TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Anna Ruokonen

Scenario-Driven Development of Service-Oriented Systems

Julkaisu 1102 • Publication 1102

Tampere 2012

Tampereen teknillinen yliopisto. Julkaisu 1102
Tampere University of Technology. Publication 1102

Anna Ruokonen

Scenario-Driven Development of Service-Oriented Systems

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB222, at Tampere University of Technology, on the 14th of December 2012, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2012

ISBN 978-952-15-2975-7 (printed)
ISBN 978-952-15-2979-5 (PDF)
ISSN 1459-2045

Abstract

Service-oriented architecture (SOA) is a software architectural style, which relies on reusable and composable services. In addition to software-orientation, SOA includes a business viewpoint, so that business requirements can be captured as high level business processes. Business processes can be implemented, for example, as an orchestration of different service components. Individual services participate in the overall execution of business processes by providing elementary service activities. In addition to flexibility and services reuse, bridging of business and information technology (IT) views is one of the claimed benefits of SOA.

Development of service-based systems includes a range of different activities. However, development of service-based systems is still lacking systematic and tool vendor independent practices and development methods. In this thesis, a development process for a service provider, called Service Product Development Process (SPDP), is presented. It consists of several development phases and related activities. The input for SPDP includes high level business process requirements. The result of the process is a new service-based product to be added to the service provider's product portfolio.

The purpose of this thesis is to study the applicability and the benefits of applying a scenario-driven approach, a type of requirement-driven development, for the development of service-based systems. Scenarios are used to capture functional system requirements as simple message sequences given as UML sequence diagrams. The scenario-driven approach is applied to different phases of SPDP including business process development, service specification, and service realization. The proposed scenario-driven approach is not limited to the SPDP context. It is rather a general purpose framework for development of service-based systems or products, called SceDA.

SceDA includes three independent scenario-based methods, which are targeted to support different development phases of service-based systems. One of the three methods is used for scenario-based business process development. The other two methods are targeted at service development, in particular, service specification and service realization. Service specification is supported by a method for automatically mining and re-documenting the development rules as scenarios. To support service realization, a method for generating source code for individual service and client applications has been developed. Each method includes a description of the developed tool support and a case study.

Case studies are used for constructing and evaluating the three scenario-based methods developed. Each method is applied as a case study in the context of development phases of SPDP. In the first case study, scenario-driven business process development method is applied. Two other case studies concern constructing and using scenarios for application development. One case study utilizes the scenario mining method. In the other case study, the code generation method is applied.

Keywords: SOA, Web service, business process, UML, model-driven development, system requirements, development process for service-oriented systems, scenario-based development, scenario synthesis, scenario mining

Preface

First of all, I want to thank my supervisor Professor Tarja Systä for her support, guidance, and ideas. Special thanks to Professor Kai Koskimes and Johannes Koskinen for reading and commenting the thesis manuscript. I would like to thank my colleagues and co-authors: Juanjuan Jiang, Johannes Koskinen, Mika Siikarla, Imed Hammouda, Timo Kokko, Lasse Pajunen, Vilho Räisänen, Mikko Hartikainen. In addition, I want to thank Alan Thomson for proofreading the thesis. Thank you for the pre-examiners, Dr. Scott Tilley and Dr. Juha Mykkänen, for reviewing the manuscript.

This work has been carried out in two research projects, EEWES (An Engineering Environment for Web Services) during 2004-2006 and MoDES (Tool Support for Model-Driven Engineering of Service Compositions) during 2007-2010. The projects have been funded by the Finnish Funding Agency for Technology and Innovation (TEKES), Nokia Research Center, Nokia Siemens Networks, Plenware, and Solita. In addition, this work has been funded by Graduate School on Software Systems and Engineering (SoSE), The Nokia Foundation, Ulla Tuomisen säätiö, and Finnish Foundation for Technology Promotion.

Mika and Jomppa, thanks for being such good company when sharing the room and plants for several years.

I want to thank all my friends. Special thanks to my cycling and running mates, outdoors and indoors. Thank you for making all the social events to promote health and fitness:) Thanks to my traveling mate Tanja for all the many adventures. My MTB mechanics team, T&H, thank you for assembling so many light-weight bike parts, building the wheels and so on. You are great!

I want to thank my son Aatos, Suvi, Kaitsu, and my parents for all the support during these years. Special thanks to my mother for taking care of Aatos while I've been traveling.

Contents

List of Figures

List of Included Publications

[I] A. Ruokonen, V. Räisänen, M. Siikarla, K. Koskimies and T. Systä Variation Needs in Service-Based Systems, The 6th IEEE European Conference on Web Services. In *The 6th IEEE European Conference on Web Services (ECOWS 2008)*, pp. 115 - 124, Dublin, Ireland, 2008.

[II] A. Ruokonen, T. Kokko, and T. Systä, Scenario-Driven Approach for Business Process Development. *International Journal of Business Process Integration and Management (IJBPIM)*, vol. 6(2012) No 1, 20 pp.

[III] L. Pajunen, and A. Ruokonen, Modeling and Generating Mobile Business Processes. In *IEEE 2007 International Conference on Web Services (ICWS 2007)*, pp. 920-927, Salt Lake City, Utah, USA, 2007.

[IV] J. Jiang, J. Koskinen, A. Ruokonen, and T. Systä, Constructing Usage Scenarios for API Redocumentation. In *The 15th IEEE International Conference on Program Comprehension (ICPC 2007)*, pp. 259 - 264, Banff, Alberta,Canada, 2007.

[V] J. Koskinen, A. Ruokonen, and T. Systä, A Pattern-Based Approach to Generate Code from API Usage Scenarios. In *Nordic Journal of Computing (NJC'06)*, vol. 13(2006), pp. 162- 179.

The permissions of the copyright holders of the original publications to reprint them in this thesis are hereby acknowledged.

Part I

Introduction

Chapter 1

Introduction

In this chapter, the context and motivation for this study is explained. In addition, research questions and research methods are presented. A summary and an outline of the thesis are also presented.

1.1 Motivation

Service-Oriented Architecture (SOA) defines a software architectural style, which is comprised of a set of services [43]. A service is a software component providing a service interface for communication with other services. Furthermore, SOA is business-oriented, meaning that services are capable of supporting execution of business activities. By executing such business activities services contribute to so called business processes. SOA and service-based systems have been suggested as a means of efficiently reusing and sharing distributed capabilities within or between different organizations [42]. It is also claimed that services, which can be composed into high level services and business processes, are able to bridge the gap between business and IT [8].

The development of a service-based system can be started by defining high level services and business processes, in a top-down approach, or from the existing code base, called a bottom-up approach. Ideally, in a top-down approach the required service and business processes are defined from the business requirements without considering the existing code base. If possible, existing services are reused in the business processes otherwise new services need to be implemented. In a bottom-up approach, services are formed by analyzing the existing code base. Service realization often includes use of some wrapping techniques to create proper service interfaces. Practical cases have shown that for a high-quality SOA project, a combination of the two approaches is often the most successful [4].

However, SOA is still lacking systematic and tool vendor independent development methods. In addition, different stakeholders even within the same company, for example software architects and business analysts, might have a different outlook on SOA [35]. In SOA, business process activities are mapped with individual services, either to existing ones or to new ones. That makes services identification an essential development activity. From a software engineering point of view, that often includes reusing some existing legacy code. From the business point of view, services identification should be driven by long-term business needs instead of the existing code base. Thus, G. Cotticchia proposes that a collaborative and iterative top-down development method should be used [14].

In practice, the functional business requirements are often given as high level use cases. Use cases can be expressed as scenarios consisting of a sequence of actions given as graphical and/or textual descriptions. One reason for this is the intuitiveness of scenarios; they are easy to understand and construct even for not-IT oriented people. Use of simple example-like scenarios does not require use of control structures and state-oriented modeling. As drawback, distinct scenarios do not capture the overall behavior of the system under development. Usually, the process modeling is assumed to be done using some workflow or business process modeling language, such as WS-BPEL. Thus, the functional requirements need to be rewritten in a form of business process notation. To achieve an executable business process description, however, requires technical skills. In general, modeling of the business processes is seen as one of the main challenges in adopting SOA [35]. In this thesis, the aim is to provide support for automating creation of an initial business process model from functional system requirements. The proposed solution includes development of a scenario-based method and tool support.

At the implementation-level, service development often includes using some existing APIs or software libraries. Exploiting existing software requires knowledge of its correct utilization. In addition to textual descriptions, this knowledge can be documented, for example, as typical usage and code examples in a developer guide or tutorial. In a similar fashion, rules for application development, in general, can be given. In the context of this study, scenarios are proposed as a means of capturing such development rules.

A reason for using scenarios for presenting system requirements and development rules is that they are seen as intuitive and easy to use, especially for people who are not IT-oriented. In most of the cases, however, construction and exploitation of the scenarios still remains as a manual task. In this thesis, a systematic mechanism to construct and apply such scenarios in development of service-based systems, including development of business processes and individual services, is presented.

1.2 Research approach

In this section, the research approach, including the research questions and methods, is presented.

1.2.1 Research questions

The main motivation for the study was to develop a method for sketching and applying functional system requirements in the context of service-based systems. The purpose of this thesis is to study how the scenario-driven approach can be applied to development of service-based systems. The aim is to develop practical methods and tools to support the development process. The developed methods include requirements engineering, model-driven engineering, re-documentation, and code generation. The study includes use of reverse-engineering and scenario synthesis techniques. However, exhaustive reverse-engineering techniques and rigorous SOA challenges are not in the focus of this thesis. Table 1.1, summarizes different themes discussed in the thesis. The column headings refer to included publications.

Theme	[I]	[II]	[III]	[IV]	[V]
Requirements-driven development		x			x
Model-driven engineering		x	x	x	x
Development process for service-based systems	x				
Scenario mining				x	
Scenario synthesis		x		x	
Process/service development	x	x	x	x	x
Re-documentation				x	
Tool support		x	x	x	x

Table 1.1: Themes of scenario-driven development acknowledged in the publications

The research questions studied are as follows:

RQ1 What activities are involved in a typical development process for service-based systems, and what kind of methods and tool support can be built to support the development activities? (Publications [I, II, III, IV, V])

RQ2 How can the scenario-driven approach be applied to business process development? (Publications [II, III])

RQ3 How can the scenario-driven approach be applied to services specification and re-documentation? (Publication [IV])

RQ4 How can the scenario-driven approach be applied to realization of individual services, i.e. service implementation? (Publication [V])

The research questions RQ1-RQ4 have been studied by applying a constructive research approach. The research method is presented in the next section.

1.2.2 Research method

Constructive research aims at producing novel solutions to relevant and practical problems [28]. In this thesis, constructive research methods are applied by constructing development methods and tool support as well as by conducting case studies.

Constructive research starts from identifying the context and problems to be studied [28]. Motivation for this thesis includes providing an intuitive starting point for development of services systems. According to our experience and current practice, people tend to draw scenarios of system behavior in order to understand, re-document or document complicated systems. Scenarios are structured and still understandable mechanisms for capturing requirements. The needs originate partly from our experiences and partly from our industrial partners' and their end-users' interests. The end-users, in particular, would benefit from easy sketching and composing of their business processes. Moreover, scenario-based modeling could employ a light-weight modeling approach for mobile users in the future. To emphasize, this is not the current situation, but one of the motivating factors for this study.

A case study is an empirical method aimed at studying contemporary phenomena in their natural context [77]. The primary objective of case study research methodology is exploratory, i.e., finding out what is happening, seeking new insights and generating new ideas and hypotheses for new research [55]. Research on software engineering is often aimed at investigating *how* software development is conducted by different stakeholders under different conditions. Thus, the research questions involved are often suitable for case study research. Conducting a case study should include the following phases (1) Case study design and planning, (2) Collecting data, (3) Data analysis, and (4) Reporting. In [55], Runeson *et al.* propose a structure for case study reporting. In this study, the reports have the following structure (modified from [55]).

- **Case study introduction**
  - **Problem statement and context**
  - **Case study design**
- **Execution**
- **Case study results and conclusions**
  - **Analysis**
  - **Conclusions**

The structure of the case study report is divided into three parts. The first part includes the description of the target system, the problem statement, and the case study design. The design includes the research questions and describes the steps required to conduct the case study, including analysis of the results. The case study design is presented independently of the target system to enable repetition in a different context. In the second part, a description of the actual case study execution is presented. The third part includes analysis and interpretation of the results. In addition, a summary of the case study conclusions is presented.

The overall research approach is presented in Figure 1.1. As shown in the figure, the context of the research is development of service-based systems. The iterative research approach starts with identifying the different development activities and related challenges. The next two steps include a constructive study of how to apply the scenario-driven approach to the particular development phase. The study includes an initial hypothesis, which is refined by conducting the case study. As a result, a description of the applied scenario-based method is given. From this research process, three methods have been developed. In addition, the development phases are identified and related activities are documented as a development process for the service-based systems called SPDP. The complete outcome of the study is the scenario-driven approach, SceDA, presented on the right of Figure 1.1.

RQ1 is addressed during the overall research process (Figure 1.1). RQ2, RQ3, and RQ4 are studied by constructing scenario-based methods and applying case studies. In the context of each case study, the related research question is refined into more specific sub-questions. In the following, the methods developed and the case studies conducted are presented.

M1,CS1	Scenario-based method for business process development: A case study for an industrial logistics provider on applying the scenario-based method for business process development has been conducted. (RQ2)

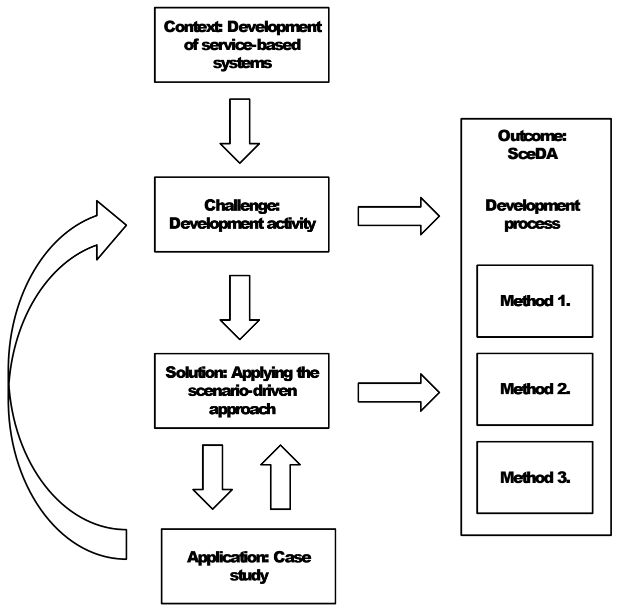

Context: Development of service-based systems
Challenge: Development activity
Solution: Applying the scenario-driven approach
Application: Case study
Outcome: SceDA
Development process
Method 1.
Method 2.
Method 3.

Figure 1.1: The research approach

M2,CS2	Scenario mining method for service specification and re-documentation: A case study on mining scenarios from application traces is conducted. The case study includes construction of development rules for application development. (RQ3)
M3,CS3	Scenario-based method for service realization: A case study using scenarios for generating application code is carried out. (RQ4)

1.3 Contributions

This thesis consists of five publications and an introductory summary of the publications. As a result of following the constructive research method, answers to the research questions presented in Section 1.2 are given.

The study includes identification of typical development phases of service-based systems. The outcome is documented as a development process called SPDP. Three scenario-based methods, which can be applied in different development phases of the service-based systems, have been developed. The development process and the three methods constitute the scenario-driven approach for development of service-based systems, called SceDA.

This thesis includes the following: (1) The scenario-based method for business process development, (2) The scenario mining method for service specification and re-documentation, (3) The scenario-based method for service realization, (4) identification of the different development phases of service-based systems, (5) examination of three case studies, (6) development of tool support.

In addition, the scenario-based methods developed contribute to the development process of services systems by promoting propagation of the requirements and providing systematic methods for business process and service development.

As a result of the study, a thesis statement can be formulated as follows: *Scenario-based methods can be applied in different development phases of service-based systems, including business process and service development. By providing a systematic approach the scenario-driven development promotes requirement propagation and better integration of different phases of the development process. In addition, scenario-based methods can be used to improve system maintainability through a better understanding of the system behavior and usage.*

Besides the included publications, the author has co-authored the following supporting publications:

- in [23], Hammouda *et al.* propose an approach and tool support to apply a generative modeling mechanism. The approach is based on capturing design rules as reusable patterns using the INARI tool,
- in [56], Ruokonen *et al.* present an approach and tool support for ensuring consistency in model-driven development by enforcing requirements propagation,
- in [29], Jiang *et al.* study variability needs and management in service-oriented systems. The study provides pattern-based support for managing variability of service interfaces and specializing service frameworks,
- in [57], Ruokonen *et al.* study model-driven development of mobile business processes. In the paper, requirements for adaptable and user-centric mobile business processes have been studied.

1.4 Outline of the thesis

This thesis is divided into four parts. In the following section, a short summary of each of the parts is presented.

Part I is called *Introduction.* In Chapter 2 essential background technologies and tools are presented.

Part II, *SOA and business process development*, is covered in Chapters 3-5. In Chapter 3, a development process for a service provider is presented. In Chapter 4, a scenario-based method for business process development is defined. In Chapter 5, the proposed method is applied as an industrial case study to support migration into a business process based system implementation.

Part III, *Service development*, is covered in Chapters 6-8. In Chapter 6, a scenario mining method for services specification and re-documentation is defined. In Chapter 7, a method and tool support for scenario-based service realization is presented. In Chapter 8, the presented methods are applied to two case studies. The case studies include mining of API rules as scenarios and applying the constructed scenarios for generation of source code.

Part IV, *Related work and conclusions*, includes Chapters 9-11. In Chapter 9, related research is discussed. In Chapter 10, a summary of the included publications, including the authors contributions, is given. Finally, Chapter 11 concludes the study and provides ideas for future research.

Chapter 2

Background

This chapter include a short introduction to Service-Oriented Architecture (SOA). In addition, established software development methods, which are utilized in the thesis and which can be considered as originators for the scenario-driven approach, are presented. Related background technologies, the relevant parts of UML and scenario synthesis, are also presented, as well as existing tools.

2.1 Requirements engineering

A software system is built for a particular purpose and specific needs. Still, software systems have traditionally suffered from a mismatch between the system and its operating environment needs [12]. For example, from business point of view a service-oriented system might be understood in terms of services, business tasks, and business processes, whereas from an IT point of view the system is seen as a collection of programming concepts such as software components and interfaces. The development method has been traditionally driven by the programming paradigm rather than the environmental requirements. This mismatch is one of the reasons which cause quality problems in software systems. As a solution, requirements-driven development has been proposed as a means to reduce the gap between the system requirements and the system being developed [12].

System requirements should be presented in a structured way but should still be understandable by the system users. For requirements modeling, use cases are often employed. Use cases are described in terms of actors and the system activities. An actor is an outside entity that interacts directly with the system. Typically each use case focuses on some particular purpose that the actor wants to achieve. The system is considered as a black box and its internal behavior is ignored. [54]

Use cases can be refined with scenarios to specify the actual execution sequences. For example, Figure 2.1 presents a simple use case and a scenario. In this thesis, scenarios are used to model functional system requirements. Related discussion on requirements propagation is included in Section 2.4.

