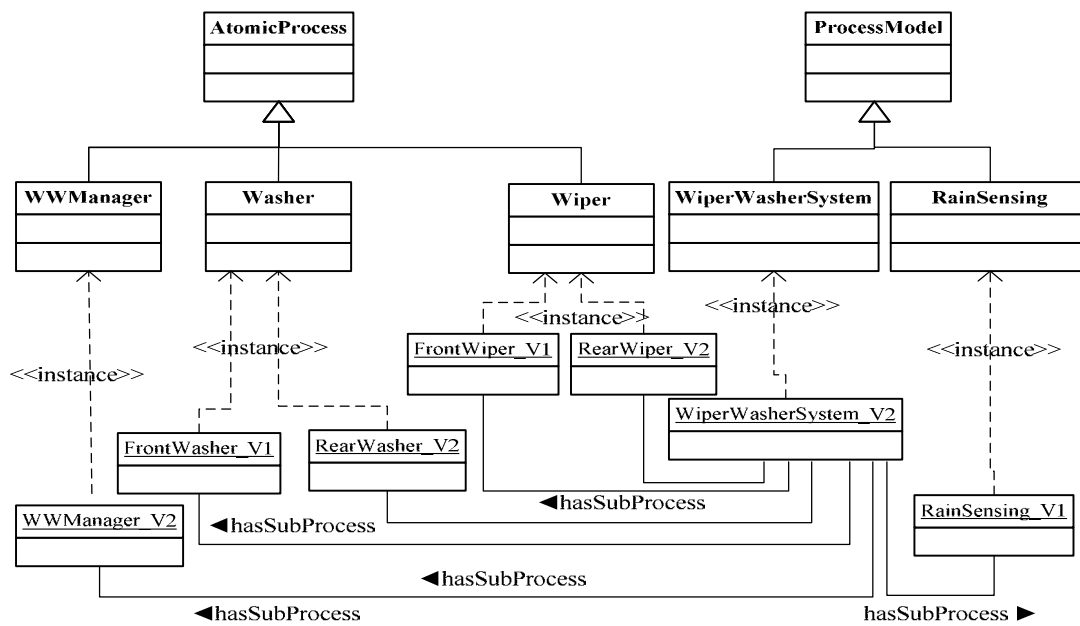


Figure 15. A partial process model of WiperWasherSystem_V2

4.1.2.3 Case 3: Creating A New Model by Reusing A Specific Service with BUM-RS

In Section 4.1.2.2, we constructed a model “WiperWasherSystem_V2” by reusing the existing process model “WiperWasherSystem_V1”. Sometimes developers may need to construct a new process by reusing specific services instead of reusing process models. We will show how to create a new model with identified reusable services in this section.

In Case 3, we assume that the developers need to reuse service “RainSensingService_V1” in a new type of car. As a service is always an implementation of a business process, we can first obtain the concrete process model, which is performed by the service “RainSensingService_V1”, and then determine whether this model can be reused without any modification or only part of the model can be reused.

We will use BUM-RS to construct the new system. After searching the service “RainSensingService_V1” in AutoPO, we can obtain the information for the processes it performs. The service searching and the results are shown in Figure 16:

<pre> prefix j.0: <http://www.owl-ontologies.com/BPO#> prefix auto: <http://www.owl-ontologies.com/AutoPO.owl#> select ?service ?process where {?process j.0:isPerformedBy ?service. ?service j.0:serviceName ?name. Filter regex(?name, "RainSensingService_V1"). } </pre>	
Results	
service	process
◆ RainSensingService_V1	◆ RainSensing_V1

Figure 16. Searching with service information (I)

Figure 16 shows that the service “RainSensingService_V1” was developed for the process “RainSensing_V1”. Therefore, the results of the SearchForServices step are:

- $RP = \{ RainSensing_V1 \}$. Since “RainSensing_V1” is a composite process, its component processes also belong to RP . This implies

$$RP = \{ RainSensing_V1, RainSensor_V1, AutoWipingEvaluator_V1 \}.$$
- $RA = \{ RainSensing, RainSensor, AutoWipingEvaluator \}$

Because the “RainSensingService_V1” is the identified reusable service, the “RainSensing_V1” model and its sub-models can be directly put into DRP. After identifying the reusable business process and service, developers can create new models, accompanying the reusable one, to satisfy the business target.

Sometimes the information of the composite processes which use the identified services indirectly is also useful for new model construction. The following query can obtain this information. The results are shown in Figure 17.

<pre> prefix j.0: <http://www.owl-ontologies.com/BPO#> prefix auto: <http://www.owl-ontologies.com/AutoPO.owl#> select ?service ?process ?compositeProcess where { ?process j.0:isPerformedBy ?service. ?service j.0:serviceName ?name. Filter regex(?name, "RainSensingService_V1"). ?process j.0:isSubProcessOf ?compositeProcess. } </pre>		
Results		
service	process	compositeProcess
◆ RainSensingService_V1	◆ RainSensing_V1	◆ WiperWasherSystem_V1
◆ RainSensingService_V1	◆ RainSensing_V1	◆ WiperWasherSystem_V2

Figure 17. Searching with service information (II)

Figure 17 shows that the service “RainSensingService_V1” implements the process “RainSensing_V1”, which is a component process of two process models “WiperWasherSystem_V1” and “WiperWasherSystem_V2”. Therefore, the initial results of the SearchForServices step are:

- $RP = \{ RainSensing_V1, WiperWasherSystem_V1, WiperWasherSystem_V2 \}$,
- $RA = \{ RainSensing, WiperWasherSystem \}$

Because “WiperWasherSystem_V1” and “WiperWasherSystem_V2” are two concrete business models to implement a business activity, the developers need to analyze whether they can satisfy the new requirement in the ReviewSearchingResult step. If either of the two models can satisfy the requirements, it can also be put into DRP. If none of the concrete processes is reusable but their template is reusable, the related class will be put into CRP. Otherwise, if neither the concrete processes nor the template can be reused, the developers must create a totally new model.

4.1.2.4 Case 4: Creating A New Model with AM-RPRS

Assuming that the “RainSensingService_V1” is a common service, several candidate process models may be obtained after the service searching step, as shown in Figure 17. We can use the query SearchByServiceAndProcess to obtain more precise results.

A prerequisite of applying AM-RPRS is that the developers should know not only the identified services but also some detailed information about the target process. For example, if the developers need to reuse “RainSensingService_V1” and the target process needs to work on the rear window, the query can be as shown in Figure 18:

```

prefix j.0: <http://www.owl-ontologies.com/BPO#>
prefix auto: <http://www.owl-ontologies.com/AutoPO.owl#>
select ?service ?process ?compositeProcess ?knownProcess
where { ?process j.0:isPerformedBy ?service.
        ?service j.0:serviceName ?name.
        Filter regex(?name, "RainSensingService_V1").
        ?process j.0:isSubProcessOf ?compositeProcess.
        ?compositeProcess j.0:hasSubProcess ?knownProcess.
        ?knownProcess j.0:processName ?pname.
        Filter regex(?pname, "Rear").
    }

```

Results			
service	process	compositeProcess	knownProcess
◆ RainSensingService_V1	◆ RainSensing_V1	◆ WiperWasherSystem_V2	◆ RearWiper_V2
◆ RainSensingService_V1	◆ RainSensing_V1	◆ WiperWasherSystem_V2	◆ RearWasher_V2

Figure 18. Combined search for the processes implemented by “RainSensingService_V1”

Figure 18 shows the combined query on AutoPO for the identified service together with the information about the process. Comparing the results in this figure

with the results in Figure 14 and 17, we can see that the results of combined query can eliminate the irrelevant results. Therefore we can focus on the most likely reusable processes. Apart from names, it is possible to generate a more precise result by using other information from the target process as the search criteria, such as the time limitation of process models and services.

Figure 18 shows that the “WiperWasherSystem_V2” model is the process model which adopts the service “RainSensingService_V1” and deals with the wiper/washer of the rear window. The results of the SearchProcessService step are

- $RP = \{ RainSensing_V1, WiperWasherSystem_V2 \}$,
- $RA = \{ RainSensing, WiperWasherSystem \}$

After the ReviewSearchingResult step, the sets DRP, CRP and RA can be identified. If the “WiperWasherSystem_V2” belongs to DRP, the developers can reuse it directly and need not create a new model again. If the “WiperWasherSystem_V2” belongs to CRP, the developers can reuse the related class WiperWasherSystem to construct a new model. If neither the “WiperWasherSystem_V2” itself nor its class can be reused, the developers can only reuse the “RainSensing_V1” model and need to create other parts of the new system model as presented in Section 4.1.2.1.

4.1.2.5 Case 5: Selecting A Suitable Method to Create A New Model

In Case 5, we will demonstrate how to select a suitable modeling method by creating a model for a front lamp cleansing system. The description of the front lamp cleansing system is as follows: The system must monitor the transparency of the lamp cover when the front lamp is lit up. An evaluator is used to analyze the collected data and to pass on the requirement that the lamp be washed. The lamp

washer will start to work when the transparency is below a certain value. The washing process lasts a certain seconds. If the status of the vehicle does not allow washing, the washing process will be suspended.

After the AbstractActivities step, the developers can obtain the following information:

- The target process $p = LampWasherSystem$;
- The process activity set $A = \{ AutoWashingEvaluator, LampWasherSystem, LampWasher, LampWashingSensing, TransparencySensor, WasherManager, WasherTank \}$.

The activity LampWashingSensing is a composite process which encapsulates the TransparencySensor and AutoWashingEvaluator.