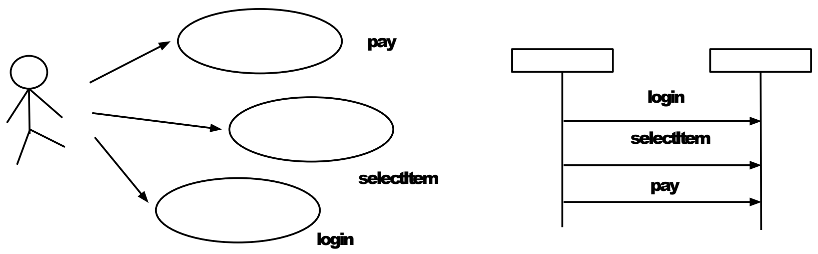


Figure 2.1: A use case and a scenario

2.2 SOA and service-oriented development

Service-Oriented Architecture (SOA) is a business-driven IT architectural style that supports integration of the reusable services into business tasks. A business task typically satisfy a certain business rule or requirement. In SOA, *services* are a means to bring needs and capabilities together. People and organizations can offer capabilities as services by acting as *service providers.* Entities with a need to make use of services are referred to as *service consumers.* Services are promoted through a service description which contains the information necessary to interact with the service. Service providers publish their service descriptions, for example, in a service registry. The service consumers need to discover the proper services and use the service description to invoke the actual service.

SOA aims at loosely coupled services. One of the main principals for SOA is that the services should be reusable. Also, it should be possible to coordinate and compose services to form new composite services and workflows. Thus, the development of service-based systems often involves utilization of existing systems or their parts [37, 61, 66].

W3C Web Services Architecture Working Group [75] defines SOA as follows:
Service Oriented Architecture is an architectural style whose goal is to achieve loose coupling among interactive software agents. A service is a unit of work done by a service provider to achieve the desired end results of a service con-

sumer. This definition of SOA is one of the most commonly used. Bieberstein *et al.* defines SOA *a framework for integrating business processes and supporting IT infrastructure as secure, standardized components or services that can be reused and combined to address changing business priorities* [8].

As stated in the latter definition, one of the main aims of SOA is that it bridges business and information technology oriented views, i.e. that business requirements can comfortably be transformed into service workflows, so called business processes. The description of a workflow of Web service activities that together provide a certain business value, such as a business rule, is called an *orchestration* or a *choreography.* A choreography is a description of service interactions which define a meaningful conversation. Control of the choreography is distributed throughout the participants. Contrary to choreography, the control of an orchestration is owned by a single mediator that manages the interaction between the participants. Furthermore, an orchestration relates to the execution of specific business processes (i.e. the process is the mediator), whereas a choreography is an externally observable multi-party collaboration. Orchestrations can also have a normal service interface such that they appear as services.

Functionality of a typical service-oriented system is often characterized by high level business processes. Business processes interact with services, which provide activities contributing to the business process. Services and processes interact through their interfaces. Figure 2.2 illustrates a simplification of SOA architecture including business processes, services, and component layers. Service components are the actual service implementations exposed as services. Services can be either atomic or realized as compositions of several services, called composite services. Services can participate in workflows to form meaningful business processes. Typically, business processes and services form a hierarchy, such that higher level services make use of lower level services. Services at the lowest level are fine grained providing resources and infrastructure services to be utilized by the other services. High level services are coarse grained business services and they are used by the business processes and end users. These service layers are also referred as granularity layers. Service granularity refers to the service size and the scope of functionality provided by an individual service. The services should have the right granularity to accomplish a proper unit of work as well as to enable service reusability and composability [36].

Service-oriented development contains several development steps. Service-Oriented Modeling and Architecture (SOMA) by IBM [4, 5] has been developed to support and document service-oriented development in a form of reference architecture. It defines different development steps including iden-

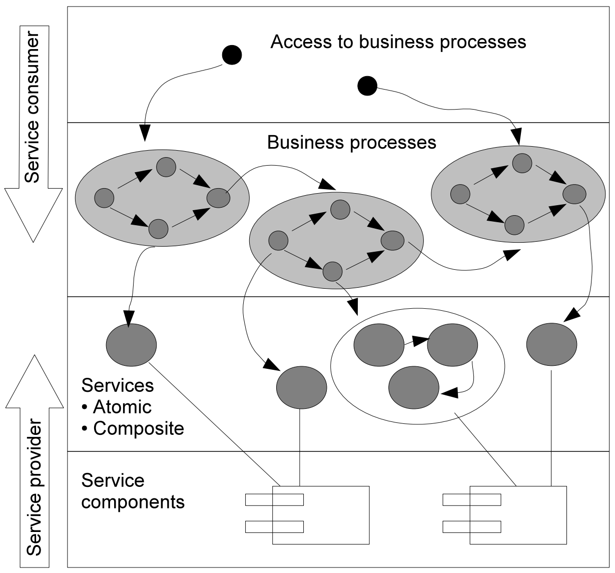


Figure 2.2: SOA architectural layers [4]

tification, specialization, and realization of services that can be used to form composite services and business processes.

The term ”business process” can be understood in many ways. However, in the context of this thesis it is defined as follows: *A business process is a service orchestration, which constitutes a meaningful workflow of business task or activities.*

2.3 Web services and WS-BPEL

Web services offer a way of implementing SOA. Web services can be built using the following standards: Web service description language (WSDL) [73] and Simple Object Access Protocol (SOAP) [74]. A WSDL description defines a Web service interface including service operations, data and message types, and bindings to a transportation protocol. The client uses the WSDL description to invoke the service. SOAP is a message format for message exchange used with Web services.

WS-BPEL is an XML-based business process language for defining executable Web service orchestrations. A WS-BPEL description defines the logic for services interaction, so called invocation rules. Interaction between Web services is enabled through their WSDL descriptions [73]. The WS-BPEL process itself is also a Web service with a WSDL description. Thus, it appears as a service and it can be a part of a composite service as well. A

WS-BPEL process can be run in a specific process engine, which is responsible for receiving invocation from external services as well as invoking other services.

WS-BPEL is a rather complicated XML-based language. In practice, use of some graphical workflow modeling notation, such as Business Process Modeling Notation (BPMN) [48] or Unified Modeling Language (UML) [47] activity diagrams is required. Various model transformation approaches and tools have been proposed to transform such graphical descriptions into executable workflow descriptions [10, 27, 31, 32, 38]. With different modeling tools, the amount of required technical details varies. The focus at business process notation level should be on business requirement modeling rather than on technical details.

2.4 Model-driven engineering

In Model-Driven Engineering (MDE) models are used as the most important artifacts leading the software development processes [45]. Usually, new system artifacts are produced through model transformations. The main idea in MDE is to start with high level models without technical details. The models are refined by adding more details and further transformed into more detailed implementation-specific models. Usually, MDE is applied as an iterative process and it includes, for example, model-to-model and model-to-code transformations. Finally, the ultimate goal is to produce executable models, code, or descriptions.

Model transformations are defined as mapping between source and target elements. Such mappings are also called transformation rules. The model transformations can include automatic, semi-automatic, and manual tasks. Typically, interpretation of the models, is built in the transformations. Some model transformation approaches also include user interaction [60, 72].

Model-Driven Architecture (MDA) is OMG's approach for MDE [45]. It emphasizes three types of abstraction levels for a model: computation independent models (CIM), platform-independent models (PIM), and platform-specific models (PSM). A CIM can also be referred to as a business or domain model. It presents the expected system behavior, but hides all the technological and implementation-specific issues. A PIM is an abstract description of the system, which is detailed enough to enable mapping to one or more platform. For example, it can be defined as a set of required services without technical details. A PSM refines the specification given in the PIM with particular platform-specific concepts. In model transformations, requirements are propagated from higher abstraction levels into lower levels, namely from

CIM to PIM and from PIM to PSM. In our earlier work, we have studied consistency and requirements propagation in MDA [56]. However, this study is not restricted to the MDA approach, but we apply MDE, including modeling and model transformations, at a more general level.

2.5 UML

Unified Modeling Language (UML) [47] is a standard general-purpose modeling language developed for software engineering purposes. The standard is created by the Object Management Group (OMG). UML combines different modeling techniques, including class modeling, state modeling, object modeling, flow modeling, and component modeling. It can be used throughout the software development life cycle to support modeling of different technologies.

UML is based on Meta Object Facility (MOF) [49]. MOF defines the common meta-model for UML, which enables model integration and exchange. The MOF meta-model defines the UML elements and the structure of UML.

UML includes several diagrams types. Structural diagrams, such as class diagrams, are targeted for capturing structural aspects of the system. Behavioral diagrams are used for capturing the system behavior and interaction. UML defines several diagram types for modeling system behavior, such as use case diagrams, state machine diagrams, activity diagrams, and sequence diagrams. From the point of view of this thesis, the most relevant diagram types are presented in this section.

2.5.1 UML activity diagrams

UML activity diagrams are used to describe the workflow behavior of a system, including system logic and business processes. The system behavior is defined by activities performed by the different parties involved in the system. UML activity models present the activity flow through the system described in terms of activities, activity edges, and control nodes. Control flow edge is used to pass control from one activity to another. Control nodes, which include initial, final, join, merge, and decision nodes, are used to coordinate the flows. UML activity diagrams are very close to UML state machine diagrams, i.e. activities correspond to actions in state diagrams. [46]

Figure 2.3 presents a simple workflow as a UML activity diagram. It consists of activities, control flows and control nodes. Control flows are presented as edges and activities are presented as by rounded rectangles. A diamond indicates a decision node to support forking of the control flow. Control flows are merged again in a join node, indicated by as a horizontal gate. The workflow defines a mobile forecast service (*getForecast*) including two alternative payment options, namely charge credit card (*chargeCredit-Card*) or charge in a phone bill (*chargeSim*). As a result, the service sends the forecast information to the customer (*returnForecast*).

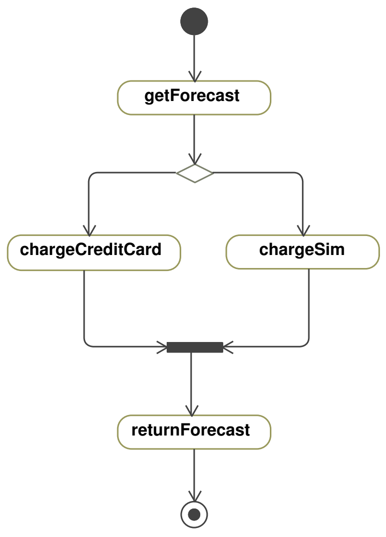


Figure 2.3: UML Activity diagram

2.5.2 UML sequence diagrams

UML sequence diagrams [46] are used to present interactions between objects in a sequential order. To express behavioral system requirements as scenarios, we use two types of UML sequence diagrams: (i) simple sequence diagrams that only consist of participants and messages sent and received between them, and (ii) sequence diagrams with control structures. Simple sequence diagrams are a subset of UML sequence diagrams that consists only of objects, shown as vertical lines called lifelines, and messages as horizontal arrows from a sender object to a receiver object. In the context of this thesis, they are called *simple scenarios*. Sequence diagrams that can contain control structures are called *scenarios*.

In UML sequence diagrams, a message is presented as a simple message *fragment*. The combination of fragments is called a *CombinedFragment* (see Figure 2.4). There are several types of combined fragments, which can be used to express control structures. For example, an *optional* fragment can be optionally executed once, an *alternative* fragment includes two or more alternative execution paths, and a *loop* fragment can be executed multiple times.

The fragment type is specified by a special operator called *InteractionOperator*. Different types of combined fragments are used to capture variations occurring in the message exchange. The UML sequence diagram introduces an *InteractionOccurence* element, which is a placeholder for another interaction sequence to be executed. It allows production of hierarchical and structured interactions.

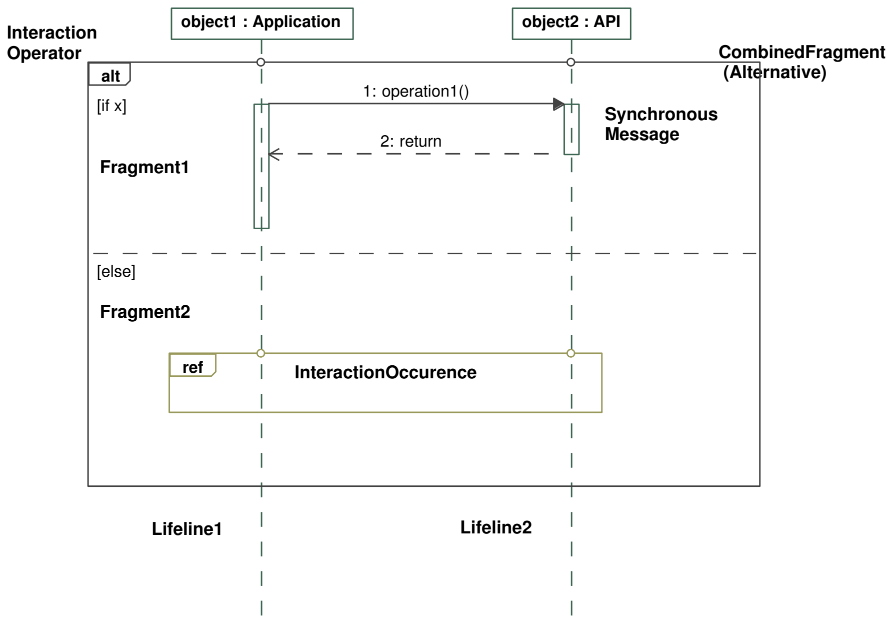


Figure 2.4: A combined fragment

2.5.3 UML profiles

UML is a general purpose modeling language. UML profiles provide a mechanism for adapting UML for a particular domain, platform, or method. UML profiling mechanism can be used as a lightweight extension to the standard UML. A profile defines a subset of UML model elements extended by stereotypes and tagged values. In addition, a profile can express rules of "well-formedness" for specifying the validity of the models. [46]

In particular, UML profiles can be used to support model-driven development in a particular context, for example, for creating new model elements as well as for validating existing design models [23].

Domain-specific UML profiles have been developed and published for several purposes. For example, IBM has proposed the Draft UML 1.4 Profile for

Automated Business Processes with mapping to BPEL 1.0 [2]. The profile is not complete but gives an idea of how to use UML for modeling executable WS-BPEL based business processes [41].

A UML profile defines new class types as meta-classes. In addition, meta-classes can be attached with constraints. For example, a profile for a simple client-service type of architecture could be defined by three stereotypes: *client*, *service*, and *use*. As shown in Figure 2.5, the two former stereotypes can be attached to UML classes and the latter is a particular case of UML dependency between the two classes. At the bottom of the figure is a design model, which follows the given UML profile.

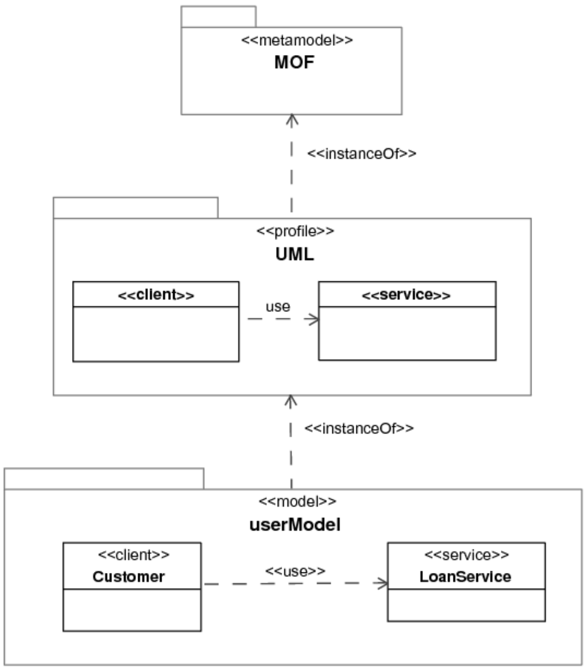


Figure 2.5: A UML profile and a model of client-service architecture

2.6 Scenario synthesis

Scenarios are often used to model functional system requirements. Thus, a scenario presents one desired interaction sequence throughout the system, similar to an execution trace. In both cases, however, that is only a partial view of the system behavior. To simulate the overall system behavior, the scenarios need to be merged. This can be done with a state machine synthesis approach. State machine synthesis from example traces has been widely studied and several approaches exist (e.g. [9, 24, 67, 76, 78]). In this section, the basic idea of the scenario synthesis is presented. The idea presented is an interpretation of the scenario synthesis problem used in this study.

2.6.1 Scenario synthesis problem

A scenario is presented as an interaction consisting of participating objects and messages sent and received between them (presenting service calls). For instance, on the left of Figure 2.6, one possible scenario constructed for a *LoanManager* process is presented. The process administers customers' loan requests and interacts with *Customer*, *RiskAssesment*, and *Approver* services.

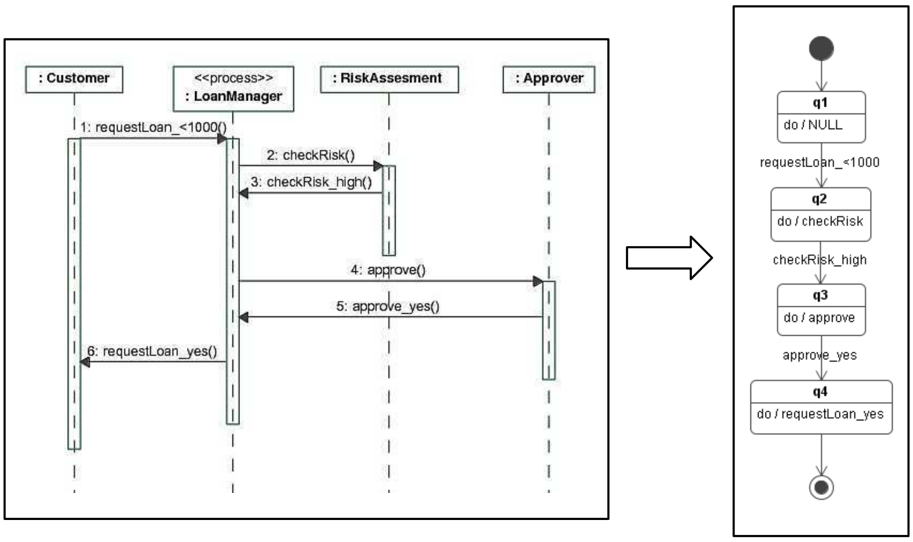


Figure 2.6: Transforming a scenario into a state machine

A scenario can be interpreted as a sequence of message pairs *(sent_message, received_message)*. If either of the messages is missing, it is defined as *NULL*. *VOID* presents the end of the scenario. For example, the *LoanManager* scenario can be expressed by the following message sequence:
(NULL,requestLoan<1000)
(checkRisk,checkRisk_high)
(approve,approve_yes)
(requestLoan_yes,VOID).

sent_message is a message sent by the *LoanManager* object and *received_message* is a message received by the *LoanManager* object. Thus, the message sequence constructed is viewed from a certain angle, i.e. the *LoanManager* object.