Then the developers need to ensure that there are reusable processes or services available for the new system design. This can be done by several queries: first to identify whether there is a LampWasherSystem in AutoPO; if the result is none, the next task is to identify whether there are any washer systems in AutoPO. For example, if the query is to find the process models which have “Washer” and the composite processes they belong to, the results are shown in Figure 19. From this query, the initial results include:

- $RP = \{ EnableDisableWiperWasher_V1, EnableDisableWiperWasher_V2, EnableDisableWiperWasher_V3, FrontWasher_V1, FrontWasher_V3, FrontWasherTank_V1, FrontWasherTank_V3, RearWasher_V2, RearWasher_V3, RearWasherTank_V2, RearWasherTank_V3, WiperWasherSystem_V1, WiperWasherSystem_V2, WiperWasherSystem_V3, \}$
- $RA = \{ LampWasherSystem, LampWasher, WasherTank \}$

- $NA = \{ AutoWashingEvaluator, LampWashingSensing, TransparencySensor, WasherManager \}$

<pre> prefix j.0: <http://www.owl-ontologies.com/BPO#> prefix auto: <http://www.owl-ontologies.com/AutoPO.owl#> select ?process ?compositeProcess where { ?process j.0:processName ?name. Filter regex (?name, "Washer"). ?process j.0:isSubProcessOf ?compositeProcess. } </pre>	
Results	
process	compositeProcess
◆ EnableDisableWiperWasher_V2	◆ WiperWasherSystem_V2
◆ EnableDisableWiperWasher_V1	◆ WiperWasherSystem_V1
◆ RearWasherTank_V2	◆ WiperWasherSystem_V2
◆ RearWasher_V3	◆ WiperWasherSystem_V3
◆ FrontWasherTank_V1	◆ WiperWasherSystem_V1
◆ FrontWasherTank_V1	◆ WiperWasherSystem_V2
◆ FrontWasher_V1	◆ WiperWasherSystem_V1
◆ FrontWasher_V1	◆ WiperWasherSystem_V2
◆ FrontWasherTank_V3	◆ WiperWasherSystem_V3
◆ RearWasher_V2	◆ WiperWasherSystem_V2
◆ RearWasherTank_V3	◆ WiperWasherSystem_V3
◆ EnableDisableWiperWasher_V3	◆ WiperWasherSystem_V3
◆ FrontWasher_V3	◆ WiperWasherSystem_V3

Figure 19. Searching the processes for “Washer”

Because the three versions of WiperWasherSystem belong to RP, all their component processes are also included in RP. In the next step, ReviewSearchingResult, the developers need to refine RP and classify different kinds of reusable objects.

The lamp washer and the front washer for the windshield can use the same washer fluid tank. Therefore the washer fluid tank can be a common service for the two systems. Assuming that this type of vehicle has adopted “WiperWasherSystem_V3” for its windshield and rear window wiping and washing, then “FrontWasherTank_V3” can be reused directly in this lamp washer system design. In reviewing the search result, we may find that the general workflow of the

wiper/washer system is similar to that of the lamp washer system. The kernel of both systems is a manager, which can control the work of the actuators (wiper or washer) according to the sensing system's requirements. Although none of the existing concrete wiper/washer systems can be used for lamp washing, their logical model can be reused for the construction of the new system. Because the RainSensing process is a specific process to evaluate the amount of rain, it is necessary to modify the logical model of wiper/washer system so that it can describe more abstract processes. Therefore, after the ReviewSearchingResult step, the reusable process set RP can be refined into

- $DRP = \{ FrontWasherTank_V3 \}$
- $CPT = \{ AutoWipingEvaluator, RainSensing, RainSensor \}$
- $CRP = \{ Washer, WiperWasherSystem, WWManager \}$
- $RA = \{ LampWasherSystem, Washer, WasherManager \}$
- $NA = \{ AutoWashingEvaluator, LampWashingSensing, TransparencySensor \}$

After identifying the reusable process models, the developers can modify the process taxonomies, create new classes for the new activities, create individuals for the classes, and organize the process models. The lamp washer system can be as shown in Figure 20.

Comparing the taxonomies in Figure 20 and Figure 13, we can see that the logical model for WiperWasherSystem has been changed. A SensingSystem class has been added to represent the common process of all the sensing process in a vehicle and the RainSensing and LampWashingSensing are two kinds of sensing systems for the wiper/washer system. The general workflow of WiperWasherSystem has been modified so that the WWManager controls the actuators according to the

requirements passed by the SensingSystem. Classes for new sensors and evaluators have also been created as the sub-classes of AtomicProcess.

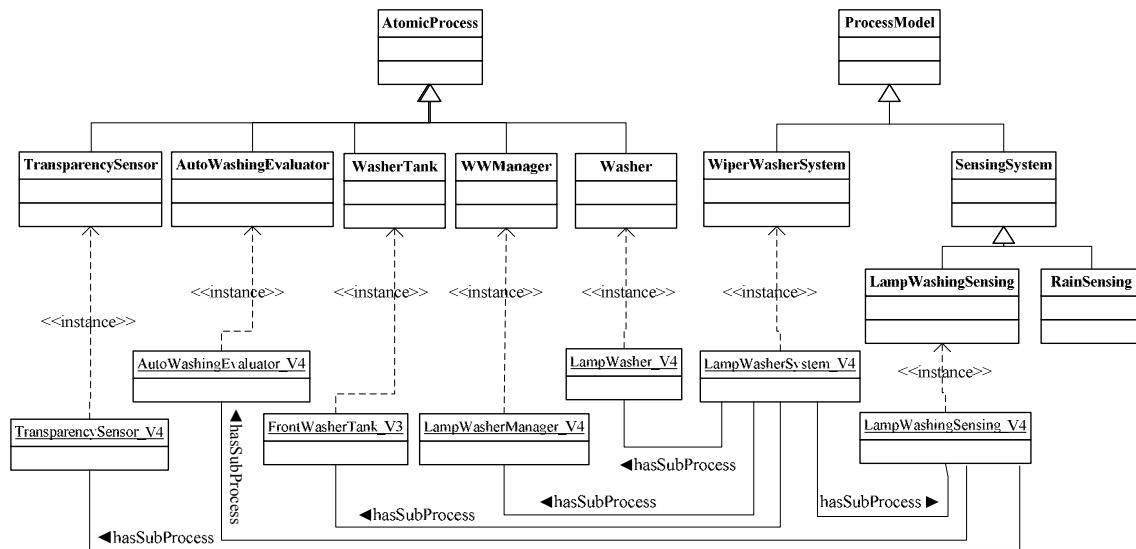


Figure 20. LampWasherSystem model

New concrete models for the lamp washer system have been constructed by creating new individuals of the classes and reusing the “FrontWasherTank_V3” process. The concrete system model is called “LampWasherSystem_V4”, and its workflow is as follows: if the “LampWashingSensing_V4” passed the requirement for lamp washing to the “LampWasherManager_V4” and the status of the vehicle was suitable, the manager would control the “LampWasher_V4” to finish the washing.

4.1.2.6 Discussion of the Case Studies

Assuming that a new wiper/washer system “WiperWasherSystem_V3” is developed in Case 3 by reusing the “RainSensingService_V1” and the system “WiperWasherSystem_V2” is reused in Case 4, we obtain the classes and individuals for Cases 1 to 4 as shown in Table 15.

Table 15. The developed process models in Case 1-4

Process Taxonomies	$\sigma_{(hasSubProcess)}$	Process Individuals
AutoWipingEvaluator		AutoWipingEvaluator_V1
EnableDisableWiperWasher		EnableDisableWiperWasher_V1; EnableDisableWiperWasher_V3
RainSensing	<RainSensing, RainSensor>; <RainSensing, AutoWipingEvaluator >	RainSensing_V1
RainSensor		RainSensor_V1
Washer		FrontWasher_V1; FrontWasher_V3; RearWasher_V2; RearWasher_V3
WasherTank		FrontWasherTank_V1; FrontWasherTank_V3; RearWasherTank_V2; RearWasherTank_V3
Wiper		FrontWiper_V1; FrontWiper_V3; RearWiper_V2; RearWiper_V3
WiperWasherSystem	<WiperWasherSystem, RainSensing>; <WiperWasherSystem, Wiper>; <WiperWasherSystem, Washer>; <WiperWasherSystem, WWManager>; <WiperWasherSystem, EnableDisableWiperWasher>; <WiperWasherSystem, WasherTank>	WiperWasherSystem_V1; WiperWasherSystem_V2; WiperWasherSystem_V3
WWManager		WWManager_V1; WWManager_V2; WWManager_V3

The classes and their relationships represent a logical model for the wiper/washer system and the individuals represent the implementations of the logical

model. The logical model shows that a wiper/washer system has six sub-processes: EnableDisableWiperWasher, RainSensing, Washer, WasherTank, WWManager and Wiper, and the RainSensing process is composed by AutoWipingEvaluator and RainSensor. The individuals which describe different concrete process models for Cases 1 to 4 are shown in Table 16.

Table 16. Concrete process models

Case 1	Case 2	Case 3	Case 4
WiperWasherSystem_V1	WiperWasherSystem_V2	WiperWasherSystem_V3	WiperWasherSystem_V2
RainSensing_V1	RainSensing_V1	RainSensing_V1	RainSensing_V1
RainSensor_V1	RainSensor_V1	RainSensor_V1	RainSensor_V1
AutoWipingEvaluator_V1	AutoWipingEvaluator_V1	AutoWipingEvaluator_V1	AutoWipingEvaluator_V1
EnableDisableWiperWasher_V1	EnableDisableWiperWasher_V2	EnableDisableWiperWasher_V3	EnableDisableWiperWasher_V2
WWManager_V1	WWManager_V2	WWManager_V3	WWManager_V2
FrontWiper_V1	FrontWiper_V1	FrontWiper_V3	FrontWiper_V1
	RearWiper_V2	RearWiper_V3	RearWiper_V2
FrontWasher_V1	FrontWasher_V1	FrontWasher_V3	FrontWasher_V1
	RearWasher_V2	RearWasher_V3	RearWasher_V2
FrontWasherTank_V1	FrontWasherTank_V1	FrontWasherTank_V3	FrontWasherTank_V1
	RearWasherTank_V2	RearWasherTank_V3	RearWasherTank_V2

In Case 1, we created a wiper/washer system process model, “WiperWasherSystem_V1”, and its sub-models with TDM. TDM follows the traditional modeling processes, analyzing the whole business task, dividing the task into separate business activities, and constructing the process model by creating models for each activity and integrate them into a composite model. In this case study, classes representing the process taxonomy of the logical model for wiper/washer systems are developed, and individuals describing the concrete process models are also constructed. These classes and individuals form the basis for the

other case studies.