The message sequence defines one possible interaction trace. For example, the expected behavior of the *LoanManager* is one of five possible scenarios, each defining a decision on the loan request based on different criteria. To construct the overall behavior, all the scenarios are synthesized into a state machine. The construction of a state machine is done based on certain rules, which might be different depending on the approach. In the context of this thesis, each sent message (e.g. service call) is mapped to an action performed in a given state and each received message (e.g. respond to a service call) is mapped to an event that causes a state transition (i.e. fires a transition). On the right of Figure 2.6, the corresponding state machine for the *LoanManager* is shown. In the state machine, transitions without an attached event result in automatic transitions (NULL). It means that no state action has occurred.

The idea of an iterative synthesis algorithm is that the first scenario produces an initial state machine, which is then extended to accept the other scenarios one by one. In principle, a received message produces a new state transition and a sent message produces a new state in the current state machine, as shown in Figure 2.6. However, the synthesis is based on state merging, i.e. equivalent states are merged into one state based on, for example, equal state actions. Thus, instead of adding a new state, an existing state (if such a state exists) can be used as a target for a state transition. In Figure 2.7, an example of merging two state machines is shown, i.e. states *(q1,q5)*, *(q2,q6)*, and *(q4,q7)* result from the merging of the corresponding states.

2.6.2 Common issues

The scenario synthesis relies on generalizations, which are a result of state merging. The generalization might also produce complex and overlapping loop structures. Identification of strongly connected components (SCC) [13] is a common mechanism for analyzing such structures from a graph. In an SCC, every vertex is reached from any other vertex in the SCC. Thus, connected loops form a SCC. A Tarjan algorithm [64] is often used for detection of SCCs. The algorithm uses a directed graph as input, and incorporates a partition of the graph's vertices into the SCCs. The basic idea is based on a depth-first search. The SCCs form sub-trees of the search tree and the roots of the sub-trees are the roots of strongly connected components.

The resulting state machine can be deterministic or non-deterministic. Some synthesis algorithms always allows generation of deterministic state machines. This is achieved by splitting states and thus separating two con-

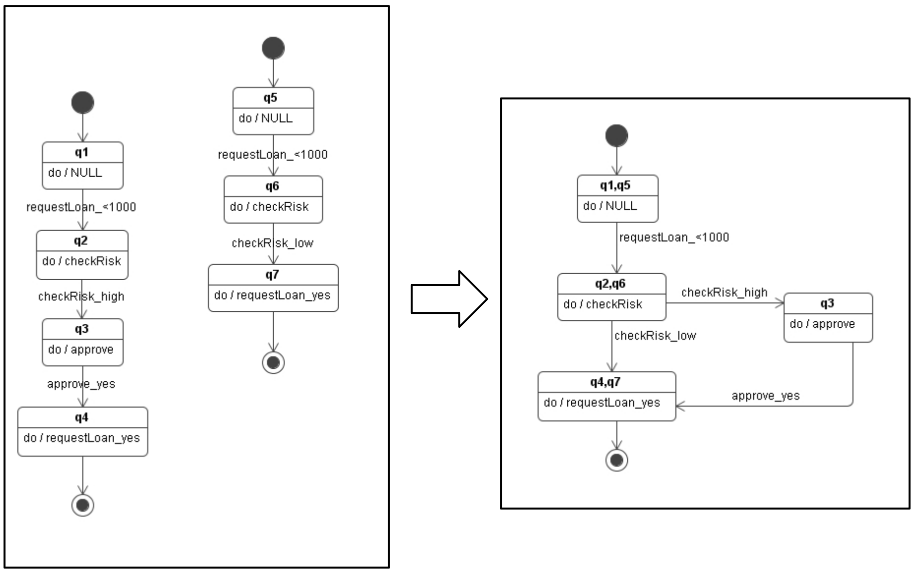


Figure 2.7: State machine synthesis

flicting paths, if necessary. In the worst cases this may result in disconnected state machines. In the context of this thesis, non-deterministic state machines are allowed.

A common problem in automatic state machine synthesizers is "over-generalization", i.e. the synthesis algorithms generalize information given in scenarios so that the resulting state machine accepts additional behaviors other than those represented in the scenarios. The over-generalization issue is not tackled in the context of this study. However, there are synthesizers which can avoid over-generalization, e.g., through interactive synthesis as implemented in MAS (Minimally Adequate Synthesizer) [39, 40].

2.7 Tools

In this section, a brief introduction to the existing tools used in Part III is given. Tools have been developed in the same research group as the study presented in this thesis. In addition, the main rational for the tool selection are given.

2.7.1 Inari

Inari is a pattern-oriented development environment built on top of Eclipse IDE [16]. Inari has been initially developed for specializing Java frameworks.

In principle, it is meant for capturing architectural rules. Inari manages different rule configurations and development concerns described as patterns [22].

An Inari pattern is a structural configuration of elements. To allow specification of patterns independently from any concrete systems, patterns consist of roles rather than concrete elements. When a pattern is applied, one instance of the pattern is bound to concrete model elements or code fragments. Each role has an attached role type, which determines the kind of elements that can be bound to the role instances. Supported role types include Java and UML roles. On the left of Figure 2.8, an example of an Inari pattern is shown. The pattern consists of three UML roles. Dashed lines indicate bindings to concrete UML model elements shown on the right of Figure 2.8.

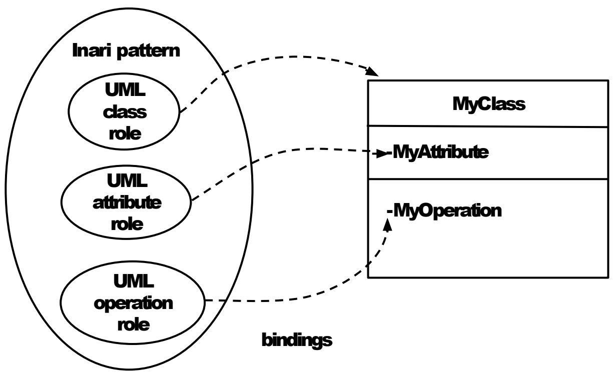


Figure 2.8: An Inari pattern and bindings

The main general requirement for selecting Inari, or other similar tool, is its ability to generate software artifacts based on input descriptions. In this study, Inari is used to generate Java elements from UML sequence diagrams. In addition, the tool should be able to handle different development options, e.g., creation of alternative or optional elements.

2.7.2 MAS

MAS (Minimally Adequate Synthesizer) [39,40] is a command line tool developed for synthesizing application traces. MAS infers a state machine diagram from simple message sequences following Angluin's framework of a minimally adequate teacher [3]. As an output MAS constructs a state machine description. Its input and output formats are textual descriptions.

MAS can be run either in an automatic or in an interactive mode. When run in an automatic mode, MAS forms a minimal state machine with respect to the number of states. In addition, the user can choose whether or not only a deterministic state machine is produced. In some cases, state merging results in over-generalizations. When run in interactive mode, MAS asks the user whether or not certain generalizations should be made. This could be used to avoid unwanted over-generalization. However, in the context of the scenario-driven approach, MAS is only used in automatic mode. Thus, it can be replaced by other similar state machine synthesizer tool.

Part II

SOA and business process development

Chapter 3

Development process for a service provider

In this chapter, a development process for a service provider is presented. Different development phases related to building service-based products are discussed. In addition, the development process provides a context for the rest of the thesis.

The development process is covered in publication [I].

3.1 SPDP overview

Service Product Development Process (SPDP) is a development process for a service provider. It is a result of exploring existing literature (e.g. [4, 5, 8, 50, 65]) and reviewing development processes of an industrial vendor. The process is developed in collaboration with an industrial partner who is a large provider of network and telecommunication based products. SPDP, presented in Figure 3.1, aims at minimizing the development effort for new service-based products developed according to the changing requirements and market. In this thesis, the development phases which are specific to SOA-based systems are presented. In the figure they are shown with solid rounded rectangles. In terms of SPDP phases, the methods and the case studies presented in this thesis address *Product specification*, *Service identification*, *New services specification*, *Service realization*, and *Product realization* phases. On the *New services specification* phase high level composite services are further decomposed into design of smaller service units. When the appropriate service granularity level is reached the process proceeds into *Service realization* phase. If the desired services already exists and they can be identified, these two phases can be skipped. This defines the branching criteria for *New* *services specification* and *Product realization*. In the following sections, the main objectives and activities related to each of the development phases are explained. Description of the required development activities is not exhaustive, but it concentrates on those, which are relevant in the context of the scenario-driven approach.

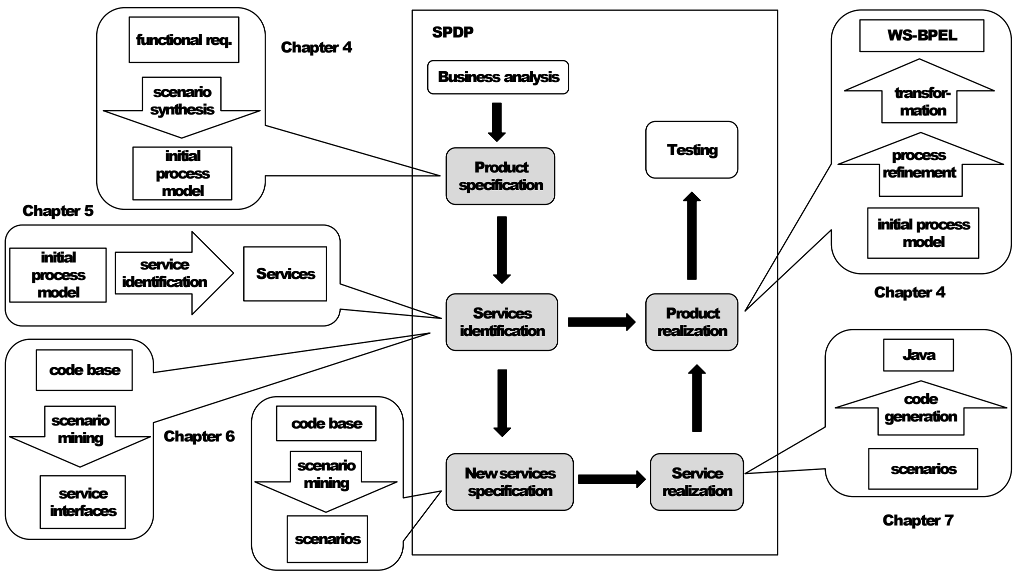


Figure 3.1: The phases of the service-based product development process

3.2 Product specification

In the business analysis phase new revenue opportunities are identified and captured as business level product descriptions. For example, use cases can be used to document the usage of new products and business ideas. In the product specification phase the outcome of the business analysis phase is refined with technical requirements and more detailed descriptions. At this point, both functional and non-functional requirements should be defined.

In Chapter 4, we present a method for using scenarios to describe functional business requirements. The scenarios are synthesized into the form of an initial business process model. Thus, instead of considering of individual use cases one model is created.

3.3 Service identification

In the services identification phase, the need for constituent services is defined and the available existing services are analyzed. Some of the services might already be available (e.g. they can be found from a service catalog), some might be created by modifying or extending existing services, and some might need to be developed from scratch. If development of new services is needed, either from the scratch or based on existing ones, they must be specified and implemented on *New services specification* and *Service realization* phases. In addition to identification of service candidates, initial mapping of business process activities and services is considered at this phase.

In the case study presented in Chapter 5, an initial process model is used as a starting point for service identification. This is an example of a top-down approach. For a bottom-up type of service identification, the method presented in Chapter 6 can be used. It can applied for mining services from existing system, such that the resulting model presents the operations provided by the service interface.

3.4 New services specification

Service identification often leads to design and specification of new services. There are several factors that affect the correct service granularity. On the one hand, unnecessary remote calls can be avoided by careful interface design (e.g. using a transfer object design pattern [1]). On the other hand, operations which take too long should be separated into several individual operations. To make error recovery easier and to avoid performance problems, each operation should be self-contained and stateless. Also, data changes should be performed as part of a single transaction. Finally, business suitability is a major concern to achieve successful service design. At business level, a good goal is to aim at operations which each complete one business task. The smaller the services are, the more, potentially, reusable they are.

In this thesis, scenarios are used for services specification. The development might include, for example, exploiting particular application type specific development rules, such as, those specific to a service family.

3.5 Service realization

Service realization often involves use of APIs, third party services, or existing legacy code. Thus, the system developed must follow certain development rules implied by the existing code base or third party software. These development rules are a type of requirements and they should be documented. However, the documentation is not always available and some sort of reverse-engineering might be needed.

In Chapter 6, a method for mining development rules from application traces as scenarios is presented. The method is based on reverse-engineering existing legacy systems. It can be applied, for example, to construct scenarios for API or service usage. The scenarios so constructed can be used in the service realization phase for generation of application code, as presented in Chapter 7.

3.6 Product realization

In the product realization phase high-level service and business processes are implemented. Initial business process models, especially those produced in the product specification phase, are refined with technical details in order to enable their transformation into executable business process descriptions. Various model transformation techniques and tools have been proposed for automatically transforming process models, given in BPMN or UML, into executable WS-BPEL descriptions [10, 27, 31, 32].

The method presented in Chapter 4, includes refinement of an initial business process model and transformation from a UML model into a WS-BPEL description

3.7 Summary

In this chapter, a development process for service-based systems, called SPDP, has been described. In addition, a summary on how the scenario-driven approach can be applied in a particular development phase has been given. Figure 3.1 presents a roadmap for the rest of the thesis, i.e. connections between the methods developed and the SPDP phases are presented.

Chapter 4

Scenario-based business process development

In this chapter, a scenario-based method for business process development is presented. It includes a systematic model-driven development chain from functional requirements to executable WS-BPEL process descriptions. In addition, the developed prototype implementation, Sketch and BPELGen tools, is presented. As a running example through the chapter a simple loan approval process is used.

These themes are covered in publications [II] and [III].

4.1 Method overview

Business process modeling includes several challenges. On the one hand, a certain amount of technical details needs to be presented at the business process modeling level to enable transformations into executable process descriptions, such as WS-BPEL descriptions. On the other hand, the modeling should emphasize the requirements capture rather than specification of the detailed technical process models. In this section, a method that aims at providing an intuitive starting point for business process modeling is presented. The modeller is expected to be a business-oriented person rather than an IT-developer.

The underlying idea is that the modeller of the business process sketches simple example sequences that show the required behavior of the business process, to be implemented as a Web service orchestration. Services needed for the orchestration can be already existing or they can be implemented in the later development phases. The modeller is not expected to define the more complicated technical details, such as different types of control structures, at this point. Furthermore, this set of simple scenarios does not need to illustrate all the possible execution sequences throughout the process, but it should rather concentrate on the main use cases. For instance, scenarios describing exceptional behavior, such as error handling, may be omitted. Such information can be defined later in the process refinement step, and more importantly, by the IT-developers.

The simple scenarios, described with sent and received messages, are synthesized into a merged process view illustrated as a UML state machine. The synthesis algorithm concludes the control structures, e.g., branching and merging of flows. The intermediate state machine model is further transformed into a process model skeleton, a UML activity model, that supports the business requirements as expressed in the input scenarios.

The process model skeleton created is further refined with WS-BPEL specific elements including stereotypes and structural patterns. Refinement cannot be fully automated and it still might require adding some technical details manually. Finally, a model transformation tool can be used for generating executable WS-BPEL specifications from the refined process model. The modeling method, including the transformation steps, are is described in detail in subsections 4.2 - 4.5. The tool support developed is presented in Section 4.6.

Figure 4.1 summarizes the manual activities (1. and 4.) and the automated model transformation steps (2., 3., and 5.) of the scenario-based process development method. The input and output models are shown in the arrows between the steps.

4.2 Scenario-based modeling

To gather the functional business process requirements, several sources of information can be used. Usually, the requirements can be gathered from the specifications produced in the business analysis phase. In case of migration into SOA, exploring of the existing system or interviewing the system developers might be needed. The requirements are modelled as simple scenarios without control structures. At this point, error handling can be omitted, the focus being on successful scenarios and the desired basic behavior. A basic rule is that each of the main use cases is presented as a simple scenario.

The interaction can be started by the process itself or by an external party. A simple scenario consists of *sent_message* and *received_message* messages. The former means that the process sends a message and the latter means that the process receives a message. A synchronous message can be modelled as a sent and received message pair (*operation* and *operation_response*).

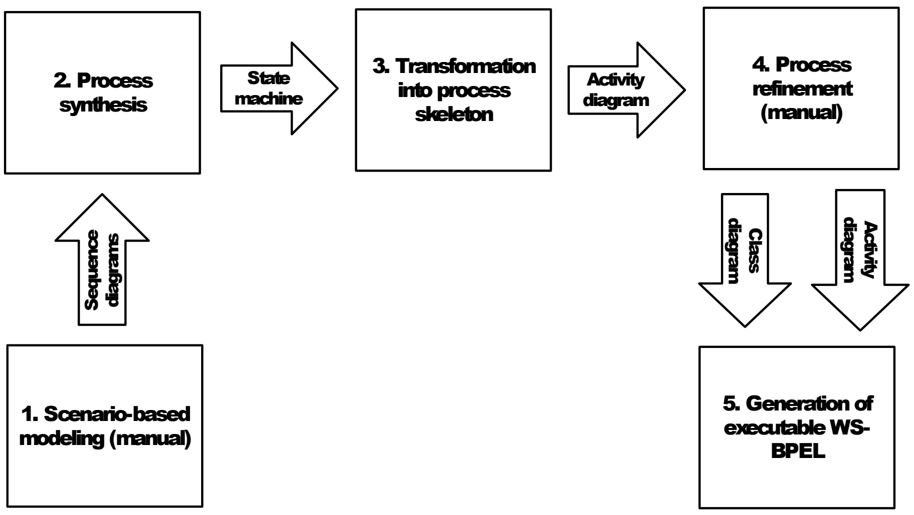

2. Process synthesis
State machine
3. Transformation into process skeleton
Activity diagram
4. Process refinement (manual)
Sequence diagrams
Class diagram
Activity diagram
1. Scenario-based modeling (manual)
5. Generation of executable WS-BPEL

Figure 4.1: Model transformation steps of the scenario-based process development

As a running example, a simple *Loan Approval* example, used also in the WS-BPEL specification [41], is used. It is a simple service to be implemented as a business process, which handles customer loan requests. For decision making, it utilizes external Web services, *Risk Assessment* and *Approver*. The customers can send their loan requests to the *Loan Manager* service, which then returns either a "loan request approved" or a "loan request rejected" message. The approval decision can be reached in two different ways, depending on the amount requested and the risk associated with the customer. Firstly if the customer requests a small loan (less than 1000) and her personal risk is evaluated as low, the request is approved with no further evaluation. Secondly if the customer asks for a big loan (more than 1000) or her personal risk is evaluated as high, the request needs to be handled by external approver. The external approver decides if the loan is to be granted or not. The customer receives a return message according to the decision.

In the *Loan Approval* process, the participants are known in advance and they are modelled with simple interfaces. The process behavior planned is presented as five simple scenarios given in Figure 4.2.