In Case 2, “WiperWasherSystem_V2” was developed by reusing the process models created in Case 1. Because the basic functions and workflows of the two systems are similar, “WiperWasherSystem_V2” was constructed by reusing the class WiperWasherSystem. For the same functions, “WiperWasherSystem_V2” can adopt the existing sub-process models and services in “WiperWasherSystem_V1”. For example, “WiperWasherSystem_V2” adopts “FrontWiper_V1” and “FrontWasher_V1” as its sub-processes, because “WiperWasherSystem_V2” has the same front window wiper/washer as Case 1. For the different functions, the developers can create new models for the implementation. For example, the wiper and washer for the rear window are the new component process models in “WiperWasherSystem_V2”; the wiper/washer manager is reconstructed because it needs to control both the systems for the front and rear windows.

The developers can benefit from TDM-RP if they have the information about existing process models which have similar functions. If the name of the reusable process model is known, the query and reuse is easy and accurate as shown in Case 2. However, if the developers do not have the necessary information about the reusable process model, the query results might be less valuable.

In Case 3, we created a new model “WiperWasherSystem_V3” by reusing an existing service “RainSensingService_V1”. If a service is to be reused, its process model can also be reused without making any changes in the new system. Reusing the process model “RainSensing_V1” and related sub-processes permits us to obtain the new model as shown in Table 16.

If the developers can identify which service will be reused in the new system, BUM-RS can help them identify models related to the identified service. The model

for the service can be reused in the new system without making any changes and the developers need to consider whether the other related models of the service can also be reused. If the identified service is a common service, it may be adopted by several systems and the developers may identify multiple related process models. In this case, the ReviewSearchingResult step would take up a greater workload. If the developers cannot clearly identify the service to be reused, they can also use BUM-RS by querying the knowledge base with the information of the service, such as the service category and service ports. However, the results of this kind of query would not be as accurate as Figure 16 shows.

In Case 4, it is assumed that the new system will reuse the “RainSensingService_V1” and can deal with the rear window wiper/washer. Because the developers have specific information for both the reusable service and processes, AM-RPRS can be used and the result can be more accurate. After query, we can see that the “WiperWasherSystem_V2” adopts the “RainSensingService_V1” and provides rear window wiper/washer. Therefore, the “WiperWasherSystem_V2” can be identified as a potential reusable process model. If it can satisfy all the requirements of the new system, the developers can directly reuse it. Reusing the process model does not mean that the relevant service can also be reused without making any changes. The developers still need to check whether the service can adapt to the new environment and whether the ports can be used.

AM-RPRS can help to identify the potential reusable process models. When the OEMs have a large process model and service asset base, the query results of AM-RPRS are more accurate than other methods because it considers both the information for reusable service and process models. This also means that the developers need to collect more information than when using other methods. Without

accurate information, there may be multiple query results and analysis of the results would take more time.

Using these four case studies, we have shown applications of the four modeling methods. The first case shows how to create models and maintain the knowledge base AutoPO; the other three cases show how to query the knowledge base and reuse the process models and services. Because the four modeling methods have the same beginning steps: *AnalyzeRequirements* and *AbstractActivities*, developers can collect information of business target in the initial analysis, and determine which modeling method is suitable for the development next. Case 5 shows this procedure.

In Case 5, a lamp washer system process model is created by reusing the logical model for those wiper washer systems. Because the original logical model is too specific for the wiping system, the related classes are modified to satisfy the new requirements. The logical models are compared in Table 17.

From Table 17, we can see that a *SensingSystem* class and a *LampWashingSensing* class are added into the ontology. The *SensingSystem* represents the common process of all the sensing process in a vehicle and the *LampWashingSensing* together with *RainSensing* are sub-classes of *SensingSystem*, representing two kinds of sensing systems for the wiper/washer system. Two classes for representing the sensors and evaluators for the lamp washing system, *AutoWashingEvaluator* and *TransparencySensor* are also created as the sub-classes of *AtomicProcess*. After modifying the logical models, the construction of the new process model is similar to the other case studies.

Table 17. Comparison between logical models

Process Taxonomies (Before implementing Case 5)	Process Taxonomies (After implementing Case 5)
ProcessModel -WiperWasherSystem -RainSensing	ProcessModel -WiperWasherSystem -SensingSystem -RainSensing -LampWashingSensing
AtomicProcess -AutoWipingEvaluator -EnableDisableWiperWasher -RainSensor -Washer -WasherTank -Wiper -WWManager	AtomicProcess -AutoWashingEvaluator -AutoWipingEvaluator -EnableDisableWiperWasher -RainSensor -TransparencySensor -Washer -WasherTank -Wiper -WWManager

Through these cases, we have shown that the knowledge base AutoPO can work as the repository for storing and managing the business process models and information about their related services. Developers can use the four modeling methods not only to create new models directly as in traditional development but also can reuse the existing logical and concrete models for the new system. The knowledge base AutoPO stores the modeling knowledge in a well structured system and can be maintained during the modeling process.

4.2 Comparison of the Automotive Software Modeling Methods

In this section, we compare some of the automotive software modeling methods presented earlier in Section 2.4 and our ontology-based method. The main items of comparison are shown in Table 18. In this comparison, the following features of the modeling methods are considered.

- **Fundamental Theory:** Describe the basic theory or techniques of the modeling methods.
- **Modeling Focus:** Identify the main modeling results.
- **Unit of Modeling:** Identify the basic building blocks of the models.
- **Hardware-coherent:** Identify whether the basic building blocks of the modeling methods include hardware.
- **Heterogeneity:** Identify whether the modeling methods support the heterogeneity of distributed systems.
- **Interactivity:** Identify whether the modeling methods support the interactivity between different sub-systems.
- **Timeliness:** Identify whether the modeling methods support real-time modeling.
- **Concurrency:** Identify whether the modeling methods support the concurrent feature of distributed systems.
- **Model reuse:** Identify whether the modeling methods support the reuse of business process specifications.
- **Service reuse:** Identify whether the modeling methods support the reuse of

existing software systems.

- **Tool Support:** Identify the automatic tools that can be used for the modeling methods.

Table 18 shows that the modeling methods are based on different fundamental theories, deal with different modeling issues, and support the attributes of automotive software at different degrees.

Table 18. Comparison of automotive software modeling methods

	MATLAB® Family tools	UML-based Methods	MSC-based Method	AutoPO-based method
Fundamental Theory	Control theory	Object-Oriented	Service-Oriented	Service-Oriented
Modeling Focus	Structural modeling; Behavior Modeling	Component modeling	Communication modeling	Business process modeling
Unit of Modeling	Component	Component	Service	Service
Hardware-Coherent	Yes	Yes	No	No
Heterogeneity	Support	Support	Support	Support
Interactivity	Support	Support	Support	Support
Timeliness	Support	Support	Support	Support
Concurrency	Support	Support	Support	Support
Model reuse	Not support	Not support	Not support	Support
Service reuse	Not support	Not support	Support	Support
Tool support	MATLAB® tool set	General UML editing tools	Prototypic tool chain	General ontology editing tools

The fundamental theory and techniques of the modeling methods include control theory, object oriented modeling, and service oriented modeling. Control theory deals with the behavior of dynamic systems and can provide mathematical models for representing the inputs, outputs and states of the systems (Kilian, 2006).

Using this theoretical base, MATLAB® family tools can provide structural modeling

and behavior modeling for the automotive systems. Object-Oriented Modeling is a modeling paradigm mainly used in computer programming (Fowler, 1997). It can address the target problem as a set of related, interacting objects and the modeling task is to specify the objects (or the class the objects belongs to), their properties and methods. By applying new notations to real-time system descriptions, UML can be used to represent the models for both the dynamical systems and information systems in an automobile. The Object-Oriented models can be mapped onto implementation classes and interfaces, which can be regarded as detailed descriptions of the system's construction. Service-oriented modeling is a software development methodology that employs business disciplines and a universal language to provide tactical and strategic solutions to enterprise problems (Erl, 2005). One of the targets of service-oriented modeling is to create models that can be understood by individuals with diverse levels of business and technical background.

Each of these three fundamental approaches supports different levels of modeling. Some are technical and focus on the detailed design of systems while others are business oriented and focus on the orchestration of the systems. They support different levels of reusability. Control theory does not provide direct support for reusability, Object-Oriented Modeling supports the reuse of object models, and Service-Oriented Modeling supports the reuse of services and applications.

Thus, these modeling methods have different modeling focuses. MATLAB® family tools mainly support the structural and behavioral modeling of the systems. UML-based methods mainly support the analysis of the problem domain and show the static structure, dynamic behavior, and run-time deployment of the collaborating objects in the system. These methods can support the component-based system modeling and development. Both MSC-based and our ontology-based methods are

service-oriented modeling methods, however, and adopt different techniques for model representation and deal with different modeling facets. The MSC-based method adopts a Message Sequence Chart to represent the models. MSC provides a graphical and textual language for the description and specification of the interactions between system components. In this method, services are considered as system components and MSC charts are used to represent the communication between the services. Our ontology-based method adopts OWL to describe the business processes and related services. The relationships between the business processes can represent the structure of a composite process and the communication between component processes in the composite process.

The four modeling methods support different levels of abstraction. The MATLAB® family tools and UML-based methods can be used for detailed modeling of a service while the MSC-based method and our ontology-based method can be used for business process modeling, which includes service composition and orchestration. Therefore, their basic units of modeling are different and the degrees of hardware-coherence are different. All four methods can support most of the specific features of automotive software (Heterogeneity, Interactivity, Timeliness, and Concurrency), as they all have specific notations to represent those features. However, the same features may have different representations under different modeling methods. For example, UML-based methods provide notations to represent the concurrency of objects; MSC-based method and our ontology method can only declare that two or more services can run concurrently.