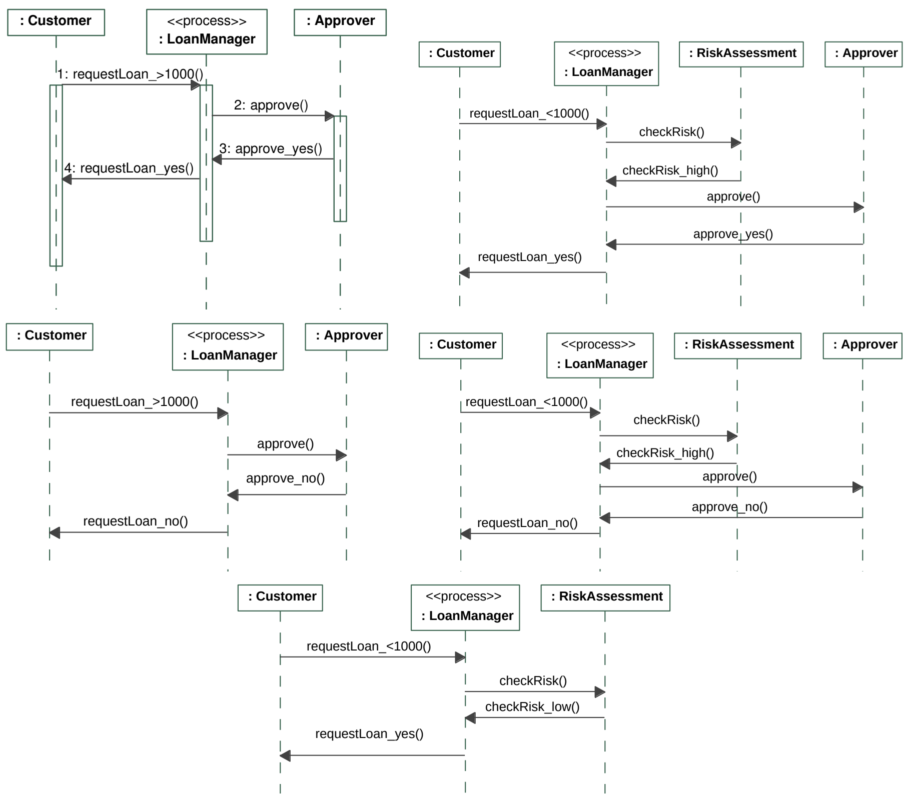


Figure 4.2: Loan Approval scenarios

4.3 Scenario synthesis and transformation into a process skeleton

The functional requirements are synthesized into a merged view presented as a state machine. The synthesis follows the simple idea of merging the states with equal state action (explained in Section 2.6), i.e. the process invokes a specific operation. The resulting state machine for the *Loan Approval* process is shown in Figure 4.3.

In some cases, the input scenarios can be in conflict. Typically, this results in a non-deterministic state machine. However, a conflict might be an accidental error or it might illustrate a deterministic choice, i.e. when the conditional statements cannot be expressed by the simple scenarios. Thus, a non-deterministic state machine is accepted as a result at this point.

WS-BPEL uses two ways of flow modeling: a structured block style and a graph-oriented flow style. WS-BPEL does not, however, allow arbitrary cycles, which can usually be presented in graph-based modeling notation.

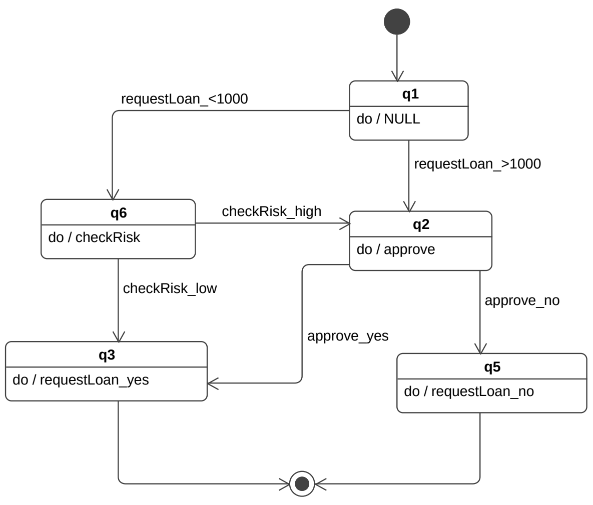


Figure 4.3: A synthesized state machine

With a human designer such unsupported structures can be avoided, for example, by using a limited subset of the modeling language and by providing proper tool support. However, for example, automatic process mining and synthesis approaches sometimes result in unstructured models, which do not allow direct mapping into WS-BPEL processes. In WS-BPEL, loops are presented as structured activities, which have only one incoming and one outgoing control flow. In this work, WS-BPEL generation is restricted to such cases where the previous rules on the control flows hold, i.e. the loop has unambiguous start and final states.

Before transformation into a process model skeleton, the state machine is validated to ensure that it contains no unstructured loops. This can be done by identifying the SCCs (i.e. loops) occurring in a state machine. If a SCC has several incoming or outgoing state transitions, the state machine contains unstructured loops. Thus, it cannot be directly transformed into a WS-BPEL description and a WS-BPEL flavored activity model cannot be created. In such cases, the removal of unstructured loops remains a manual task.

To transform a state machine into a WS-BPEL flavored activity model, WS-BPEL specific transformation rules must be applied. These rules include the use of WS-BPEL specific stereotypes and following “well-formedness” rules defined as so called workflow patterns [68]. The workflow patterns used, including *fork*, *join*, *decision*, and *loop*, support mapping into WS-BPEL descriptions. Table 4.1 summarizes mapping from a state machine into a WS-BPEL flavored activity model. WS-BPEL specific modeling rules are given in detail in the form of a UML profile for WS-BPEL in Section 4.5. As an example, the resulting WS-BPEL specific process skeleton, created from the state machine, for the *Loan Approval* process is shown in Figure 4.4.

State machine	**Activity diagram**	**Stereotype**
a state transition	a call behavior action (and a control flow)	receive
a state action	a call behavior action (and a control flow)	invoke
a loop	a structured activity node with nested substructure	while
a start state	an intial node	-
a final state	a flow final node	-
several deterministic outgoing transitions	a fork node	-
several non-deterministic outgoing transitions	a decision node	-
several incoming transitions	a join node	-
a loop	a structured activity node with nested substructure	while

Table 4.1: Mapping from a state machine to a WS-BPEL flavored activity diagram

4.4 Process model refinement

However, the input scenarios usually do not contain all the information needed for creation of a complete process model. Thus, the process refinement still involves some manual tasks. For example, in and out parameter values are not usually presented in the scenarios. As shown in Figure 4.4, they can be defined using input and output pins to *receive* and *invoke* actions. Also, copying process variables is not defined in the scenarios and these activities must be added manually as *Assign-copy* activities.

In addition, a complete process model includes some static definitions, which are not a part of the synthesized process skeleton. These issues are discussed in the next section.

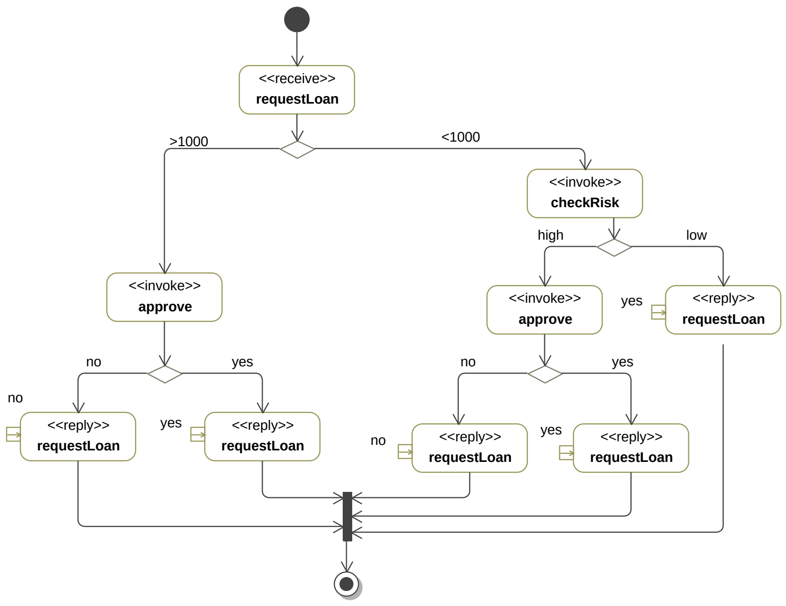
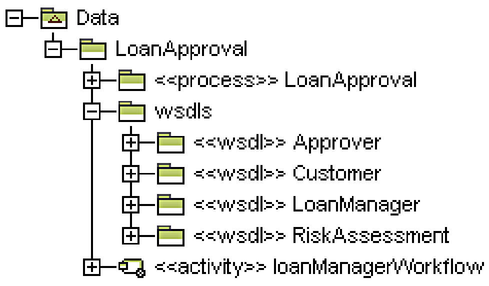


Figure 4.4: A WS-BPEL flavored activity model

4.5 Generation of WS-BPEL descriptions

In this section, generation of WS-BPEL descriptions from UML activity and class diagram based process models is discussed. A complete process model consists of three parts: a static process definition (a class diagram), WSDL definitions (a class diagram(s)), and a workflow model (activity diagram). An example of a process model structure is shown in Figure 4.5. A UML profile for WS-BPEL process models is presented in the next sections. The profile rules are built in a prototype tool called BPELGen, which translates the process model into XML-based WS-BPEL and WSDL descriptions.



```
Data
  LoanApproval
    <<process>> LoanApproval
    wsdls
      <<wsdl>> Approver
      <<wsdl>> Customer
      <<wsdl>> LoanManager
      <<wsdl>> RiskAssessment
    <<activity>> loanManagerWorkflow
```

Figure 4.5: Loan Approval process model

4.5.1 Process profile

Static parts of a process definition are placed in a UML package, stereotyped as *process*. It includes a *process* class, a *correlation* class, *partner link* classes, and *partner* classes. *process* is the main class, which defines the process name, namespaces and the variables used. The *correlationSet* class defines properties for identification of process instances. The *partner* and *partner link* classes are used to define the process participants, i.e. the participating services. A metamodel for a process definition is presented in Figure 4.6.

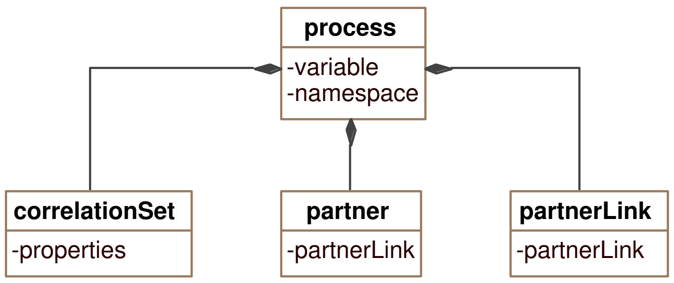


Figure 4.6: Process definition metamodel

4.5.2 WSDL profile

A WSDL package includes definitions of port types (i.e. service interfaces) including their operations and messages. It also defines a service class and namespaces. *A service class* defines the service name, location, binding and port name. A metamodel for a WSDL description is presented in Figure 4.7. As shown in the figure, a WSDL model also includes WS-BPEL specific extensions, which define the service as a participant in the process (i.e. partner link type). A complete UML profile and validation support for WSDL descriptions have been proposed, e.g., in [30]. In the context of this thesis, only the elements necessary to generate WSDL and WS-BPEL descriptions are modeled.

4.5.3 Workflow profile

A process flow is defined as a UML activity model. The supported set of UML activity elements, to be used in a workflow model, is presented in Figure 4.8.

Mapping rules for the workflow elements, from an activity model into WS-BPEL, are presented in Figure 4.8. The first column specifies a stereotype of

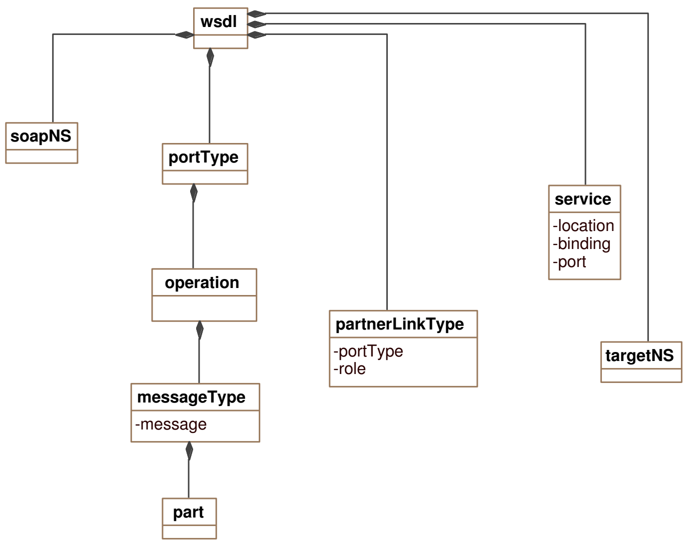


Figure 4.7: WSDL definition metamodel

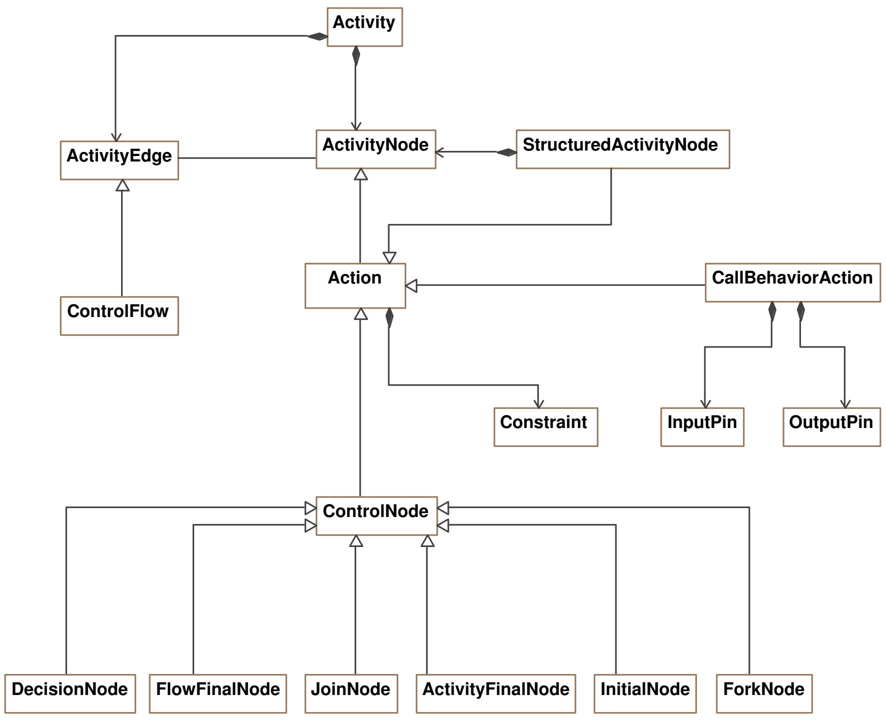


Figure 4.8: Workflow metamodel

a UML element, if required. A short description as well as related constraints are included in Table 4.2.

Table 4.2: UML to WS-BPEL mapping

Stereotype	**UML element**	**Explanations**	**WS-BPEL construct**
invoke	Call behavior action	Input pin for parameters and output pin for return value	Invoke action
while	Structured activity node	Attached UML constraint defines the while condition	While action
pick	Structured activity node	UML note defines value for 'createInstance' attribute. Default value is 'yes'.	Pick action
onMessage	Call behavior action	Inside pick structured activity node. Input pin defines input variable.	onMessage action
	Decision node	Outgoing control flow name defines the case condition	Switch case structure
receive	Call behavior action	inputPin defines input variable value. UML note defines value for 'createInstance' attribute. Default value is 'yes'.	Receive action
	Initial node	Start of sequence	Start of sequence
	Activity final node	End of while loop	End of while loop
reply	Call behavior action	Follows synchronous invoke action. Output pin defines output variable.	Reply action
	Join node	Merging of control flows	Ends switch case
assign	Call behavior action	inputPin=from variable, outputPin=to variable	Assign copy
	Flow final node	End of process flow	Ends the process
	Control flow	Only one incoming and outgoing control flow for each action is allowed	Sequence
	Constraint	Used to attach constraints	E.g. while condition
	Input pin	Input variable	inputVariable/ from variable
	Output pin	Output variable	outputVariable/ to variable
	Fork node	Splitting of a control flow	Flow

According to the WS-BPEL specification a *sequence* consists of sequential activities, such that only one outgoing and incoming control flow is allowed to arrive and leave each basic and structured activity. Thus, such activity sequences are supported in the workflow modeling.

4.5.4 Example: Generation of a WS-BPEL description for Loan Approval process

The Loan Approval process includes four participants: the actual process *Loan Manager* and external services *Customer*, *Approver*, and *Risk Assess-*

ment. The external services are presented as partners in the *Loan Approval* process. In WS-BPEL processes, partners must be defined in separate files using WSDL. It is assumed that the services already exist. Thus BPELGen can be used to import existing WSDL descriptions into a UML class diagram.

A complete structure of the process model is shown in Figure 4.5. The *LoanApprover* package includes the static process definition stereotyped as *process*. The generated WSDL models for the external services are placed in *wsdl* packages. The static process definition is presented in Figure 4.9. Parts of the static information, such as a process name and namespaces, must be defined manually in the *process* class. Process variables are usually used for copying of request and response messages defined in WSDLs. Thus, variables corresponding to the message types are created automatically in the *process* package. Partner link definitions are also created as extensions in the partners' WSDLs.

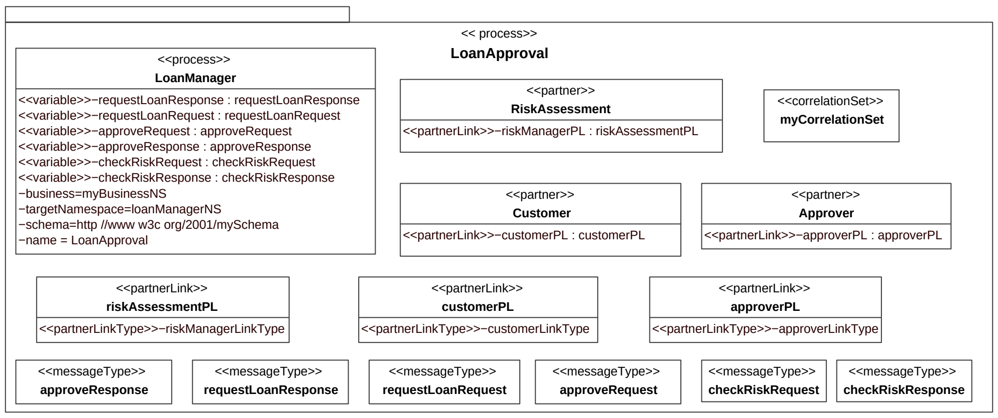

<< process>>
LoanApproval
<<process>>
LoanManager
<<variable>>−requestLoanResponse : requestLoanResponse
<<variable>>−requestLoanRequest : requestLoanRequest
<<variable>>−approveRequest : approveRequest
<<variable>>−approveResponse : approveResponse
<<variable>>−checkRiskRequest : checkRiskRequest
<<variable>>−checkRiskResponse : checkRiskResponse
−business=myBusinessNS
−targetNamespace=loanManagerNS
−schema=http //www w3c org/2001/mySchema
−name = LoanApproval
<<partner>>
RiskAssessment
<<partnerLink>>−riskManagerPL : riskAssessmentPL
<<correlationSet>>
myCorrelationSet
<<partner>>
Customer
<<partnerLink>>−customerPL : customerPL
<<partner>>
Approver
<<partnerLink>>−approverPL : approverPL
<<partnerLink>>
riskAssessmentPL
<<partnerLinkType>>−riskManagerLinkType
<<partnerLink>>
customerPL
<<partnerLinkType>>−customerLinkType
<<partnerLink>>
approverPL
<<partnerLinkType>>−approverLinkType
<<messageType>>
approveResponse
<<messageType>>
requestLoanResponse
<<messageType>>
requestLoanRequest
<<messageType>>
approveRequest
<<messageType>>
checkRiskRequest
<<messageType>>
checkRiskResponse

Figure 4.9: Process definitions for Loan Approval process

The listing in Figure 4.10 presents the WS-BPEL description generated for the *Loan Approval* process. Detailed mapping rules are given in Section 4.5.3. The WS-BPEL description begins with static definitions, such as partner and variable descriptions, and continues with the workflow definition.