Different levels of abstraction also reflect on the different levels of support for reusability. MATLAB® family tools and UML-based methods show limitations in reusing the specification and applications because they focus more on the detailed

design of the applications and the composition of the components. On the other hand, the MSC-based method supports the reuse of services but does not support the management of the process model asset bases and the service asset bases. Therefore, developers using the MSC-based method cannot retrieve reusable services. Compared with all of these methods, our ontology-based method exhibits good levels of reusability of both specifications and services.

In summary, all four methods are based on different fundamentals, can support modeling from different levels of abstraction, and all support the basic features of automotive software modeling. However, only the ontology-based method can support both specification and service reuse.

4.3 Verification of SOA Models

After the modeling phase, model testing should be conducted to ensure the quality of the models. Researchers have developed detailed standards and guidelines on software products quality, as described in Section 2.3. However, there is still a lack of quality attributes specifically for SOA models. This section proposes a set of quality attributes for SOA models based on the ISO 9126 standard and related quality models presented in Section 2.3 and comments on how our methodology can support these quality attributes.

4.3.1 Quality Attributes for SOA Models

Among the quality attributes in ISO 9126, four quality attributes and their eight sub-attributes are relevant to SOA model, as shown in Table 19.

Comparing Table 19 with Table 2, we can see that SOA models should have

most of the sub-attributes of Functionality. An SOA model should describe the business processes accurately, consider the interoperability of component services, and specify the security strategy of the software. These are relevant functional quality attributes for an SOA model. The model should also be understandable so that it can be used in the following phases of the development lifecycle. Generally, a model is required to be understandable by human (users and developers); however, the SOA model is required to be understandable by both human and computers, so that the development can be automated. Beside these attributes, the model should be maintainable so that it can be extended or changed to satisfy the changing user requirements.

Table 19. The ISO 9126 Quality attributes applicable to SOA model

Attributes	Sub-Attributes	Description	Applicable to SOA model
Functionality	Accuracy	To provide the right or agreed results with the needed degree of precision	Yes
	Suitability	To provide an appropriate set of functions for specified tasks and user objectives	Yes
	Interoperability	To interact with one or more specified systems	Yes
	Security	To protect data from unauthorized persons	Yes
Usability	Understandability	To enable the users to understand whether the software is suitable, and how it can be used.	Yes
Efficiency	Resource Utilization	To use appropriate amounts and types of resources when the software performs its function under stated conditions.	Yes (to be discussed)
	Time Behavior	To provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.	Yes (to be discussed)
Maintainability	Changeability	To enable a specified modification to be implemented	Yes

The other attributes in Table 2 are not suitable for SOA model. For example, the reliability attribute and its sub-attributes emphasize more on the maturity of software

processes and software performance in case of failure; the portability attribute is unsuitable because the SOA application is a distributed system with both the application and its component services available on the Net and it does not need installation and migration.

As an SOA model also captures the workflow of the activities (which are implemented by services), it should be predictable so that developers can predict the resources and execution time needed and detect any conflicting resource. For this reason, we believe the two sub-attributes, “resource utilization” and “time behavior”, are also relevant to SOA model; however, their meaning is not exactly as defined in ISO 9126.

As these attributes are extracted from the quality model for general software, they do not cover all the relevant attributes for SOA model. We need some additional complementary quality attributes. Table 20 shows the mapping of quality attributes of various quality models presented in Section 2.3.

Table 20. Mapping of the quality attributes

ISO 9126	UML model	Information Model
Accuracy		
	Completeness	Correctness
	Consistency	
Suitability	Detailedness	Relevance
	Conciseness	
Interoperability		
Security		
Resource Utilization		Systematic Design
Time Behavior		
Understandability	Complexity	Clarity
	Esthetics	
Changeability		
	Traceability	
		Economic Efficiency

	Balance	
	Correspondence	
		Comparability

On the basis of these quality attributes, a set of quality attributes for SOA models is proposed as follows.

- **Accuracy:** The SOA model should be accurate and unambiguous, describe the requirements clearly, and direct to the right results.
- **Completeness:** All the conditions in the user requirement should be covered in the SOA model.
- **Consistency:** The information in the SOA model should not be contradictory.
- **Suitability:** The SOA model should provide an appropriate set of functions for business targets. The model should be described in suitable detail and its components should be in suitable granularity.
- **Interoperability:** The SOA model should ensure the interoperability of different component services.
- **Security:** The SOA model should address the security concerns.
- **Resource Utilization:** The SOA model should enable the organization to predict the resource requirement and detect any conflicting resource.
- **Time Prediction:** The SOA model should enable the organization to predict the execution time of a business process.
- **Understandability:** The SOA model is required to be understandable by human (users and developers) and computers.
- **Maintainability:** The SOA model should be maintainable so that the application can be changed and extended in the future.

As the changeability, traceability, and economic efficiency attributes are related to the changes and reuses of SOA models, we use the Maintainability attribute to synthesize them. The Correspondence attribute of Table 20 is ignored here because it considers the consistency between the model and its system. To test Comparability, we need a universal modeling standard. Although WSBPEL is popular, it is not accepted by all the users. It is not practicable to compare models described in different languages without conversions. For those reasons, Comparability is not included in our set of quality attributes for SOA models.

4.3.2 Verification of Our SOA Models

Verification⁵ is the activity which ensures the work products of a given phase fully implement the inputs to that phase. It concerns that the product was built right. In this section, we discuss how to verify the SOA models developed with our methodology and how the models can satisfy the proposed quality attributes.

The SOA models generated by our methods are ontologies in a knowledge base described in OWL. The verification for the models can be divided into two parts, one is ontology evaluation and the other is model checking.

Although a well-evaluated ontology cannot guarantee the absence of problems, it will make its use safer (Gómez-Pérez, 2004). The quality of ontologies will affect the modeling performance directly. Therefore, besides the quality attributes for SOA models, the quality of ontology construction is also a verification item for the models.

The possible errors in building ontology can be classified into three categories

⁵ <http://www.critech.com/vv.htm>

and methods have been proposed to deal with some of them (Gómez-Pérez, 1996; Gómez-Pérez, 2004; Fahad et al., 2007; and Qadir et al., 2007). The error categories include inconsistency errors, incompleteness errors and redundancy errors and each category has been further divided into different sub-categories, as shown in Table 21.

Table 21. Category of possible errors when developing ontologies

Category	Explanation
Inconsistency	The errors cause inconsistent in an ontology.
Circularity Errors	A class is defined as a specialization or generalization of itself.
Partition Errors	A class is defined as sub-classes of more than one disjointed partition classes of another class. Or, an instance belongs to more than one sub-class of the defined partition.
Semantic Errors	A class is defined not respecting the real world.
Incompleteness	The errors cause incompleteness in an ontology.
Incomplete Concept Classification	Concepts are classified without accounting for all of them.
Omission of disjoint knowledge	The definition of the partition between a set of classes is omitted.
Redundancy	Multiple definitions for a class or an instance in an ontology.
Grammatical	More than one explicit definition of subclassOf or instanceOf relations.
Identical formal definition of some classes	There are two or more classes in the ontology with the same formal definition but with different names.
Identical formal definition of some instances	There are two or more instances in the ontology with the same formal definition but with different names.

When extending and using BPO and the relevant knowledge base, the developers need to evaluate the ontologies to ensure that no error listed in Table 21 occurs. Ontology editing tools (e.g. Protégé (2008)) usually provide reasoners (e.g. RacerPro) to assist in the ontology checking, which can typically check several kinds of errors. Generally, only the inconsistency errors can be automatically detected. Using the function consistency checking, the reasoner will go through the whole

ontology, check according to the defined axioms and issue warning of possible inconsistent classes and individuals. Then the developers can modify the ontology according to the checking result. For example, *DisjointClasses (AtomicProcess CompositeProcess)* axiom is a basic axiom in BPO which ensures that a business process can only be a composite process or an atomic process and it can be used to detect partition errors in the ontology.

Protégé provides redundancy checking for users in its editing function. The classes or individuals should have different identifications. If the identification of a new class or individual is the same as others, the input will be denied. In this way, grammatical redundancy problems can be avoided. For the other two redundancy problems, techniques such as ontology alignment (Ehrig, 2007) and ontology lexicons (Hirst, 2004) may help. Ontology alignment can be used to map two ontologies to find similar structures and ontology lexicons can be used to solve the problem of different terms with the same meaning.

There is no specific tool to check for incompleteness error. The successful checking of incompleteness is determined by how much knowledge has been built-in in the ontology. For example, if the *hasSubProcess* property has a restriction, “one composite process should have one or more business processes as its sub-processes”, the reasoner can check for the following situation: A is a composite process and A has no sub-process. However, the following situation cannot be checked: A should have three sub-processes in the real world and only two of them have been defined in the ontology.

According to the above discussion, we can see that most of the ontology errors can be avoided or tested automatically and semi-automatically. If the problem is the inconsistency between the ontology definition and the real world, it can only be

checked manually.

Comparing the quality requirements of ontology development and the quality attributes of SOA models, we can find that the former ensure the models are well described and the latter ensure the models are well constructed. A well-evaluated ontology forms the basis of a high quality SOA model. Next, we will discuss how our methodology can satisfy the quality attributes discussed in Section 4.3.1.

➤ **Accuracy**

OWL is adopted for the model description in our methodology. As introduced in Section 3.1, OWL is an XML-based formal language for ontology description, which provides vocabulary and axioms to clearly and accurately describe the structure of the models and the relationship between the model components. As discussed above, this ensures the accuracy of model description, but cannot ensure that the model is designed right. To test whether the model is right, developers need to use specific techniques, such as simulation and model checking, which will be discussed at the end of this chapter.