Loan Approval is a simple example, which illustrates the whole chain of scenario-driven business process development. The functional process requirements are presented as five input scenarios (Figure 4.2). The simple scenarios are synthesized into a process model containing WS-BPEL specific activities, stereotypes, and control structures (Figure 4.4). After the process refinement step, executable WS-BPEL code is generated (Figure 4.10).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
...
business="myBusinessNS" name="LoanApproval" targetNamespace="loanApprovalNS">
    +<partnerLinks></partnerLinks>
    <partners>
        <partner name="Customer" partnerLink="customerPL"/>
        <partner name="Approver" partnerLink="approverPL"/>
        <partner name="RiskManager" partnerLink="riskManagerPL"/>
    </partners>
    <variables>
        <variable messageType="approveRequest" name="approveRequest"/>
        <variable messageType="approveResponse" name="approveResponse"/>
  ...
</variables>
    +<correlationSets></correlationSets>
    <sequence>
        <receive createInstance="yes" operation="requestLoan" partnerLink="customerPL"
portType="LoanManager" variable="loanRequest"/>
        <switch>
           <case condition="loanRequest:amount<10000">
               <sequence>
                <invoke inputVariable="checkRiskRequest" operation="checkRisk"
outputVariable="checkRiskResponse"
partnerLink="riskAssessmentPL" portType="RiskAssessment"/>
                    <switch>
                        <case condition="risk=low">
                            <sequence>
                                <assign>
                                    <copy>
                                        <from variable="yes"/>
                                        <to variable="decision"/>
                                    </copy>
                                </assign>
                            </sequence>
                        </case>
                        <case condition="risk=high">
                            <sequence>
                                <invoke inputVariable="approveRequest" operation="approve"
outputVariable="approveResponse" partnerLink="approverPL"
portType="Approver"/>
                            </sequence>
                            ...
                        </case>
                    </switch>
                    <reply name="decision" operation="requestLoan" partnerLink="customerPL"
portType="LoanManager" variable="decision"/>
                </sequence>
            </case>
            +<case condition="loanRequest:amount>10000"></case>
        <reply name="decision" operation="requestLoan" partnerLink="customerPL"
portType="LoanManager" variable="decision"/>
    </sequence>
</process>
```

Figure 4.10: Generated WS-BPEL code

4.6 Implementation

To support the scenario-based business process development, two prototype tools have been developed. Sketch is used for scenario synthesis and creation of a process model skeleton, whereas BPELGen is used to generate WSDL and WS-BPEL descriptions from UML models. Both the tools are implemented as Eclipse [16] plug-ins and they use Eclipse UML2 model API [16]. Sketch is presented in publication [II] and BPELGen is discussed in publication [III].

4.6.1 Sketch

Input scenarios for Sketch are given as a simple UML sequence diagram without control structures. The output models are a UML state machine and activity model.

To create a process model skeleton, Sketch supports the following activities:

1. Import simple scenarios,

2. Create a state machine (intermediate),

3. Create a WS-BPEL flavored activity model.

The first activity imports the scenarios as a UML sequence diagram. The second activity is used to synthesize the scenarios into a state machine. The third activity produces the process skeleton as an activity model.

Algorithms

The main functionality of Sketch is described by the following algorithms.

A1: Scenario synthesis (Input: A sequence diagram, Output: A state machine)
The synthesis algorithm implemented in Sketch is iterative; a first scenario produces an initial state machine, which is extended to accept the other scenarios one by one. The algorithm tries to map each message sequence to the current state machine always starting from the initial state of the current state machine. If the state machine already contains a transition corresponding to a received message, no extension to the current state machine is made and the execution continues from the next received message. If the received message is not defined by the current state machine, a new state transition is created in the state machine. The next sent message determines whether the new transition ends up in an existing state machine state (if such a corresponding state exists) or in a new state. In addition, each scenario produces a final transition to the final state. As a result, a deterministic or non-deterministic state machine is created.

A2: Loop handling (Input: A state machine, Output: A state machine)

Loop handling starts with a subtask called *validate*. It first uses the Tarjan algorithm to identify all the loops occurring in the state machine. For each loop, it counts incoming and outgoing state transitions. If more that one of each is found, the state machine contains unstructured loops and the algorithm stops. Otherwise, the algorithm detaches the loops from the state machine. It utilizes *detachOutgoingTransitions* and *detachIncomingTransitions* algorithms to detach incoming and outgoing transitions from the loop. In addition, it eliminates the loop structure by removing the transition from the loop end state to the loop start state. As a result, the loop structure is replaced by a super state with one incoming and one outgoing transition, an initial and a final state, and the actual repeated sequence.

A3: Activity model creation (Input: A state machine, Output: An activity diagram)

The transformation begins from an initial state of the state machine with an empty activity model. For each state and transition in the state machine, a corresponding activity and a control flow is created. Automatic transitions and empty state actions are ignored in the transformation, because they do not correspond to any particular action.

The transformation is divided into two subtasks, namely, *handleState* and *handleStateTransition*. *handleState* is used to recursively create an activity model from a state machine representation. Activities are created as a child either for the activity model or for a structured activity (i.e. loop). Each single state creates an invoke activity whereas a super-state creates a structured activity. A *handleStateTransition* algorithm is used to create a receive activity for each state transition. The transition target is again handled by the *handleState*. While creating the activities, the corresponding stereotypes are also created.

A4: Pattern detection (Input: An activity diagram, Output: An activity diagram)

Pattern detection includes detection for *decision*, *join*, *invoke-reply*, and *loop* structural workflow patterns. Algorithms are recursive such that they are also ran for internal states of each structured activity (i.e. loop).

detectJoin finds parallel deterministic incoming flows and creates a join node to combine them. *detectDecision* finds parallel non-deterministic outgoing transitions. To remove the non-determinism, it creates an additional decision node with the corresponding branches. *detectFork* algorithm is used to handle similar branching structures except for deterministic control flows.

invoke-reply pattern is implemented by simply traveling the activity model to find a corresponding *invoke-receive* pair (presenting *operation* and *operation_response*). Thus, the *invoke-reply* pair presents a synchronous message with invocation and reply messages.

Sketch does not reuse the synthesis algorithm developed for MAS. The main reason is that MAS uses a language recognition approach and handles the input and output as strings. Whereas, Sketch is based on Eclipse UML2 models and model transformations, so a graph-based approach was chosen. Details of the algorithms are presented in the publication [II].

Limitations

Sketch aims at a minimal state machine with respect to the number of states. This means in practice that state merges are made whenever possible. That is, states with equal state actions are merged. Sometimes this leads to over-generalization. This means that the resulting process model accepts invocations, which are not specified in the original scenarios. In the context of process modeling, we do not see over-generalization as a major concern. In addition, the synthesized model is not considered to be a final model, but it is often used as an initial one for further refinement or analysis.

The algorithms implemented in the Sketch tool do not support removal of unstructured loops. Hence, such state machines are not further transformed into process model skeletons.

4.6.2 BPELGen

BPELGen is a prototype tool developed for generating WS-BPEL descriptions from UML models. For collecting the requirements for the required tool support, different practices and activities involved in the business process modeling were analyzed. (a) Modeling an orchestration based on existing WSDLs instead of starting the process modeling from the scratch. To support this setting, existing WSDLs can be imported and translated into a UML model. In addition, information in WSDLs is used to create a process model template. For example, WSDL operations and messages present a selection of available process activities and frequently used variables. (b) If desired, the process and WSDL models can also be constructed fully manually. (c)

In addition, scenario-based modeling and Sketch can be used to produce an initial process model. As a final step, the process model is translated into WSDL and WS-BPEL descriptions. An overview of the model-based business process development is presented in Figure 4.11. The alternative modeling options are presented as three choices (a, b, and c). Transformations implemented in the BPELGen tool are discussed in detail in the publication [III].

BPELGen is a standalone tool, but it can be used together with Sketch as they are both based on UML2 models. BPELGen supports the following activities:

1. Import existing WSDL descriptions (optional),
2. Define the process model,
   (a) Generate a process model template using imported WSDL descriptions, or
   (b) Define the process model manually, or
   (c) Use scenario-based modeling and Sketch,
3. Export WSDL descriptions, and
4. Export WS-BPEL descriptions.

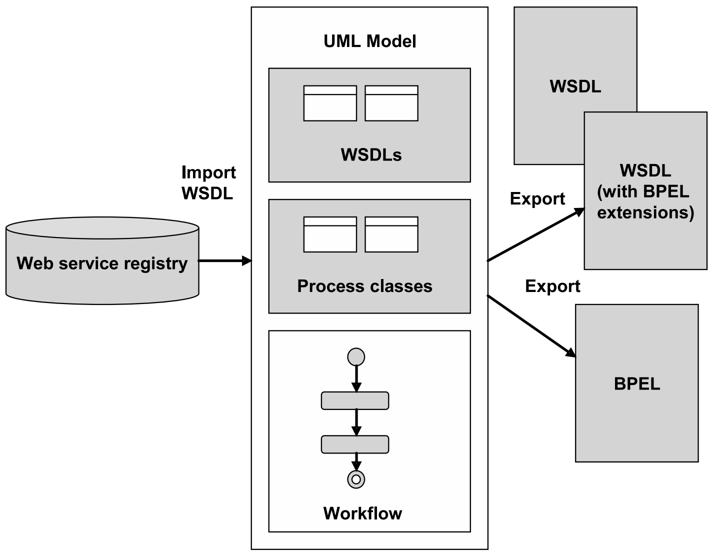


Figure 4.11: Model-based business process development

Transformations into WSDL and WS-BPEL descriptions are based on the WS-BPEL profile presented in Section 4.5. Thus, an input for the model transformation consists of UML class and activity diagrams. As an output, BPELGen produces WSDL and WS-BPEL descriptions as separate XML files.

4.7 Discussion

According to the proposed method, the functional requirements are automatically synthesized into a process model skeleton. However, the process model generated is not expected to be a final one. It usually requires some manual refinement, usually by more IT-oriented people rather than by a business modeller.

In some cases, automatic detection of WS-BPEL specific patterns may not be an optimal, since alternative patterns may need to be applied, for example, on how to handle different types of branching and flow synchronization. This kind of variation could be provided by using interactive or configurable model transformations. However, in our current implementation, only automatic detection of simple structures is supported and the more advanced WS-BPEL structures must be created manually during the process refinement phase if necessary.

The method is targeted at construction of business processes that utilize operation-centric Web services, which perform business activities. Based on the case study results, it has only limited application in construction of process models for resource-oriented and data-centric services, which concentrates on resources and data transfer rather than on operations. In practice, this seems quite natural since WS-BPEL is targeted at operation-centric orchestrations.

Chapter 5

Business process development - A case study

In this chapter, a case study of a scenario-driven business process development is presented. The case study includes two parts. The first part covers development of an initial process model. The initial process model defines the process workflow, but is still lacking mapping to concrete services. The second part continues the business process development using the initial process model as a starting point for services identification. The target is identification of services and reusable units from the existing legacy system.

In the case study, the scenario-based process development method, presented in Chapter 4, is used.

5.1 Case study introduction

The initial target of the case study is to provide the industrial software vendor support for migration into business process based implementation. On the other hand, the research target is to utilize scenario-driven business process development in an industrial context.

5.1.1 Problem statement and context

The context for the case study is an operational legacy system called *Speech Guidance System* developed by an industrial software vendor. The purpose of the system is to provide better guidance and allocation of resources within the sorting process of a large industrial logistics provider. When starting the case study, the system had already been implemented, but it did not utilize Web services or business process based technologies. In addition, the logic for the sorting process was implemented in the user interface component. The industrial vendor's long term goal was to remove system logic from the user interface component through migration into Web services and business process based implementation. They thus wanted support for the migration effort.

Speech Guidance System consists of five interacting systems illustrated in Figure 5.1. The user interface component and configuration systems were new operational systems implemented by the vendor while the other three were existing systems that were integrated with the developed system.

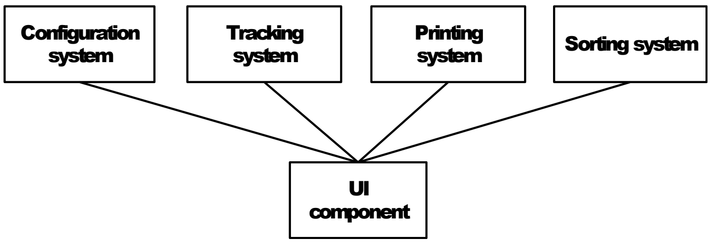


Figure 5.1: Systems involved in *Speech Guidance System*

In the specification, the system requirements were described as use cases and the main scenario for each use case was further illustrated as a sequence diagram. However, the description of the sorting process was spread over several documents and interrelations between the different use cases were unclear.

Speech Guidance System is a pure operation-centric system and it seemed a good candidate for applying the scenario-driven approach. The logistics provider also hosts a market place for their customers. It is rather a resource-oriented system called *Shopping cart*. To gain more information, the scenario-based method was also applied in the *Shopping cart* system as a sub-unit of the main case study. The findings are discussed in analysis of the case study results (Section 5.4.1).

5.1.2 Case study design

The vendor's aim was to improve the maintainability of the system by removing the system logic from the user interface component by adding additional business process layer. Thus, we agreed to the application of the scenario-based method by collecting the scenarios from the requirements specification.

The initial process model is constructed based purely on the business requirements without any knowledge of the underlying services and the legacy system. Next, the focus is on finding proper services to match the initial business process model. Finally, the findings are compared with the existing component model. In addition, the results are evaluated during interviews with the system developers and with project managers.

The purpose of the case study is to answer the following questions related to scenario-driven process development:

RQ2.1	How can the scenario-driven approach be applied in business process development in the industrial context?
RQ2.2	What are the possible limitations?
RQ2.3	How can the scenario-driven approach support migration into business process based implementation? What are the achieved benefits?

Conduction of the case study includes the following steps:

1. Collect functional system requirements from the requirements specification.
2. Model the main use cases as simple scenarios.
3. Synthesize the scenarios and construct an initial process model.
4. Try to identify the service candidates from the legacy system.
5. Compare the findings with the existing component model or implementation, and interview the system developers concerning the results.

The case study execution is presented in the next sections.

5.2 Business process development for Speech Guidance System

In this section, the construction of an initial process model is presented.

5.2.1 Gathering system requirements

Speech Guidance System specification of requirements defines the main use cases for the sorting process. A prerequisite for sorting tasks is that the user has logged into a sorting point. After each sorting task the user can logout from the sorting point or start a new sorting task. The *Speech Guidance*

System instructs the user on the handling of containers and the placing of the sorting items in the right containers.

The main use cases, presented below, were gathered from the specification and modeled as simple scenarios.

Use cases for *Speech Guidance System*

1. **uc006**: Logout from the sorting point
2. **uc007**: Login to the sorting point
3. **uc011**: Move a sorting item to a container
4. **uc014**: Report exception
5. **uc015**: Remove a sorting item from a container
6. **uc018**: Close a container
7. **uc019**: Print a container label
8. **uc020**: Print a container label set
9. **uc021**: Combine two container units

Use cases *uc011*, *uc014*, *uc015*, and *uc018-uc021* are the main sorting tasks, which form the sorting process. Use case *uc007* is a precondition to all the other use cases and a sorting process must end with logging out of the sorting point, i.e. *uc006*. These pre and post conditions must be referenced in each sorting task scenario.

Speech Guidance System has connections to the following external systems: sorting item tracking system (iTR), container label printing system (CLPS), sorting machine, and sorting point data storage. These high level components are present in the use cases. However, they do not necessarily present the actual physical services necessary to be mapped to the business process.

As an example of the use cases, "Print a container label" (uc019) is illustrated as a sequence diagram in Figure 5.2. The user first inputs the container position, type, and identifier. The *Speech Guidance System* invokes CLPS for printing. It receives a status code from CLPS and informs iTR. Finally, *Speech Guidance System* sends the user an "end of printing" message and the user can then continue with new sorting tasks.

As described in Section 4.3, scenarios were synthesized into a state machine. The resulting state machine is presented in Figure 5.3. The state machine is non-deterministic, as it has one non-deterministic branching point. From *setPosition_return* there are several automatic transitions that end up in different states. In addition, the state machine contains one loop.

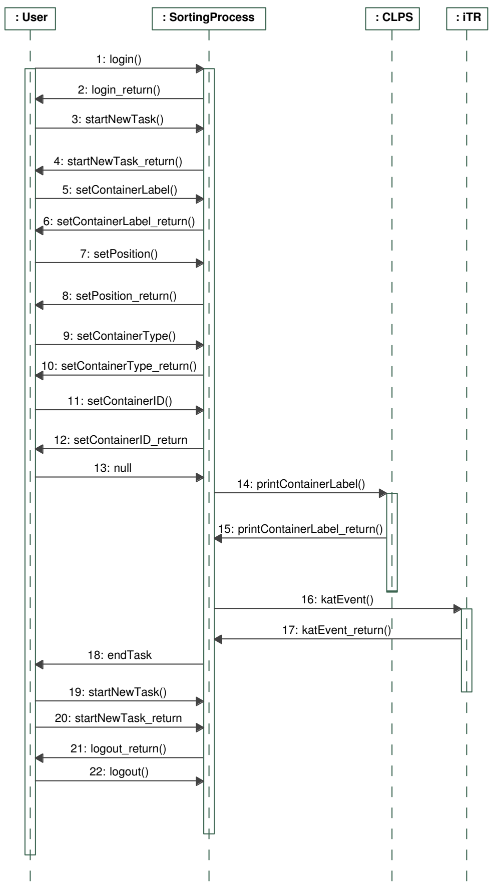


Figure 5.2: Print a container label (uc019) as a scenario

5.2.2 Creation of initial business process model

Before transforming the state machine into an activity diagram, loops must be removed. The state machine contains one loop, which is detached and replaced by a structured activity in the process model. The process model generated from the state machine is presented in Figure 5.4.

The process model has several fork nodes, which represent deterministic branching points. They are not interpreted as parallel executions, but just the splitting of the control flow. The process model contains join nodes, which present a merging point for several flows. Again, the join node does not imply synchronization between the incoming branches. In addition, the process model contains one decision node, which is the result of a non-deterministic branching point (*setPosition_return* in the state machine). Defining the eval-

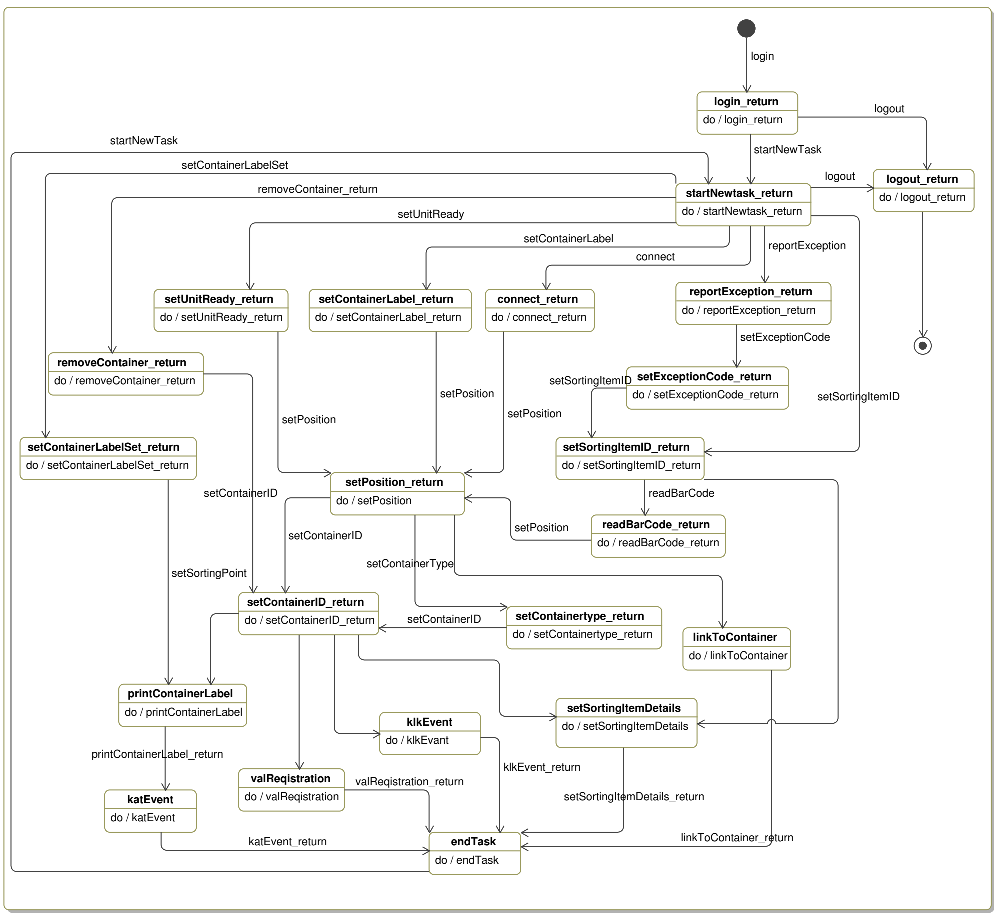
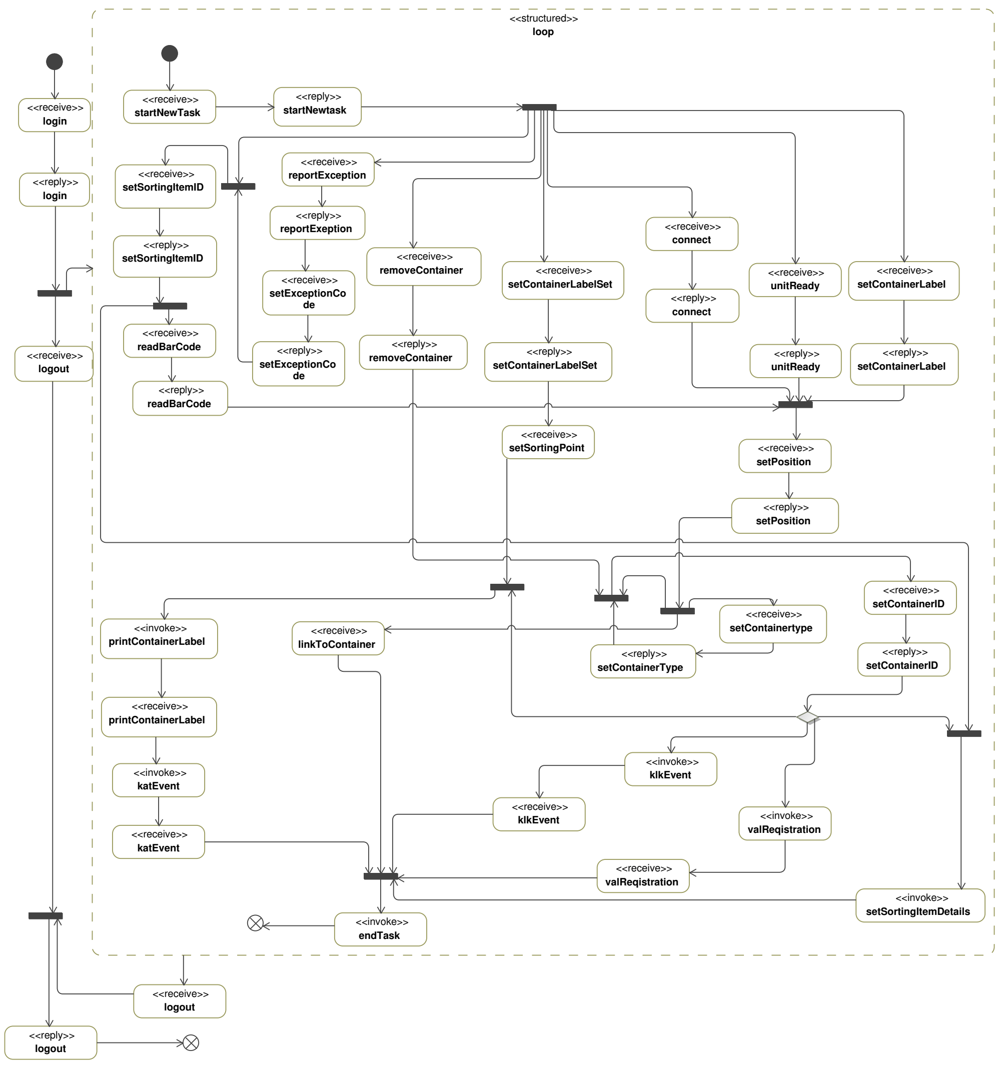


Figure 5.3: *Sorting process* state machine

uation criteria for the choices remains a manual task, as they cannot be derived from the state machine.

The interviews discovered that the vendor's approach have some similarities to the scenario-driven method. The vendor had also started their design process from the use cases and sequence diagrams, which they called dialogs. However, as described in the following, their approach had several drawbacks compared to the scenario-driven approach. It required unnecessary manual work to update parallel diagrams. In addition, the vendor did not model connections between the use cases. Moreover, they did not have tools to enable model-driven development and the diagrams were used as a specification document for the developers. They had problems in discovering the overlaps and dependencies between the different use cases, which led to architectural changes to the design and implementation.



<<structured>>
loop
<<receive>> login
<<reply>> login
<<receive>> logout
<<reply>> logout
<<receive>> startNewTask
<<reply>> startNewtask
<<receive>> setSortingItemID
<<reply>> setSortingItemID
<<receive>> reportException
<<reply>> reportExeption
<<receive>> setExceptionCode
<<reply>> setExceptionCode
<<receive>> removeContainer
<<reply>> removeContainer
<<receive>> setContainerLabelSet
<<reply>> setContainerLabelSet
<<receive>> connect
<<reply>> connect
<<receive>> unitReady
<<reply>> unitReady
<<receive>> setContainerLabel
<<reply>> setContainerLabel
<<receive>> readBarCode
<<reply>> readBarCode
<<receive>> setSortingPoint
<<receive>> setPosition
<<reply>> setPosition
<<receive>> setContainerID
<<reply>> setContainerID
<<receive>> setContainertype
<<reply>> setContainerType
<<invoke>> printContainerLabel
<<receive>> printContainerLabel
<<receive>> linkToContainer
<<invoke>> katEvent
<<receive>> katEvent
<<invoke>> klkEvent
<<receive>> klkEvent
<<invoke>> valReqistration
<<receive>> valReqistration
<<invoke>> setSortingItemDetails
<<invoke>> endTask
<<receive>> logout

Figure 5.4: *Sorting process* process model

5.3 Services identification for Speech Guidance System

The case study was continued to obtain mapping to initial service candidates, which perform the business tasks to implement the constructed business process model. The synthesized state model, created in Section 5.2, was used as a starting point for service identification. It can be considered as an initial process model without binding to a specific process modeling notation, namely, WS-BPEL.

So as to provide the vendor with proposed service mapping, the aim is to identify those *components*, which are self-contained and enable or perform some business functionality.

In the case study, simple heuristics, presented below as Rules 1-3, for identification of service candidates were used. Similar heuristics and approaches have been used already in the literature [6, 20, 34]. The number of states in the state machine was reduced by combining states. Thus, the resulting state machine is an abstraction from the original one. More precisely, Rule 1 defines when two sequential states, A and B, will be combined.

Rule 1: Sequential activities States A and B are combined if A and B are not initial or final states, there is a transition from A to B, B has exactly one incoming transition, and A has exactly one outgoing transition.

As a result, states A and B are replaced by one combined state. In a business process A and B correspond to a sequence of activities. For example, a business rule *The user must login in the system* which can be implemented by three sequential activities: *Set user name*, *Set password*, and *Confirm.* Thus, the activities are combined into one state.

A service decomposition is concluded by analyzing the state machine. In other words, the (most of) potential low level services and their interfaces are identified. Firstly, if a state has several incoming transitions, it refers to potential reusable functionality, thus we propose that the activity considered as a service candidate. Secondly, activities in a combined state might be good candidates to form a service, since the activities are always executed in sequence. Thus, the following states are selected as service candidates:

Rule 2: Reuse If a state (or combined state) is reached multiple times, this implies a potential to reuse, thus it should be considered as a service candidate.

Rule 3: A business rule/requirement In addition, state actions within a combined state potentially provide one business task. Thus, they should be grouped into one service candidate.

Depending on the service granularity, a service candidate might be composed of a (sub)process instead of a single service. In the context of our case study project, only service interfaces are considered. After identification of the service candidates, each candidate should be considered separately.

After applying the presented rules, the state actions *reportException* and *setExceptionCode* are combined into one state since they are done in a sequence without interruption by any other service. They can thus be executed as one business task.

Table 5.1 presents states, which are reached many times. The first column shows the state name and the second column shows the number of actual paths reaching the state. Due to the synthesis process, the number of incoming transitions may not match to the number of actual paths defined in the simple scenarios.

These state actions are thus proposed as a starting point for specification of new services. In addition, the rest of the state actions in the state machine are grouped within these service candidates as suggested.

State action	**Incoming paths**	**Present in scenarios**	**Matching component**
login	1	8	*UserCommand-Manager*
logout	2	8	*UserCommand-Manager*
startNewTask	2	7	*UserCommand-Manager*
endTask	5	7	*UserCommand-Manager*
setPosition	4	4	*ContainerPosition*
printContainerLabel	2	2	*PrintContainerLabel*
setSortingItemID	2	2	*SortingItem*
setContainerID	3	4	*Container*
setSortingItemDetails	2	2	*SortingItemManager*

Table 5.1: Service candidates

The resulting components identified were compared with the *Speech Guidance System* documentation. In the component model, the corresponding components were identified. In Table 5.1, the last column indicates the existing component which provides the corresponding functionality. Apart from *UserCommandManager*, each service action maps a single component. Thus, our recommendation on service candidates seems to fit well with the current implementation. The reason for grouping all the user tasks in the *UserCommandManager* component is that a single service for managing the user interaction is thus provided.

5.4 Case study results and conclusions

The case study results were discussed in two interviews with the industrial software vendor. In this section, analysis on the results is presented.

5.4.1 Analysis

In the first interview, the resulting process model was discussed. According to the interviews with the software vendor, the resulting process model seems to be useful in the following ways: (1) it gives the missing definition for the overall sorting process, including the critical components, (2) it can be used as a model for guiding maintenance and development, and (3) it seems a good starting point for WS-BPEL based process implementation. The vendors' motivation for process-based implementation was to remove the business logic from the user interface component. However, the resulting process model is still lacking in mappings to concrete services.

We did not have a deep understanding, as intended, of the system while applying the scenario-driven approach. Later comparison with the architectural documentation and discussion with the system developers and architects confirmed that the identified components could provide the key services in the business process based implementation. The actual implementation of the business process was not done in the scope of this case study.

The case study included a sub-unit of modeling requirements for the *Shopping cart* system. The shopping cart functionality utilizes several low-level data-oriented services each providing standard operations such as *save*, *add*, and *remove*. After renaming the operations, to distinguish between different services, the scenarios were synthesized into a state machine. The resulting state machine contained several overlapping loops and lots of state merges was done. It seemed obvious that the resulting model was not compatible for direct mapping into a WS-BPEL description.

5.4.2 Conclusions

The aim of the case study was to provide a starting point for migration into business process based implementation. The scenario-driven approach was applied to construct an initial process model for the case study system. According to the interviews, it seemed a good starting point for process-based implementation. In addition, the system developers agreed that it can be used for guiding the system development and improve maintainability. (RQ2.1)

Service identification is an essential factor in successful SOA projects.

From the results of the case study, it can be concluded that the initial process model can be used as a starting point for service identification. In the case study system, the underling legacy system matches quite well with the proposed service candidates. Furthermore, the analysis presented could be useful when estimating the required migration effort, i.e. how much modifications on the existing system is needed. (RQ2.3)

For modeling of low level and/or data-oriented services, the scenario-driven approach might not be an optimal starting point. Furthermore, synthesis is likely to produce unstructured state machines, which do not allow direct mapping into WS-BPEL. In addition, unwanted state merges are likely to produce over-generalization and loss of information. In addition, as already mentioned, rigorous SOA development and migration issues are not focus of the study. (RQ2.2)

Part III

Service development

Chapter 6

Scenario mining for service specification and re-documentation

In this chapter, a method for mining scenarios from application traces. The scenarios constructed can be exploited in development of service and client applications and to support maintenance of existing applications.

Scenario mining is covered in the publication [IV].

6.1 Introduction

A successful effort for mining development rules should be able to differentiate between application-specific and common parts from the execution trace. For example, in the case of service families we should be able to distinguish between varying application-specific parts and common family-specific parts.

Let us consider the *eService* service family that consists of simple Multimedia Messaging Service (MMS) based services. The business goal is to support development of a set of similar service-based products that use the existing MMS platform. The user can invoke an application of the *eService* family by sending an MMS message to the service. The *eService* family includes services, which allow the user to create and send a postcard or related mail item from a picture taken with a mobile phone. It supports two alternative payment methods: charging on the user's phone bill or on a credit card. The method used must be chosen during development time. In addition to MMS, the service family directly uses two external services: a credit card service and a traditional mail service.

Actual reverse-engineering techniques are not in the focus of this study.

It is assumed that traces are gathered using some existing tracing or Web services monitoring tools. In the following sections, a method for scenario mining is presented. The *eService* family is used as a running example of mining and constructing of scenarios for the service family.

6.2 Method overview

The method for scenario mining is based on tracing existing services systems to discover common behavioral patterns, which suggest rules for application development. Identifying the relevant information, and distinguishing it from the typically large amount of uninteresting information included in the traces, is a challenging task. When gathering trace information it is often necessary to capture only one specific type of interaction. To achieve this goal the traces are filtered and merged into a state machine. Then, common, optional, and alternative message sequences are identified based on the state machine. The resulting message fragments are visualized as scenarios using UML sequence diagram notation with control structures. In the following description, scenarios, which present development or design rules rather than system requirements, are called development scenarios. Figure 6.1 presents an overview of the scenario mining method.

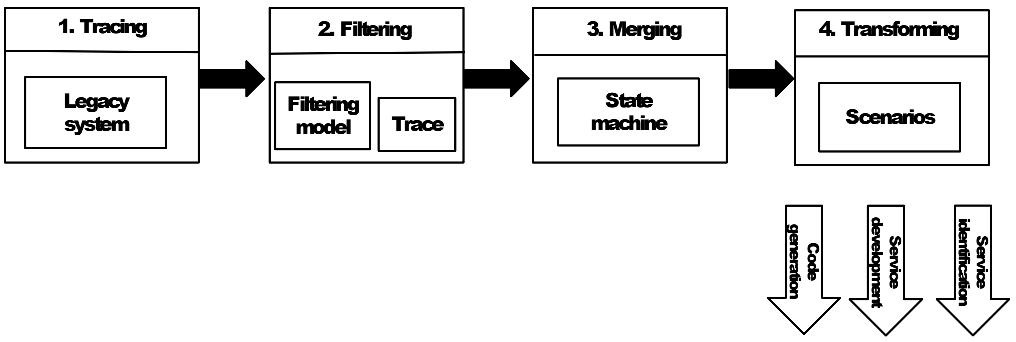


Figure 6.1: Overview of the scenario mining method

Construction of development scenarios for an *application* includes the following steps:

1. **Tracing** of *applications* that use the *system*.
2. **Filtering** the traces and categorizing participants into two groups (i.e. the *application* classes and the *system* classes).
3. **Merging** the filtered traces into a state machine.

4. **Transforming** the state machine into development scenarios.

In the tracing step, traces are gathered from running existing applications. In the filtering step, the traces are filtered so that only direct interaction between *application* and *system* classes is saved. In the merging step, the traces are synthesized into a state machine presenting the merged view of the application behavior. Finally, the state machine is transformed into scenarios that present development rules for the application.

The presented method is not restricted to client-service type of applications, but it can be applied to construct scenarios for any type of application that interacts with the system. As shown in Figure 6.2, internal structure of the system is ignored. The method presented can also be used in service interface mining, i.e. incoming requests in the resulting scenario show the public methods provided by the application.

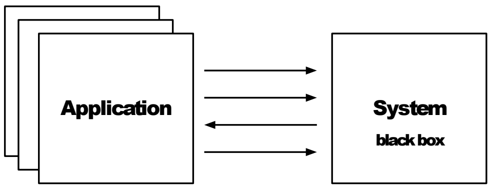


Figure 6.2: Interacting entities

In the following sections, we explain the method in detail and introduce the developed implementation.

6.3 Tracing and filtering the application traces

First, the execution traces are constructed using some tracing or monitoring tool. Instrumented code can also be used. Typically, the resulting traces contain large amounts of information and the interesting interaction occurs only between particular parties, such as a service and its client applications. So as to be able to limit the amount of trace information, those two parts need to be identified and distinguished from the trace. A static UML class model is used to to divide the classes in two categories: application classes and other classes. It is called a *filtering model.* It includes two UML packages. Classes belonging to the subject application are placed in one package and the system classes are placed in the other package. Thus, the resulting filtering model defines a structural model of the analyzed trace. The filtering model is used to automatically remove all unnecessary method calls that do not occur directly between the two parties specified in the filtering model. In addition, all operation calls inside one group, for example subroutine calls inside the application, are omitted.