➤ **Completeness and Consistency**

As discussed earlier, ontology techniques can only ensure the completeness and consistency of the ontology itself. To evaluate whether the ontology describes all the facts in the real world and whether the model can obtain right results, developers need to use other testing methods, such as simulation and model checking. In this study, we check these attributes manually by using the ontology query tool SPARQL to check whether the results are consistent with the reality.

➤ **Suitability and Security**

The verification of these two attributes is directly related to the specific

strategy of the organization. In our study, we assume that the knowledge base is constructed on the basis of an organization's asset base. The services and processes should be granulated according to the organization's rules and the security of the existing processes and services should have been tested.

➤ **Interoperability, Resource Utilization, Time Prediction**

Specific properties for processes and services are defined in our methodology to represent their interoperability, resource and time limitation. For example, processIO specifies the message transition between processes; processTimeLimit can specify the time requirement for a process execution. In simulation or model checking, these properties can be used to check the interoperability of the processes, and predict the execution time needed for the application. Resource related properties currently are not defined in AutoPO; however, they can be easily added to the models in the extension of the ontology.

➤ **Understandability**

We adopt OWL as the model description language. OWL has XML-based syntax, with which the models can be understood by both human and computer. Ontology editing tools (e.g. Protégé (2008)) also provide visual interface to represent the structure of ontology.

➤ **Maintainability**

The extensibility of ontology ensures the maintainability of our models. In our methodology, model maintenance is a procedure of knowledge base maintenance, because the SOA models are constructed on the basis of BPO or AutoPO. To maintain the models, developers only need to know how to construct new sub-classes and identify their defined attributes. The developers

can do this without learning all the features of OWL. However, the extension of the basic concepts in BPO or AutoPO should be maintained by ontology experts.

Through this discussion, we can find that some attributes, such as understandability and maintainability, are satisfied by adopting ontology-based techniques; other attributes such as interoperability and time prediction are satisfied by defining specific properties or axioms in BPO and AutoPO; and attributes such as suitability and security are dependent on the specific strategies of the organization. However, all these can only ensure producing a good quality model. To ensure that a model is qualified, more testing is still needed, such as simulation (An et al., 2005) and model checking (Clarke et al., 1999).

Simulation and model checking are common methods for process analysis. Simulation can be used to analyze the behavior of either real or imaginary systems over time. In business process modeling, it can be used to verify the executing order of the business activities, detect the conflicting resource and message mismatching, and predict execution time. Nowadays, simulation is still a valuable method for complex system testing (An et al., 2005). Model checking is a systematic way to exhaust all possible states of a model to detect any potential violation that the model has against its requirement (Clarke et al., 1999). Recently, it has been adopted to verify web services (Huang et al. 2006) and assure the e-commerce transactions (Anderson et al. 2005).

However, none of these methods and tools can support the testing of all the quality attributes of SOA models (Liao et al., 2007b). Although some of the simulation tools or model checking tools can support the automatic test or

semi-automatic test, they only support specific modeling languages (e.g. WSBPEL).

In addition, none of those methods and tools supports OWL. If we can translate our models into other XML-based business process languages, such as WSBPEL, our SOA models may be assessed by more methods and tools. This can be one of the future works of our investigation. Therefore, in this study, we check the models manually by comparing the results of knowledge base query and the expected results.

4.4 Summary

In this chapter, we adopted an automotive software development scenario to validate our ontology-based SOA modeling methodology. A knowledge base AutoPO was constructed by extending the core ontology BPO. On the basis of AutoPO, we used a series of case studies to demonstrate the execution of the modeling methods. After illustrating the application of our methods through these case studies, we compared our modeling method with other automotive software modeling methods, which shows the benefits of our ontology-based service-oriented modeling methodology. This chapter also presented the verification of SOA models by proposing a set of quality attributes for SOA models and discussing how our methodology can support these quality attributes.

Chapter 5 Conclusion and Future Work

This thesis has proposed an ontology-based modeling methodology for SOA development. The methodology begins with a core ontology (Business Process Ontology) and a modeling and developing framework (OBPMDF) using the extension of the BPO. Based on BPO and OBPMDF, we proposed four modeling methods to deal with different kinds of modeling requirements.

The benefits of our methodology include:

(1) Our methodology provides a generic framework BPO for SOA modeling which bridges the business process modeling and services.

BPO provides hierarchical structure to represent business processes and services. It provides classes and properties to represent atomic processes/services and the construction of composite processes/services. The binding information between the business process and services provides a bridge between business process modeling and relevant implementation.

(2) Our methodology provides a method to create and maintain a sharable knowledge base for organizations.

Using knowledge management techniques, BPO can be extended into a knowledge base for the business process modeling in a specific domain, providing the information of business processes and services the organization owns. This knowledge base contains process model templates, concrete business processes and their implementation (services).

To validate this, we applied BPO to the automotive domain. The initial extension includes adding new concepts according to AUTOSAR to represent

specific domain features for automotive software, such as automotive software category and service ports. The knowledge base AutoPO is constructed on the basis of this initial extension. AutoPO allows the adoption of knowledge management techniques to support querying and maintenance of the knowledge base. After further extension, AutoPO can store all the business process models and information for the relevant services, providing a knowledge base for the processes involved in operating a car. In AutoPO, classes can represent the logical model of the process models and individuals can represent concrete process models and services. The individuals can be reused directly if the process model or service exactly suits the requirements of the new system. The classes also provide the reusable modeling templates.

(3) Our methodology provides flexible modeling methods to satisfy different requirements.

To support searching for the reusable objects in AutoPO and to assist in constructing business process models, four SOA modeling methods can be used.

- **Top-Down Modeling (TDM)** assists the organization in creating new process models directly and uses AutoPO to help to generate the formal requirement specification.
- **Top-Down Modeling based on Reusable Process (TDM-RP)** assists the organization in constructing new process models by reusing similar process models already defined in AutoPO.
- **Bottom-Up Modeling based on Reusable Services (BUM-RS)** assists the organization in constructing new process models by reusing some specific services.
- **Agile Modeling based on Reusable Process and Reusable Services**

(AM-RPRS) provides specific queries in AutoPO and can help the organization efficiently choose suitable models. The queries are for both service and business process information.

(4) Our methodology supports the reuse of business process templates, concrete business processes and relevant service, which can help to improve the software quality and reduce the development cost.

Because all the processes and services stored in the knowledge base have been well evaluated by users, their quality can be ensured in the reuse. This can be seen as the unit testing has been finished and the developers only need to integrate each unit and do integration testing. Therefore our methodology can help to improve the software quality and reduce the cost.

Based on an analysis of different quality models for software, we also proposed a set of quality attributes for SOA models. The attributes include: accuracy, completeness, consistency, suitability, interoperability, security, understandability, maintainability, resource utilization and time prediction. Not all the attributes can be tested, for example, the understandability and maintainability of a model are dependent on the capability of model description language. The models generated by our method should have good understandability, because our method is based on OWL, which can provide good understandability for both humans and computer. These models also offer good maintainability because extensibility is one of the most important features of ontology.

Because automated tools for SOA model testing are still lacking (Liao et al., 2007b), we only checked the ontology consistency with automatic tools provided by Protégé (2008). We checked the other attributes manually.

Although our modeling methodology can formally describe the automotive

models and improve the reusability of the automotive processes and services, a number of issues remain to be addressed.

(1) Besides the application in automotive software development domain, more extensive evaluation is needed.

In our study, we also applied BPO and the modeling methods to the e-banking domain (Liao et al., 2007), and validated the usage of the modeling methodology with a series of case studies for the loan process. This experiment demonstrates that BPO is very generic and has wide applicability. However, more comprehensive evaluation of the methodology is still needed.

(2) To use the queries in the methodology, the developers need the knowledge of SPARQL.

In the thesis, we only provide some examples of the queries, such as *SearchByProcessName* and *SearchByTimeLimit*. To query the knowledge base effectively and efficiently, the developers need to learn the query language SPARQL first. This may cause difficulty for the developers. To solve this problem, next we will design a series of query templates, which can cover the functional and performance query requirements for the processes and services.

(3) AutoPO is constructed on the basis of AUTOSAR, however, it may not be fully comprehensive for usage in the automotive industry.

The framework of AutoPO is constructed by extending BPO with automotive software categories and ports for services. We validate the usage of AutoPO with case studies. However, the real industry environment is more complex, and the current version of AutoPO may not be fully comprehensive for representing those complex situations. Therefore, we need to further improve the framework of AutoPO according to the requirements of industry organizations in the future.

(4) The maintenance of ontologies may need automatic tools.

In the thesis, we adopted an ontology editor Protégé (2008) to build up and maintain the ontologies. In our methodology, the ontologies are maintained manually. The manual approach to ontology maintenance involves a lot of work for the experts, and may generate less structured taxonomy, with few complex relations and axioms (Blomqvista et al., 2008).

The quality of ontologies affects the modeling performance in two aspects:

- The definitions of concepts and relations in the ontologies should be consistent. Inconsistent concepts in the ontologies may cause confusion, and may result in invalid knowledge for the organizations. Generally, this kind of faults can be detected automatically.
- The definitions in the ontologies should be consistent with the reality. Incomplete definitions and inconsistency between concepts and reality may cause wrong result in the modeling process. This kind of faults is difficult to detect by tools. Generally, this is highly dependent on the experience of the ontology developers.

Thus, in future work we plan to apply ontology learning and linguistic analysis (Salam et al., 2007) to automatically or semi-automatically construct ontology taxonomies by extracting and tagging business information from plain text or other evidence. Such an automated modeling should be able to abstract concepts and relationships from the initial specification written in natural languages and support automatic query and extension of the ontology.

Besides fixing the above limitations, more investigation can be done in the following domains:

(1) To introduce more formal treatments into the methodology.

Currently, we use OWL DL (W3C, 2004c) to describe BPO and AutoPO. The axioms we adopted include: domain, DisjointClass, inverseOf, range, subclassOf, subPropertyOf, transitiveProperty. These axioms can help to describe specific attributes of the entities in BPO and AutoPO, such as hierarchical relationships between the classes/properties and the partition of classes/properties. These axioms can also be used for reasoning. To express more complex properties, more axioms can be added. For example, “hasValue” can be used to restrict the property value to a specific value resource.