During the filtering step, a sequence diagram for each trace is generated. In the sequence diagram, all method calls only occur between two participants which correspond to *application* and *system* packages. The actual participants inside the packages, the message senders and receivers, are saved to be used later for creating the actual development scenarios. By using that information the final scenarios are expanded to contain the original participants.

For example, in Figure 6.3, traces gathered after running two different existing *eService* applications are shown.

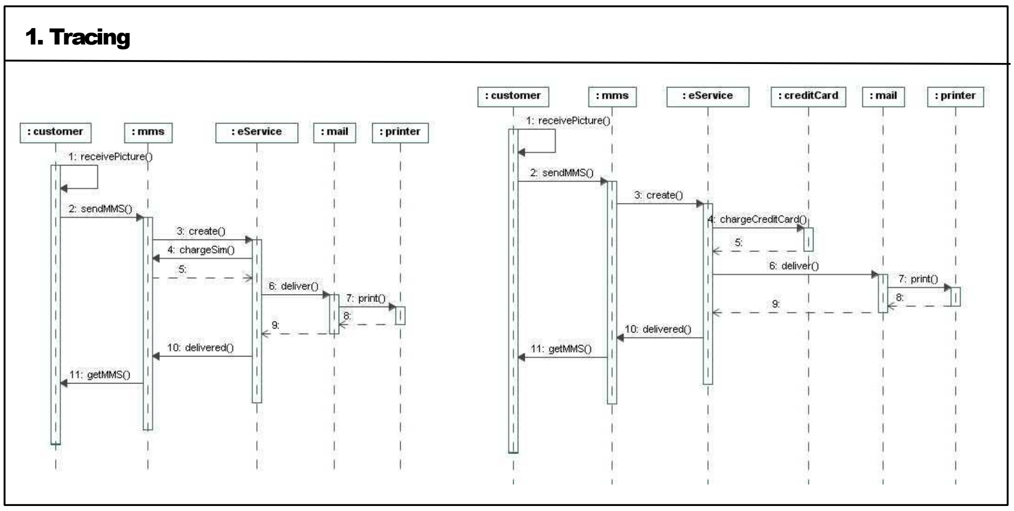


Figure 6.3: eService traces

As shown in Figure 6.3, the *eService* traces include all the system interactions. However, the interesting part of the trace is direct interaction with an *eService* application. To filter out unnecessary information a filtering model is constructed. It consists of two packages: *eService* and *extServices*. The filtering model is presented on the left-hand side of Figure 6.4. In the filtering step, all the classes interacting with *eService* (MMS, Mail, CreditCard, Print) are replaced by one class, *extServices*. As a result of the filtering, messages occurring between *extServices* classes are removed because they do not directly involve the *eService* class. The filtered traces are shown on the right-hand side of Figure 6.4.

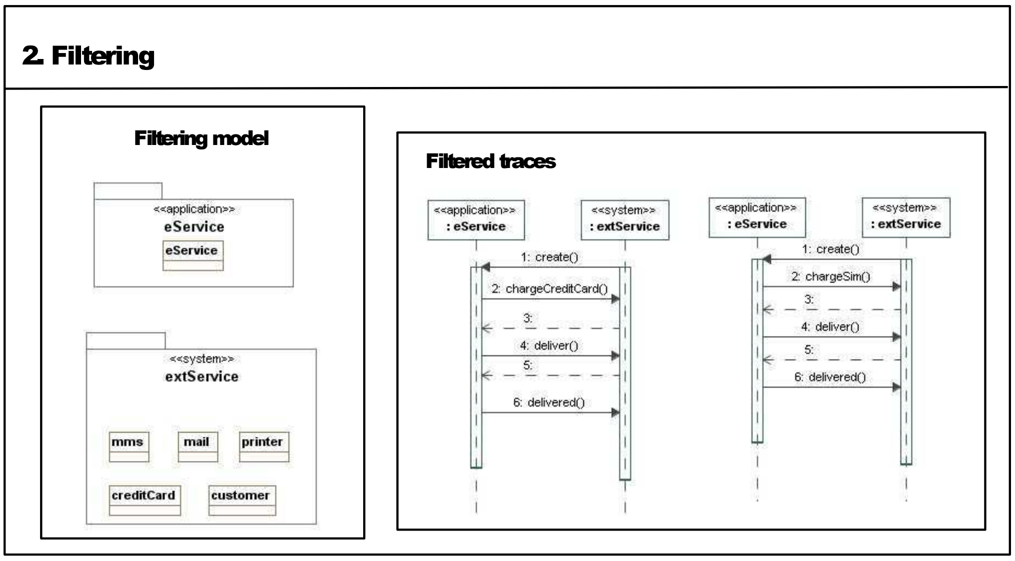


Figure 6.4: Filtered eService traces

6.4 Merging the filtered traces into a state machine

Usually, a single application trace shows only a partial view of the system behavior. For example, the user's choices might affect the program execution. To get more accurate and complete information, the application traces need to be merged. To synthesize the traces, the MAS tool, presented in Section 2.7.2, is used.

The input of a synthesis algorithm consists of participating objects and messages between them. Each send method call is mapped to a state action and each receive method call is mapped to a state transition. The resulting state machine is synthesized based on scenarios representing the system execution traces. It thus represents the overall system behavior.

For example, to identify the common and varying message sequences for the *eService* family, the filtered traces are first synthesized into a state machine. The resulting state machine is shown in Figure 6.5. The variation in the execution traces results in a branching point in the state machine.

The size of the generated state machine depends on the length and similarity of the input traces. Without generalizations, every synchronous method call is translated to its own state having one outgoing transition (i.e. return of the call). During the generalization process, states calling the same method are merged if possible.

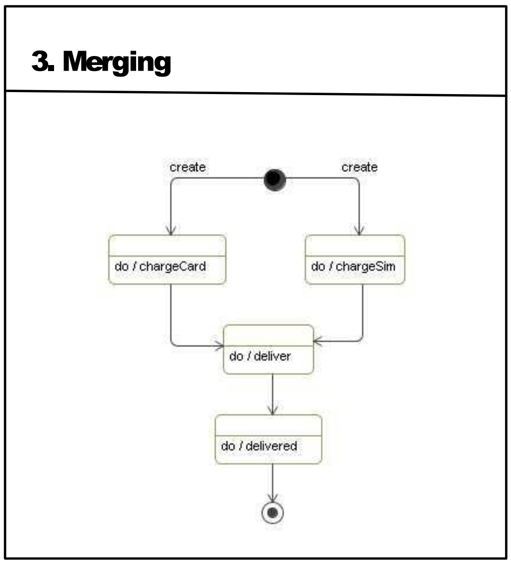


Figure 6.5: Merged eService traces

6.5 Transforming the state machine into scenarios

In the final step, the development scenarios are constructed from the synthesized state machine utilizing participant information given in the original traces. In addition, common behavioral patterns are identified from the state machine. The patterns include optional, alternative, and recurring message sequences. These patterns are translated into UML combined fragments.

The model transformation is implemented using Atlas Transformation Language (ATL) [17]. ATL is used to define mapping between state machine and UML sequence diagram constructs. The transformation is implemented by constructing a sequence of send and receive messages by navigating the state machine diagram and applying predefined transformation rules. The transformation rules are listed in Table 6.1. State machine constructs are given in the left-hand column and corresponding UML sequence diagram concepts are listed on the right-hand column. First of all, the transformation generates a *Collaboration*, an *Interaction*, two *Classes* (application and system), two *Properties* referring to the generated *Classes*, and two *Lifelines* representing the created *Properties*. For each state action, a sent *Message* and a reply *Message* are created to represent a synchronous operation call. A similar transformation is applied to state transitions. The messages generated are grouped inside combined fragments. For instance, multiple outgoing transitions are grouped in alternative fragments.

State Machine	**UML2 Sequence Diagram**
State Machine	Collaboration, Interaction, Lifeline, Class, Property
State action	Message, Operation of Class
State transition	Message, Operation of Class
Multiple outgoing transitions from a State	Alternative CombinedFragment (*alt*)
State which is unnecessary to be passed through	Optional CombinedFragment (*opt*)
Loop structure	Loop CombinedFragment (*loop*)

Table 6.1: Mapping between state machine and UML sequence diagram constructs

The final scenario is constructed based on the state machine by using the original participant information found from the original traces. It is shown in Figure 6.6.

Use of combined fragments enables grouping of sequence diagrams in a compact structure. However, in practice a sequence diagram with more than four or five levels of nested fragments cannot be easily understood. To reduce the number of nested layers in the resulting development scenario, it was decided to disconnect SCCs into separate loops.

An algorithm developed for removing the edges that connect loops into a SCC contains three step. In step *i*, a depth-first search in a state machine is performed to derive a depth-first spanning tree, where the edge x→y between states x and y is marked as a *sp-back* edge if y = x or y is an ancestor of x in the spanning tree [64]. *sp-back* edges result in loops and thus form a SCC. In step *ii*, one of the *sp-back* edges that is connected to the largest number of loops is chosen for removal. If there are several edges connecting to the same amount of loops, the algorithm randomly chooses one of the edges to be removed. Steps *ii* and *iii* are repeated until the state machine diagram no longer contains nested loops. An example of running the algorithm is shown in Section 8.1.3.

6.6 Summary

Application traces are filtered to remove unnecessary operation calls. The resulting traces contain only direct communication between interested par-

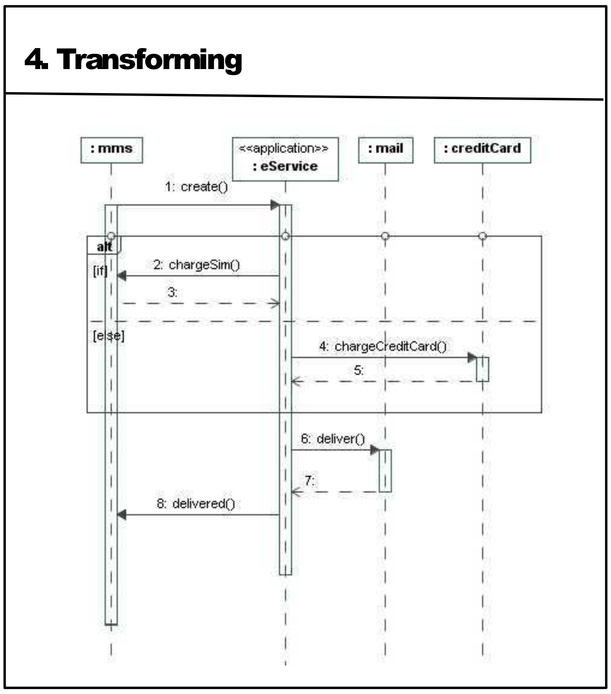


Figure 6.6: Transforming into eService scenario

ties, e.g., the client and the service. To construct the actual rules for their communication, the filtered traces are merged and transformed into scenarios.

The constructed scenarios present development rules for an application, i.e. a sequence of mandatory and optional operation calls. They can be used for guiding the development of future applications, for re-documentation, and for comprehension purposes. In the following section, an approach and tool support for applying the constructed scenarios for generating application code is presented.

6.7 Limitations

In some cases, the scenarios constructed might also accept paths, which are not shown in the original scenarios. This is the result of over-generalization made by MAS, when run in fully automatic mode.

In case of connected loop structures, simplifying the development scenarios is carried out by removing edges from the state machine. Thus, it means that those invocations do not appear in the resulting scenarios. The information on the removed edges is stored and can be used for later revising of the scenario. Furthermore, SCCs can result in the over-generalization made by MAS. Thus, the removed edges are not part of the original traces and no harm is done.

However, in the context of our application we do not consider these two issues to be a major problems. We recommend analyzing the resulting scenario and revising it if needed. Moreover, if the proposed method is used for interface mining the issues are irrelevant, the service interface can be constructed by collecting the state transitions (received messages) directly from the synthesized state machine.

Chapter 7

Scenario-based service realization

Usually, service realization includes utilization of existing APIs, libraries or services. The use of existing software requires learning and knowledge about the system to be used. For example, to build a Web service client with an existing API, requires knowledge on which API calls are needed and in which order they should be invoked.

As shown in the previous chapter, knowledge on system usage can be captured as development scenarios. In this chapter, a method and tool support to exploit the development scenarios as reusable patterns to be used for code generation is described.

This topic is covered in publication [V].

7.1 Introduction

Service realization requires following development regulations. On the one hand, these rules arise from exploiting a certain API, library, or framework. On the other hand, any integration needed with existing legacy system places some requirements on the new services developed. For example, in a developer's guide the API usage is typically described by giving a couple of example applications and some example code fragments. The examples are often sequential in their nature and they are suitable to be described as scenarios.

In Section 6.1, a development scenario for the *eService* family was constructed. The scenario defines the required service operations and the interaction supported by an *eService* application. In this chapter, the example is continued by using the development scenario constructed for building an eCard service. It allows creation of a postcard from a picture taken with a mobile device. From the two alternative payment methods the first alternative, which charges the customer's SIM card, is chosen. The development scenario constructed is used as the basis for the implementation. The development rules are defined as follows. In order to initialize a new eCard, the service interface must provide a *create* method. For the payment it invokes MMS service *chargeSim* operation. To put the eCard on delivery the service invokes the *deliver* operation in Mail service. Finally, the eCard service notifies the MMS service by invoking the *delivered* operation.

In the following sections, the application code generation based on the development scenarios is discussed in detail. In addition, tool support that can be used to manage variations, such as alternative and optional blocks, in the development scenarios is proposed.

7.2 Method overview

Scenarios can be used to capture system requirements and development rules as described in the previous sections. Next, a method for using scenarios for the generation of application code is presented. To start with, the development scenarios can be constructed manually or created with an automatic scenario mining method such as the one described in Chapter 6. It is assumed that the scenarios are presented as one UML sequence diagram that can contain control structures such as combined fragments and nested interactions. Control structures captures variation points in the scenarios, for example, alternative and optional branches.

For the code generation, the scenarios are translated into generative Inari (described in Section 2.7.1) patterns. The detailed configuration of an Inari pattern is explained in the next section. A particular pattern can be applied multiple times, for example, when developing several client applications for a target API or service. Code generation is based on the creation of the default Java elements associated with the pattern roles. As a result of applying a pattern, a skeleton code for the application is generated. An overview of the method is shown in Figure 7.1.

7.3 Inari pattern generation

Java-specific Inari patterns are defined as a configuration of Java class roles, operation roles, and code fragment roles. Each role is associated with a cardinality, which specifies the number of concrete elements that can be bound to a given role. The default cardinality value is one. Other possible cardi-

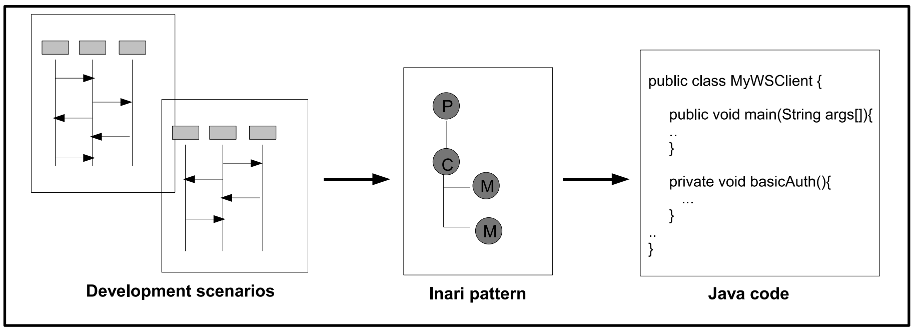

Development scenarios
P
C
M
M
Inari pattern
public class MyWSClient {
public void main(String args[]){
..
}
private void basicAuth(){
...
}
..
}
Java code

Figure 7.1: Overview of the code generation method

nalities are: '?' for optionality, '+' for at least one and '*' for zero or more occurrences. Dependencies, defined between two roles, specify the order in which the elements should be bound. The additional role type *XOR* presents two alternative development options. While instantiating a pattern, pattern roles are transformed into tasks. By performing the proposed tasks the user can automatically create the default elements, namely Java code fragments. The user can also choose to perform all the mandatory tasks automatically. To enable traceability, the pattern also saves the role bindings to the concrete elements.

For example, an Inari pattern created based on the *eService* family development scenario is shown on the right of Figure 7.2. It consists of a Java class role, a Java method role and six code fragment roles. In addition, a branching point is presented as a XOR role in the pattern.

The following listing shows rules for transforming development scenarios into Java roles including associated default Java elements:

1. As a starting point an empty Inari pattern is created.

2. For the *application* object a class role is created. The associated default element is a Java class template and a *main* method.

3. For each sub-interaction, a method role is generated. The default element is a private Java method with the same name as the interaction name.

4. *A send message fragment* produces a method invocation in the code template of the parent method role. The operation signature, as well as the required parameters, are found from the static part of the sequence diagram.

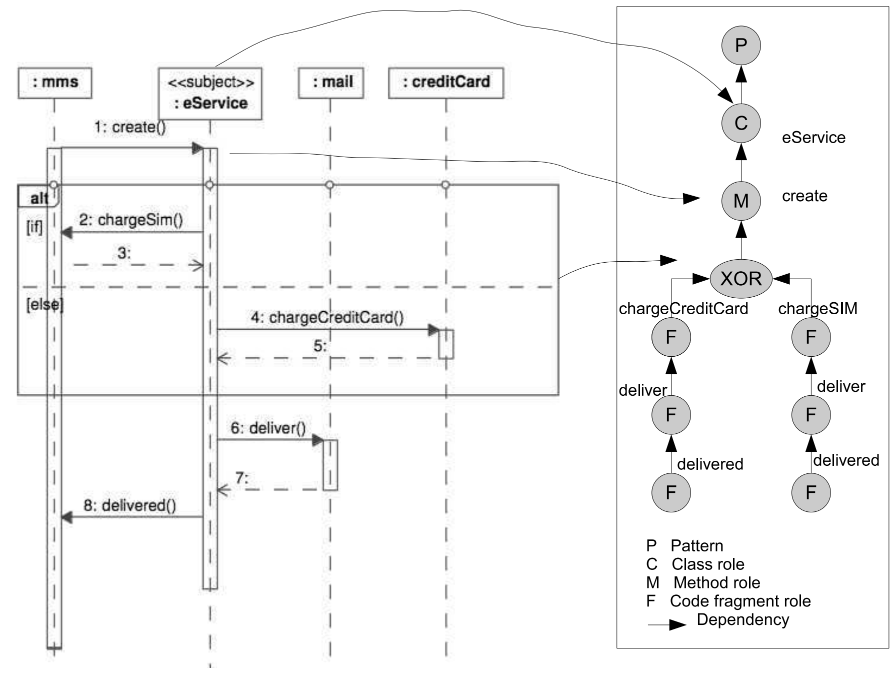


Figure 7.2: Translating a scenario into pattern roles

5. *A receive message fragment* produces a method role in the parent class role. The default element is a public Java method with the same name as the interaction name. The operation signature is created based on the static part of the sequence diagram.

6. *An interaction occurrence fragments* (a reference to another interaction) creates a method call in the parent role's code template.

7. *A combined fragment* presents a control structure, which requires user input and thus it generates a new task in the pattern. For the optional and loop fragments, a new code fragment role is generated. The optionality is translated into cardinality of 0..1 while the cardinality for a loop is 0..*. An alternative fragment presents a branching point and produces three new roles: a *XOR role* (a special role that allows the user to choose between two tasks) and two operand roles for the alternative tasks.

8. For each code fragment role an insertion tag (a place holder) is added in the original code fragment of the parent role.

In addition to the above mentioned rules, a UML element can be associated with an alternative code fragment to be used as a default Java element.

This is done with a specific <*code*> tag defined in the element's property sheet provided by the UML editor.

7.4 Code generation

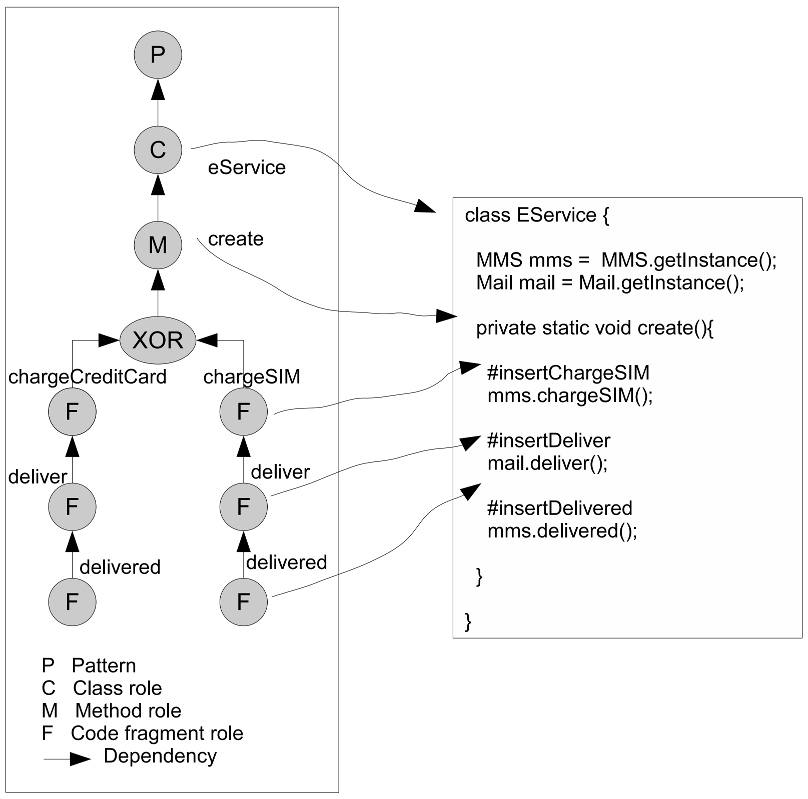
In Section 8.2, a case study to provide code generation for different application clients using a specific programming API is discussed. In the following, the basic code generation principles used in this study are discussed.

Each synchronous message produces a synchronous message invocation (e.g., `stub._setProperty(name, value)`) and a create method creates a new object (e.g., `name = new QName(name, value)`). Each message fragment specifies possible parameter values for a send message and a return value for a receive message. If a non-void return value is specified, a substitution is created accordingly. A static method invocation references the corresponding class (e.g., `Factory.getInstance()`). In addition, each interaction (translated into a Java method) may introduce some local variables and parameters.

For simple data types UML data types (String, Integer, Boolean and Unlimited natural), which are built into the CASE-tool, is used. During the code generation, they are mapped to corresponding Java primitive types. Any complex data types, which are used in the interactions, must be defined in the static part of the UML model to create the corresponding Java classes.

Code defined by the message fragments is placed as code templates in the Java roles. Code templates are used to generate the actual Java code for the application. When applying the generated pattern, a Java class including its methods and code fragments are generated.

For example, code generation for a *eService* application include generation of the public interface and simple application logic. *eService* pattern, shown on the left of Figure 7.3, can be used to create Java code for a service application as described below. Unbound pattern roles appear to the user as a "to-do" task list. The first task is to provide a Java class *EService* and a public *create* method. On the right of Figure 7.3, the corresponding Java code is shown. Next, the Inari user must select between two alternative payment options. If she selects the right branch, she must perform three tasks: 'Provide invocation *chargeSim*', 'Provide invocation *deliver*' and 'Provide invocation *delivered*'. The generated stub code for the *create* method includes insertion tags for code fragment roles appearing as child roles for the *create* method role (e.g. *#insertChargeSim*). By performing the tasks, code fragments for the method invocation are generated after the insertion tags.