(2) To apply more ontology techniques for more benefits.

For example, in automotive software modeling, we can use techniques such as ontology alignment (Ehrig, 2007) to map and merge two ontologies and ontology lexicons (Hirst, 2004) to map concepts from an ontology which has different terms for the same meaning. The narrow goal of this would be to accelerate the construction and application of AutoPO in OEMs but a broader goal would be to construct a universal ontology as a standard business process template repository with standardized services supporting the automotive software domain, thereby accelerating knowledge sharing between OEMs and suppliers and further improving the reuse of software.

(3) To develop suitable methods for verifying our SOA models.

According to our survey on verification and validation methods for models, effective and efficient testing tools specific for SOA model are lacking (Liao et al., 2007b). Although some of the simulation tools or model checking tools can support the testing of SOA models, they only support specific modeling languages (e.g. WSBPEL). Therefore, developing language translating tools and enhancing the

model testing methods and tools to address more quality requirements are also valuable works in the future.

(4) To extend the methodology to integrate with other phases of SOA development.

Our methodology focuses on the modeling of business process models. Modeling is only one step in the SOA application development. To further improve the development efficiency and the software quality, our methodology should be able to seamlessly integrate with other phases of SOA development, such as assembling and integration testing.

In conclusion, bringing ontology techniques to SOA paradigm can improve the reusability of business models and services and thereby reduce the costs and improve the quality of the software. This study has sought to make a useful contribution towards this goal.

Bibliography

- AMI-C, 2003. AMI-C release 2, architectural overview. Available at <http://www.ami-c.org/>.
- An, L. and Jeng, J.J., 2005. On Developing System Dynamics Model for Business Process Simulation. Proceedings of the 2005 Winter Simulation Conference, pages 2068-2077.
- Anderson, B.B., Hansen, J.V., Lowry, P.B. and Summers, S.L., 2005. Model Checking for E-Business Control and Assurance. IEEE Transactions on Systems, Man and Cybernetics, Part C, 35 (3), pages 445-450.
- Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Leymann, J.K.F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S., 2002. Business process execution language for Web services. Available at <http://www.ibm.com/developerworks/webservices>.
- Angele, J., Erdmann, M. and Wenke, D., 2008. Ontology-Based Knowledge Management in Automotive Engineering Scenarios. Ontology Management, 7, pages 245-264.
- Arsanjani, A., 2004. Service-oriented modeling and architecture. Available at <http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/>.
- AUTOSAR, 2006a. AUTOSAR technical overview v2.0.1, Available at http://www.autosar.org/download/r2/AUTOSAR_TechnicalOverview.pdf.
- AUTOSAR, 2006b. AUTOSAR software component template v2.0.1, Available at http://www.autosar.org/download/r2/AUTOSAR_SoftwareComponentTemplate.pdf.
- Baida, Z., Gordijn, J. and Omelayenko B., 2004. A shared service terminology for online service provisioning. Proceedings of the 6th international conference on Electronic commerce, Delft, The Netherlands, pages 1-10.
- Bass, L., Clements, P. and Kazman, R., 2003. Software Architecture in Practice, Second Edition, Boston: Addison-Wesley.
- Becker, J., Rosemann M. and von Uthmann, C., 2000. Guidelines of Business Process Modeling. In van der Aalst, W., Desel, J. and Oberweis. A. (eds.), Business Process

- Management: Models, Techniques and Empirical Studies, Springer-Verlag, pages 30-49.
- Bercovici A., Fournier F. and Wecker A.J., 2008. From Business Architecture to SOA Realization Using MDD. Lecture Notes in Computer Science, Model Driven Architecture – Foundations and Applications, pages 381-392.
- Blomqvista, E. and Öhgren, A., 2008. Constructing an enterprise ontology for an automotive supplier. Engineering Applications of Artificial Intelligence, 21(3), pages 386-397.
- Boggs, W. and Boggs, M., 2003. Mastering Rational XDE, San Francisco, Calif.: SYBEX.
- Bouras, A., Gouvas, P., Mentzas, G., 2007. A Semantic Service-Oriented Architecture for Business Process Fusion. In Salam, A.F. and Stevens, J.R. (eds.), Semantic Web Technologies and E-Business - Toward the Integrated Virtual Organization and Business Process Automation, pages 40-76.
- Broy, M., 2005. Automotive software and system engineering. Proceedings of the ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE), pages 143-149.
- Broy, M., 2006. The “Grand Challenge” in informatics: engineering software-intensive systems, Computer, 39(10), pages 72-80.
- Broy, M., Krüger, I.H. and Meisinger, M., 2007a. A formal model of services. ACM Transactions on Software Engineering and Methodology (TOSEM), 16(1), pages 1-40.
- Broy, M., Krüger, I.H., Pretschner, A. and Salzmann, C., 2007b. Engineering Automotive Software. Proceedings of the IEEE 95(2), pages 356-373.
- Chen-Burger, Y.H. and Robertson, D., 2005. Automating business modelling: a guide to using logic to represent informal methods and support reasoning. Springer.
- Cimiano, P., 2006. Ontology learning and population from text: algorithms, evaluation and application. Springer.
- Clarke, E., Grumberg, O. and Peled, D., 1999. Model Checking. MIT Press.

- Cybulski, J.L. and Reed, K., 2000. Requirements Classification and Reuse: Crossing Domain Boundaries. Proceedings of the 6th International Conference on Software Reuse, Lecture Notes in Computer Science 1844, Springer, Berlin, pages 190-210.
- Dannenbergh, J. and Kleinhans, C., 2004. The coming age of collaboration in the automotive industry. Mercer Management Journal, 17, pages 88-94.
- de Deugd, S., Carroll, R., Kelly, K.E., Millett, B. and Ricker, J., 2006. SODA: Service-Oriented Device Architecture. In IEEE Pervasive Computing, 5(3), pages 94-96.
- Desai, N., Mallya, A.U., Chopra, A.K. and Singh, M.P., 2005. OWL-P: A methodology for business process modeling and enactment. Proceedings of the Workshop on Agent Oriented Information Systems, pages 50-57.
- Ehrig, M., 2007. Ontology Alignment-Bridging the Semantic Gap, New York, Springer Science + Business Media, LLC.
- Emaus, B., 2005. Hitchhiker's guide to the automotive embedded software universe. Proceedings of the SEAS'05 Workshop, Keynote Presentation. Available at http://www.inf.ethz.ch/personal/pretscha/events/seas05/bruce_emaus_keynote_050521.pdf.
- Erl, T., 2005. Service-Oriented Architecture Concepts, Technology, and Design. Prentice Hall PTR.
- Fahad, M., Qadir, M.A. and Noshairwan, M.W., 2007. Semantic Inconsistency Errors in Ontology. Proceedings of the IEEE International Conference on Granular Computing, 2007, GRC 2007, pages 283- 286
- Fang, L.N., Tang, S.Q., Yang, Y., Xiao, R.L., Li, L., Deng, X.G., Xu, Y. and Xu, Y.W., 2007. A User-Driven Slight Ontology Framework Based on Meta-Ontology for Change Management. Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, Volume 1, pages 1007-1014.

- Fennel, H. and Bunzel, S., 2006. Achievement and exploitation of the AUTOSAR development partnership. CONVERGENCE 2006.
- Fowler, M., 1997. Analysis Patterns: Reusable Object Models. Menlo Park. Calif: Addison-Wesley.
- Gardner, T., 2003. UML modeling of automated business processes with a mapping to BPEL4WS. Proceedings of the First European Workshop on Object Orientation and Web Services at ECOOP. Available at <http://www.cs.ucl.ac.uk/staff/g.piccinelli/eoows/documents/./study-gardner.pdf>.
- Gennari, J., Musen, M., Ferguson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy N. and Tu, S., 2003. The evolution of Protégé 2000: An environment for knowledge-based systems development. International Journal of Human Computer Studies, 58(1), pages 89-123.
- Gerard, S. and Terrier, F., 2003. UML for real time: Which native concepts to use?. In Lavagno, L., Martin, G and Selic, B. (eds.), UML for real: design of embedded real-time systems, pages 271-299.
- Gómez-Pérez, A., 1996. A Framework to Verify Knowledge Sharing Technology. Expert Systems with Application, 11(4), pages 519-529.
- Gómez-Pérez, A., 2004. Ontology Evaluation. In Staab, S. and Studer, R. (eds.), Handbook on ontologies, New York : Springer-Verlag, pages:251-274.
- Gouvas, P., Bouras, T. and Mentzas, G., 2007. An OSGi-Based Semantic Service-Oriented Device Architecture. Lecture Notes in Computer Science, On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, pages 773-782.
- Graham, I., 2006. Business Rules Management and Service Oriented Architecture, John Wiley and Sons, Chichester, England, pages 17-51.
- Grimm, K., 2003. Software technology in an automotive company: Major challenges. Proceedings of the 25th International Conference Software Engineering 2003, pages 498-503.

- Gruber, T., 1995. Toward principles for the design of ontologies used for knowledge sharing. *The International Journal of Human-Computer Studies*, 43(4-5), pages 907-928.
- Haller, A., Oren, E. and Kotinurmi, P., 2006. m3po: An Ontology to Relate Choreographies to Workflow Models. *Proceedings of the IEEE International Conference on Services Computing*, pages 19-27.
- Hanson J., 2005. Event-driven services in SOA, Javaworld. Available at <http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html>.
- Harel, D. and Thiagarajan, P.S., 2003. Message Sequence Charts. In Lavagno, L., Martin, G and Selic, B. (eds.), *UML for real: design of embedded real-time systems*, pages 77-105.
- Herzum, P. and Sims, O., 2000. *Business Component Factory: A Comprehensive Overview of Component-based Development for the Enterprise*, John Wiley and Sons, Inc.
- High, R.Jr., Kinder, S. and Graham, S., 2005. IBM's SOA Foundation: An Architectural Introduction and Overview Version 1.0, IBM white study. Available at <http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitestudy.pdf>
- Hirst, G., 2004. Ontology and the Lexicon. In Staab, S and Studer, R. (eds.), *Handbook on Ontologies*, pages 209-229.
- Huang, H. and Mason, R.A., 2006. Model Checking Technologies for Web Services. *Proceedings of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*.
- Hunter, J., 2003. Enhancing the Semantic Interoperability of Multimedia Through a Core Ontology. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(1), pages 49-58.
- ISO, 2001. International Standard, ISO/IEC 9126, Information Technology - Product Quality - Part1: Quality Model.

- ISO, 2004. Industrial automation systems and integration - Process specification language- Part 1: Overview and basic principles, ISO 18629-1.
- ITU-T, 2004. ITU-T Recommendation Z.120 - Message Sequence Chart, Apr. 2004. Available at <http://www.itu.int/ITU-T/studygroups/com17/languages/Z120.pdf>.
- Karagiannis, D., Junginger, S. and Strobl, R., 1996. Introduction to Business Process Management System Concepts. In Scholz-Reiter, B. and Stickel, E. (Eds.): Business Process Modelling, Springer, pages 81-106.
- Karhunen, H., 2005. Dynamic Method for Service-Oriented Software Design. Proceedings of the 28th Information Systems Research Seminar in Scandinavia, IRIS 28, Kristiansand, Norway. Available at <http://www.hia.no/iris28/Docs/IRIS2028-1034.pdf>
- Karsai, G., 2004. Automotive software: a challenge and opportunity for model-based software development. Proceedings of the First Automotive Software Workshop, ASWSD 2004, San Diego, In Broy M., Krüger, I. H. and Meisinger, M. (eds.) Revised Selected Studys, Lecture Notes in Computer Science 4147, pages 103-115.
- Karsai, G., Sztipanovits, J., Ledeczi A. and Bapty, T., 2003. Model-integrated development of embedded software. Proceedings of the IEEE, 91(1), pages 145-164.
- Kilian, C.T., 2006. Modern control technology: components and systems. Clifton Park, N.Y. : Delmar/Thomson Learning.
- Komoda N., 2006. Service Oriented Architecture (SOA) in Industrial Systems. Proceedings of the 2006 IEEE International Conference on Industrial Informatics, pages 1-5.
- Krafzig, D., Banke, K. and Slama, D., 2004. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR.
- Krüger, I.H., Nelson, E.C. and Prasad, V., 2004. Service-based Software Development for Automotive Applications. CONVERGENCE 2004.
- Kuziemsky, C.E., Lau, F., Bilykh, I., Jahnke, J.H., McCallum, G., Obry, C., Onabajo, A. and Downing, G.M., 2003. Ontology-based information integration in health care: a focus

- on palliative care. Proceedings of the 11th Annual International Workshop on Software Technology and Engineering Practice, pages 164-172.
- Laliwala Z. and Chaudhary S., 2008. Event-driven Service-Oriented Architecture. Proceedings of the International Conference on Service Systems and Service Management, 2008, pages 1-6.
- Lam, W., McDermid, J.A. and Vickers, A.J., 1997. Ten Steps Towards Systematic Requirements Reuse. Requirements Engineering 2(2), pages 102–113.
- Lange, C.F.J. and Chaudron, M.R.V., 2005. Managing Model Quality in UML-based Software Development. Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pages 7-16.
- Lankhorst M., et al., 2005. Enterprise Architecture at Work: Modelling, Communication, and Analysis, Springer Berlin Heidelberg.
- Leen, G. and Heffernan, D., 2002. Expanding automotive electronic systems. IEEE Computer 35(1), pages 88-93.
- Liao, L. and Leung, H., 2007a. An Ontology-based Business Process Modeling Methodology. Proceedings of the International Conference on Advances in Computer Science and Technology (ACST 2007), Phuket, Thailand.
- Liao, L. and Leung, H., 2007b. Testing Techniques for SOA Model. Proceedings of the International Conference on Software Engineering and Data Engineering. (SEDE-2007), Las Vegas, USA.
- Lim, S.S., Park, D.W. and Kwon, H.C., 2007. Ontology-Based Semantic Representation of Context in Port Supply Chain. Proceedings of the 6th International Conference on Advanced Language Processing and Web Information Technology, Pages 446-451.
- Liu, H., Ng, W.K., Song, B., Li, X. and Lu, W.F., 2007. Enabling Mass Customization through Semantic Web Services. Proceedings of the second IEEE Asia-Pacific Service Computing Conference, Pages 239-245.

- Mahleko, B. and Wombacher, A., 2006. Indexing Business Processes based on Annotated Finite State Automata. Proceedings of the International Conference on Web Services, 2006, ICWS '06, pages 303-311.
- Mallya, A.U., Desai, N., Chopra, A.K. and Singh, M.P., 2005. OWL-P: OWL for protocol and processes. Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems, pages 139-140.
- Manolescu, I., Brambilla, M., Ceri, S., Comai, S., and Fraternali, P., 2005. Model-driven design and deployment of service-enabled web applications. ACM Transactions on Internet Technology (TOIT), 5 (3), pages 439- 479.
- MATLAB, 2008. Available at <http://www.mathworks.com/products/matlab/>. Accessed by Sep. 2008.
- Mayer, R., Menzel, C., Painter, M., Witte, P., Blinn, T., and Perakath, B., 1995. Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report. Knowledge Based Systems Inc., September 1995.
- McGovern, J., et al. 2003. Java Web Services Architecture. Morgan Kaufmann.
- Michelson B. M., 2006. Event-Driven Architecture Overview, Patricia Seybold Group. Available at <http://soa.omg.org/Uploaded%20Docs/EDA/bda2-2-06cc.pdf>.
- Microsoft, 2007. Distributed Component Object Model (DCOM) Remote Protocol Specification. Available at [http://msdn.microsoft.com/zh-cn/library/cc201989\(en-us\).aspx](http://msdn.microsoft.com/zh-cn/library/cc201989(en-us).aspx)
- Neema, S. and Karsai, G., 2004. Software for Automotive Systems: Model-Integrated Computing. Proceedings of the First Automotive Software Workshop, ASWSD 2004, San Diego. In Broy M., Krüger, I.H. and Meisinger, M. (eds.) Revised Selected Studys, Lecture Notes in Computer Science 4147, pages 116-136.
- Newcomer, E. and Lomow, G., 2004. Understanding SOA with Web Services. Addison Wesley Professional Pub.

- Niles, I. and Pease, A., 2001. Towards a Standard Upper Ontology. Proceedings of the International Conference on Formal Ontology in Information Systems, pages 2-9.
- OASIS, 2004. UDDI Version 3.0.2, UDDI Spec Technical Committee Draft 19 Oct. 2004, Available at http://uddi.org/pubs/uddi_v3.htm.
- OASIS, 2006. Reference Model for Service Oriented Architecture 1.0, OASIS Standard 12 Oct. 2006. Available at <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html>.
- OASIS, 2007. Web Services Business Process Execution Language Version 2.0, OASIS Standard 11 Apr. 2007. Available at <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
- OMG, 2003. MDA Guide Version 1.0.1. Available at <http://www.omg.org/docs/omg/03-06-01.pdf>.
- OMG, 2004. Common Object Requesting Broker Architecture version 3.0.3. Available at <http://www.omg.org/cgi-bin/doc?formal/2004-03-12>.
- OMG, 2005. Unified Modeling Language: Infrastructure version 2.0. Available at <http://www.omg.org/docs/formal/05-07-05.pdf>.
- OMG, 2007. OMG SysML specifications. Available at <http://www.omgsysml.org/>.
- OMG, 2008. Business Process Modeling Notation, V1.1, OMG Available Specification 17 Jan. 2008. Available at <http://www.omg.org/spec/BPMN/1.1/PDF>.
- Osterwalder, A., 2002. An e-Business Model Ontology for Modeling e-Business. Proceedings of the 15th Bled Electronic Commerce Conference.
- Ouvans, C., Dumas, M., ter Hofstede, A.H.M. and van der Aalst, W.M.P., 2006. From BPMN Process Models to BPEL Web Services. Proceedings of the International Conference on Web Services, 2006, ICWS '06, pages 285-292.
- Pelz, G., Oehler, P., Fourgeau, E. and Grimm, C., 2005. Automotive system design and AUTOSAR. In Boulet, P. (eds.), Advances in Design and Specification Languages for SoCs, pages 293-305.

- Pretschner, A., Broy, M., Krüger, I.H. and Stauner, T., 2007. Software engineering for automotive systems: a roadmap. Proceedings of the International Conference on Software Engineering 2007, pages 55-71.
- Protégé, 2008. Available at <http://protege.stanford.edu/>. Accessed by Mar. 2008.
- Qadir, M.A., Fahad, M. and Shah, S.A.H., 2007. Incompleteness Errors in Ontology. Proceedings of the IEEE International Conference on Granular Computing, 2007, GRC 2007, pages 279-282.
- Rao, A.C., Dhadyalla, G., Jones, R.P. and McMurran, R., 2006. Systems modeling of a driver information system - automotive industry case study. Proceedings of the IEEE/SMC International Conference on System of Systems Engineering, Los Angeles, pages 254-259.
- Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H. and Wroe, C., 2004. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. Proceedings of the 14th International Conference on Engineering Knowledge in the Age of the Semantic Web (EKAW 2004), UK, pages 63-81.
- Rozman, T., Horvat, R.V. and Polancic, G., 2004. Towards true process descriptions interoperability. Proceedings of the 26th International Conference on Information Technology Interfaces, pages 549-554.
- Salam, A.F. and Stevens, J.R., 2007. Semantic Web Technologies and E-Business, Toward the Integrated Virtual Organization and Business Process Automation, Idea Group Publishing.
- Selic, B., 2003a. Modeling quality of service with UML: How quantity changes quality. In Lavagno, L., Martin, G and Selic, B. (eds.), UML for real: design of embedded real-time systems, pages 189-204.
- Selic, B., 2003b. The Pragmatics of Model-Driven Development. IEEE Software, 20(5), pages 19-25.

Simulink, 2008. Available at

<http://www.mathworks.com/products/simulink/>. Accessed by Sep. 2008.

Sommerville, I., 2007. Software Engineering 8th, Addison-Wesley.

Stateflow, 2008. Available at

<http://www.mathworks.com/products/stateflow/>. Accessed by Sep. 2008.

Stojanovic, Z., Dahanayake, A. and Sol, H., 2004. Modeling and design of service-oriented architecture. Proceedings of the 2004 IEEE Conference on Systems, Man and Cybernetics, pages 4147-4152.

Stumme, G., Ehrig, M., Handschuh, S., Hotho, A., Mädche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., Stojanovic, N., Studer, R., Sure, Y., Volz, R. and Zacharias, V., 2003. The Karlsruhe View on Ontologies, Technical report. University of Karlsruhe, Institute AIFB.

Tetlow, P., Pan, J.Z., Oberle D., Wallace, E., Uschold, M. and Kendall, E. (eds.), 2006. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering, W3C Editors' Draft 11 Feb. 2006. Available at <http://www.w3.org/2001/sw/BestPractices/SE/ODA/>.

Therani, M., 2007. Ontology Development for Designing and Managing Dynamic Business Process Networks. IEEE Transactions on Industrial Informatics, 3(2), pages 173-185.

The Open Group, 2009. The Open Group Architectural Framework (TOGAF) Version 9 'Enterprise Edition'. Available at <http://www.opengroup.org/togaf/>.

Torres, V., Munoz, J. and Pelechano, V., 2005. A model driven method for the integration of Web applications. Proceedings of the Third Latin American Web Congress, LA-WEB 2005.

Tran, V.X. and Tsuji, H., 2007. OWL-T: An Ontology-based Task Template Language for Modeling Business Processes. Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management and Applications, pages 101-108.

- Tsai, W.T., Lee, Y.H., Cao, Z.B., Chen, Y.N. and Xiao, B.N., 2006. RTSOA: Real-Time Service-Oriented Architecture. Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE), pages 49-56.
- Uschold, M. and Gruninger, M., 1996. Ontologies: Principles, methods and applications. Knowledge Engineering Review, 11(2), pages 93-155.
- van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros A.P., 2003. Workflow Patterns. In Distributed and Parallel Databases, Springer Netherlands, 14(1), pages 5-51.
- van Dongen, B.F., Mendling, J. and van der Aalst, W.M.P., 2006. Structural Patterns for Soundness of Business Process Models. Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, EDOC '06, pages 116-128.
- von der Beeck, M., Braun, P., Rappl, M. and Schroeder, C., 2003. Automotive UML: a (meta) model-based approach for systems development. In Lavagno, L., Martin, G and Selic, B. (eds.), UML for real: design of embedded real-time systems, pages 271-299.
- Wada, H., Suzuki, J. and Oba, K., 2006. Modeling non-functional aspects in service oriented architecture. Proceedings of the IEEE International Conference on Service Computing, Chicago, pages 222-229.
- Wang, X., Zhang, Y.C. and Shi, H., 2007. Scenario-Based Petri Net Approach for Collaborative Business Process Modelling. Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference, pages 18-25.
- Weber, M. and Weisbrod, J., 2003. Requirements engineering in automotive development: Experiences and challenges. IEEE Software 20, pages 16-24.
- W3C, 2001a. Web Services Description Language (WSDL) 1.1, W3C Note 15 Mar. 2001. Available at <http://www.w3.org/TR/wsdl>.
- W3C, 2001b. DAML+OIL (March 2001) Reference Description, W3C Note 18 Dec. 2001. Available at <http://www.w3.org/TR/daml+oil-reference>.

W3C, 2001c. XML Schema language, W3C Recommendation 2 May 2001. Available at <http://www.w3.org/XML/Schema>.

W3C, 2002a. Web Services Conversation Language (WSCL) 1.0, W3C Note 14 Mar. 2002. Available at <http://www.w3.org/TR/wscl10/>.

W3C, 2002b. Web Service Choreography Interface (WSCI) 1.0, W3C Note 8 Aug. 2002. Available at <http://www.w3.org/TR/wsci/>.

W3C, 2004a. Web Services Architecture, W3C Working Group Note 11 Feb. 2004. Available at <http://www.w3.org/TR/ws-arch/>.

W3C, 2004b. RDF/XML Syntax Specification (Revised), W3C Recommendation 10 Feb. 2004. Available at <http://www.w3.org/TR/rdf-syntax-grammar/>.

W3C, 2004c. OWL: Web Ontology Language semantics and abstract syntax, W3C Recommendation 10 Feb. 2004. Available at <http://www.w3.org/TR/owl-semantics/>.

W3C, 2004d. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 Feb. 2004. Available at <http://www.w3.org/TR/rdf-schema/>.

W3C, 2004e. OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004. Available at <http://www.w3.org/Submission/OWL-S/>.

W3C, 2005a. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation 9 Nov. 2005. Available at <http://www.w3.org/TR/ws-cdl-10/>.

W3C, 2005b. Web Service Semantics - WSDL-S Version 1.0, W3C Member Submission 7 Nov. 2005. Available at <http://www.w3.org/Submission/WSDL-S/>.

W3C, 2006. Extensible Markup Language (XML) 1.1, W3C Recommendation 16 Aug. 2006. Available at <http://www.w3.org/TR/xml11/>.

W3C, 2007. SOAP Version 1.2, W3C Recommendation (Second Edition) 27 Apr. 2007. Available at <http://www.w3.org/TR/soap/>.

- W3C, 2008a. OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax, W3C Working Draft 8 Jan. 2008. Available at <http://www.w3.org/TR/owl11-syntax/>.
- W3C, 2008b. SPARQL Query Language for RDF, W3C Recommendation 15 Jan. 2008. Available at <http://www.w3.org/TR/rdf-sparql-query/>.
- Yin, R. 2003. Case Study Research: Design and Methods (3rd Edition). Newbury Park, CA: Sage
- Yu, C., Wu, G.Q. and Yuan, M.T., 2005. Business process modeling based on workflow model reuse. Proceedings of the International Conference on Services Systems and Services Management, 2005, ICSSSM '05, Vol. 2, pages 951-954.
- Zdun, U., Hentrich, C. and Dustdar, S., 2007. Modeling process-driven and service-oriented architectures using patterns and pattern primitives. In ACM Transactions on the Web (TWEB) , 1(3), pages 14-43.
- Zhao Y., 2006. Enterprise Service Oriented Architecture (ESOA) Adoption Reference. Proceedings of the IEEE International Conference on Services Computing, page 512.
- Zhou, N.J., Chee, Y.M. and Zhang, L.J., 2008. Coding-Free Model-Driven Enablement Framework and Engineering Practices of a Context-Aware SOA Modeling Environment. Proceedings of the IEEE International Conference on Web Services, 2008, ICWS '08, pages 553-560.
- zur Muehlen, M. and Ho, D.T., 2008. Service Process Innovation: A Case Study of BPMN in Practice. Proceedings of the 41st Annual Hawaii International Conference on System Sciences, pages 372-382

Appendix A: Glossary

Activity	An activity refers to a business task where some sort of business function is carried out.
AMI-C	Automotive Multimedia Interface Collaboration
AML	Automotive Modeling Language
AM-RPRS	Agile Modeling based on Reusable Process and Reusable Services
API	Application Programming Interfaces
AutoPO	Automotive Process Ontology
AUTOSAR	AUTomotive Open System ARchitecture
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
BPO	Business Process Ontology
BUM-RS	Bottom-Up Modeling based on Reusable Services
Business function	Business function defines what a system is designed to perform.
Business logic	Business logic is a non-technical term to describe the functional algorithms of processes and services.
Business process	A business process is the abstraction of an activity, which describes the model design for an activity. It also called process in this thesis.
CORBA	Common Object Requesting Broker Architecture
CRM	Customer Relationship Management
DAML+OIL	DARPA Agent Markup Language and Ontology Interchange Language
DCOM	Distributed Component Object Model
ECU	Electronic Control Unit
ERP	Enterprise Resource Planning
IDEF3	Integration DEFinition Language

ISO	International Standards Organization
MDA	Model Driven Architecture
MIC	Model-Integrated Computing
MSC	Message Sequence Charts
m3po	multi meta-model process ontology
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OBPMDF	Ontology-based Business Process Modeling and Developing Framework
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OWL	Web Ontology Language
OWL-P	OWL for Processes and Protocols
OWL-S	OWL for Services
OWL-T	Task ontology language
PIM	Platform-Independent Model
P-port	Provide-port
Process	Short for Business Process
PSM	Platform-Specific model
PSL	Process Specification Language
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	RDF Schema
R-port	Require-port
RTE	Runtime Environment
RTSOA	Real Time SOA
Service	In SOA, service is considered as a self-contained and self-describing business driven functional unit and can be invoked across networks to

provide flexible enterprise application integration.

In this thesis, “Service” also represents a basic class in BPO. For this meaning, we name it as “the Service class”.

SeSODA	Semantically-enabled Service-Oriented Device Architecture
SOA	Service-Oriented Architecture
SOAP	Simple Object-based Access Protocol
SODA	Service-Oriented Device Architecture
SOF	Slight Ontology Framework
SPARQL	Simple Protocol And RDF Query Language
SysML	Systems Modeling Language
Task	Task is a piece of specific work to be performed in the real world.
TDM	Top-Down Modeling
TDM-RP	Top-Down Modeling based on Reusable Process
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
VFB	Virtual Functional Bus
WSBPEL	Web Services Business Process Execution Language
WSCDL	Web Services Choreography Description Language
WSCI	Web Services Choreography Interface
WSCL	Web Services Conversation Language
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Appendix B: Framework of AutoPO