```
class EService {

  MMS mms =  MMS.getInstance();
  Mail mail = Mail.getInstance();

  private static void create(){

    #insertChargeSIM
    mms.chargeSIM();

    #insertDeliver
    mail.deliver();

    #insertDelivered
    mms.delivered();

  }

}
```

Figure 7.3: Applying *eService* pattern for code generation

7.5 Discussion

The code generation rules might be slightly different depending on the application context. The following three application types, at least, can be distinguished: (i) application clients (only application logic, no call-backs), (ii) interfaces (public methods), and (iii) services (public methods and application logic). The first two types, an application client and interface, are straightforward and can be considered as a simplification of the third application type.

Inari patterns are used to capture design-time variation of a role configuration. When applying the pattern, the variation points are fixed by the Inari user. The presented code generation rules thus support only simple static interactions between the application and the system. In the method presented, support for dynamic interactions, i.e. creation of control statements, is not considered.

Chapter 8

Scenario-based service development - Case studies

In this chapter, two case studies on constructing and using scenarios for application development is presented. The first case study utilizes the method presented in Chapter 6. In the second case study, the method presented in Chapter 7 is applied.

8.1 Mining design rules for JAX-WS API

In this section a case study on automated mining of scenarios from application traces is presented. The case study utilizes the approach presented in Chapter 6 for mining design rules implied by JAX-WS API.

This case study is presented in detail in publication [IV].

8.1.1 Case study introduction

The purpose of this case study is to apply the scenario-driven approach for re-documentation and specification of development rules.

Problem statement and context

Web service development usually exploits some existing frameworks or software libraries. Use of existing services or APIs requires knowledge of the system to be used.

Java Web Services Developer Pack (JWSDP) [63] is an example of a tool package used to develop Web applications. It includes Java API for XML Web Services (JAX-WS), which is a technology for building Web services and clients that communicate using XML. It supports three kinds of stand-alone

Web service clients: Static Stub, Dynamic Proxy, and Dynamic Invocation Interface (DII) clients.

An assignment in a graduate course "Web service development techniques" at the Tampere University of Technology include the building of three different JAX-WS client applications. Each client uses Google SOAP Search API [21] that provides three operations for submitting queries to Google. The client applications can make a typical Google search, to access cached pages, and to query a Google's spelling suggestion. In this case study, the aim is to extract JAX-WS rules by tracing these applications.

Case study design

In this case study, the method described in Chapter 6 for mining development scenarios is applied. The case study focuses on the following research questions:

RQ3.1	How can development rules be captured as scenarios?
RQ3.2	How can scenarios be constructed automatically from application traces?

Conduction of the case study includes the following steps:

1. Select the applications.
2. Construct the filtering model.
3. Gather and filter the traces.
4. Synthesize the traces and construct the scenarios.
5. Compare the resulting scenarios with the instructions found in a development guide or tutorial. Alternatively, compare the results with existing manually constructed scenarios.

Execution of the case study is presented in the following sections.

8.1.2 Tracing and filtering

To construct the development scenarios nine client applications utilizing Google SOAP Search API Web service is used. The applications include three implementations for each type of standalone clients, namely, Static Stub, Dynamic Proxy and Dynamic Invocation Interface (DII) clients.

In the first step, Eclipse TPTP [18] is used to monitor the client applications and to produce three execution traces for each type of JAX-WS clients. The produced traces include information on participating classes and a sequence of messages sent between them. Each trace concerns one type of API usage. In this case study report, DII client, which is a fully dynamic Web service client and requires no generated stub classes, is used to illustrate the case study execution. A complete report can be found in publication [IV].

To construct a filtering model a case tool is used to reverse engineer the source code classes into UML. The classes are organized into two packages. The *Application* package includes all classes created by the application developer, and the *API* package contains reverse engineered JAX-WS API classes. The traces are filtered using the filtering model as described in Section 6.3. As a result the filtered traces include two participants: *Application* and *API*. As an example, a part of one filtered trace for DII is shown in Figure 8.1.

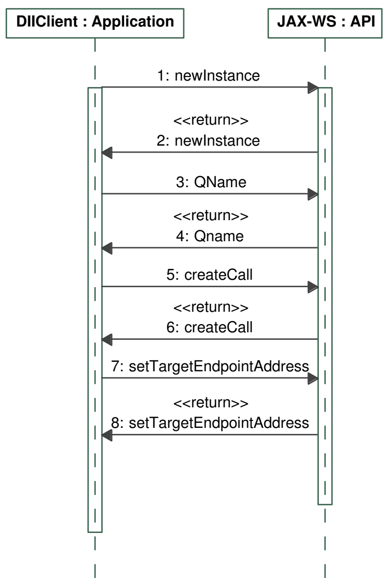


Figure 8.1: A filtered trace

8.1.3 Merging and constructing scenarios

To identify common and application specific parts, the filtered traces are merged into state machines as described in Section 6.4. The aim is to construct intuitive scenarios. Thus, a state machine is constructed for each client type separately instead of merging all the traces.

The scenario construction follows the guidelines presented in Section 6.5. The state machine for DII clients includes SCCs, which need to be removed using the Tarjan algorithm. For example, the edges q8→q2, q3→q2, and q9→q2 in Figure 8.2 are *sp-back* edges that form loop1<q2, q6, q7, q8>, loop2<q2, q3>, and loop3<q2, q10, q11, q9> containing a subloop <q9, q2>. All the loops are connected with each other and therefore they form an SCC. The algorithm removes transition q9→q2 on the first round. The state machine still contains an SCC and the algorithm goes back to step *ii* and chooses the edge q3→q2 to be deleted. After that, the state machine contains no SCCs and the algorithm stops.

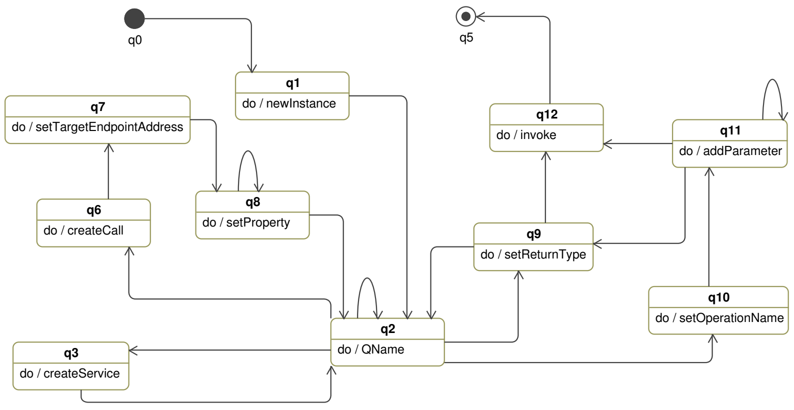


Figure 8.2: State Machine for the DII Clients

A development scenario constructed for DII client is presented in Figure 8.3. The scenario starts with a mandatory part in which a service object is initialized. While transforming the state machine into a scenario, loops are placed in separate sub-scenarios (shown as *ref* fragments in the figure). Inside *loop1*, a call object and a method parameter are both created and the target endpoint address is set. A nested loop fragment is used to set properties defining a SOAP binding style. The invoked operation signature is defined at runtime. It includes defining the operation name, parameters, and the return type. In Figure 8.3, this is done in the second outermost *alt* fragment in two alternative ways.

As an example of over-generalization during the state machine synthesis, the diagram allows the passing the first operand without defining the return type, which is defined inside *opt* fragment. In addition, the second operand passes the operation name definition. After specifying the operation signature, the actual service invocation is done (*invoke* method in the figure). Placing of the last message in the outermost *alt* fragment is due to a deleted edge from the state machine (Figure 8.2 q3→q2). In addition, because of over-generalization, the original state machine allowed the calling of a service before invoking *createService*. To avoid this kind of misbehavior it is possible to run MAS in an interactive mode and to mark the invalid paths. Naturally, this would require some knowledge of the API.

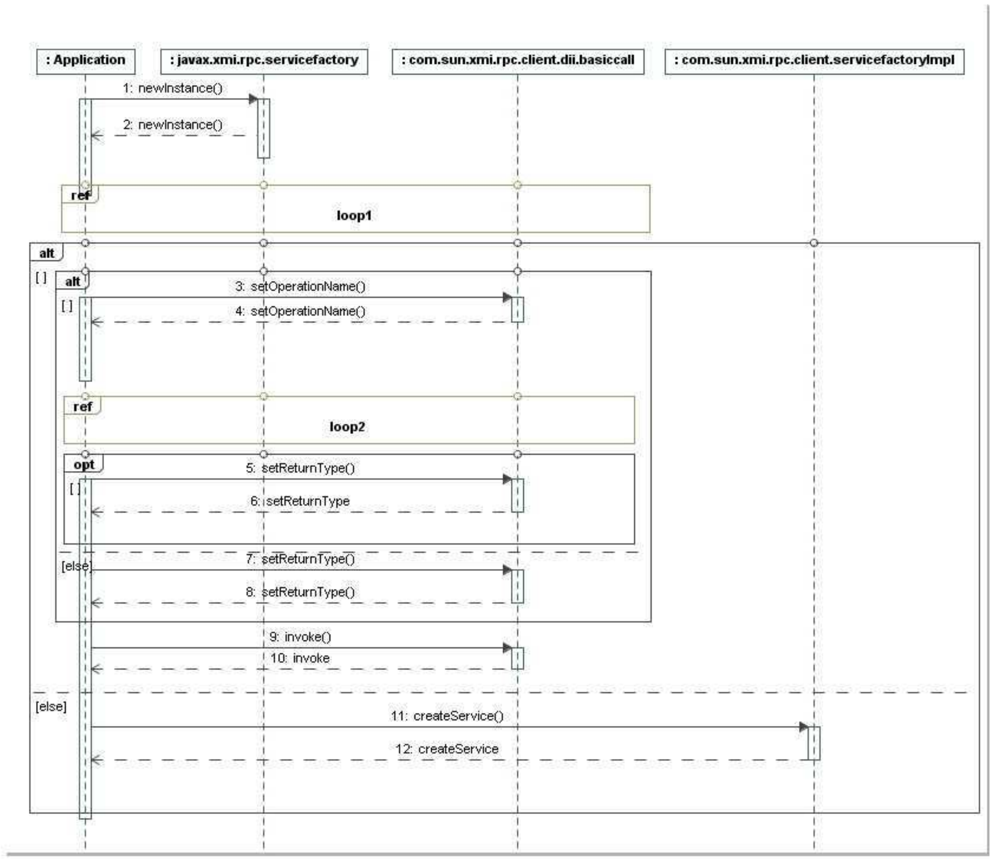


Figure 8.3: A DII client development scenario

8.1.4 Case study results and conclusions

The result of comparing the scenario with existing scenarios constructed manually based on the J2EE tutorial is presented below.

Analysis

As a result three scenarios were constructed, namely, for a Static Stub, Dynamic Proxy, and DII clients. The resulting scenarios were compared with the existing scenarios, which were constructed manually based on the J2EE tutorial. Scenarios for Static and Dynamic clients followed the guidelines given in the tutorial. The DII scenario was not a complete match due to the removal of overlapping loops. Otherwise, it captured the development rules.

In the filtering step, the number of message calls in traces were reduced significantly. Furthermore, when the traces are synthesized into the final development scenario, some of the calls are merged and recurring sequences are collapsed into a loop fragment. Thus, the number of total calls can be even less than in separate traces. The number of message calls while conducting the case study is presented in Table 8.1.

Client type	Traces	Filtered traces	Development scenarios
Static	4751	3	4
	3052	3	
	6183	3	
Dynamic	5926	5	9
	6156	27	
	5932	5	
DII	2287	16	11
	2305	17	
	2295	16	

Table 8.1: Number of method calls in different steps

Conclusions

A challenge when constructing development scenarios is the ability to recognize a type of interaction that determines the development rules we want to capture. In addition, the resulting development scenarios should be able to distinguish between application specific and common mandatory interaction. To understand the overall picture, it is essential to keep the size of the development scenarios as small as possible. In addition, the usage of sub-interactions and control structures raise the abstraction level, which makes the scenarios more structured and easier to comprehend. (RQ3.1)

In the case study, the scenario mining method is used for automatically constructing the development scenarios. By using the filtering model, the number of message calls in the traces is reduced significantly during the filtering step. To identify optional and mandatory parts, the traces are merged. In the final step, the state machine is transformed into a UML sequence diagram with control structures. (RQ3.2)

8.2 Code generation for JAX-WS API based applications

In this section a case study on using scenarios for code generation is presented. The method presented in Chapter 7 is applied for Web service client development with JAX-WS.

This case study is presented in detail in publication [V].

8.2.1 Case study introduction

The purpose of this case study is to apply the scenario-driven approach to service realization.

Problem statement and context

JAX-WS supports three kinds of stand-alone clients: Static Stub, Dynamic Proxy, and Dynamic Invocation Interface (DII) clients. Guidelines for client development with JAX-WS are described in J2EE Tutorial [62]. The three alternative application types are explained as textual descriptions, code examples, and application examples. The developer needs to become familiar with this documentation. For beginners, instead of trying to understand the JAX-WS API, the easiest way to get started is often to copy a sample application and modify that to match their own purposes.

In this case study, JAX-WS API rules are captured as scenarios which can be used to guide the application development. Instead of copy and paste programming, the target is to directly apply the development rules for code generation.

Case study design

As a starting point, a tutorial or developer guide is used to construct the scenarios. If running applications are already available, the scenario mining approach, defined in Section 6, can be used to construct the scenarios. This case study is carried out following the method described in Chapter 7. The case study focuses on the following research questions:

RQ4.1	How can scenarios be used in service realization?
RQ4.2	How to use scenarios to generate application stub code?

Conduction of the case study includes the following steps:

1. Construct a scenario which captures the desired design or development rules.

2. Transform the scenario into a generative pattern.

3. Apply the pattern to generate application code.

4. Compare the resulting application code with the instructions and example code found from a developer guide or tutorial. For evaluation, estimate the amount of generated code versus the need for manual effort.

Execution of the case study is described below.

8.2.2 Constructing development scenarios

The development scenarios for each client type were constructed based on examples given in J2EE tutorial. In this experiment, UML sequence diagrams with control structures were used. One scenario with three alternative sub-scenarios, one for each client type, was thus defined.

A structure for the defined scenario is shown in Figure 8.4. It consists of one main scenario (*ClientApplication*), with three alternative sub-scenarios. Each subscenario specifies one type of a client application: Static Stub, Dynamic Proxy, or DII client.

8.2.3 Pattern-based code generation

The development scenario is transformed into a generative pattern according to the mapping presented in Section 7.3. The pattern is capable of capturing alternative development scenarios. It can thus present development rules for Static Stub, Dynamic Proxy, and DII types of clients. When applying the pattern, the Inari user selects the desired client type. By completing the proposed tasks, Java code (a Java class, methods, and attributes) for the client is generated according to the API rules.

The code fragments associated with the pattern roles follow the code generation principles described in Section 7.4. The following code listing shows the code generated for a Static Stub client with basic authentication and one service invocation. The insertion tags, which are placeholders for the inserted code fragments, are shown between the code lines.

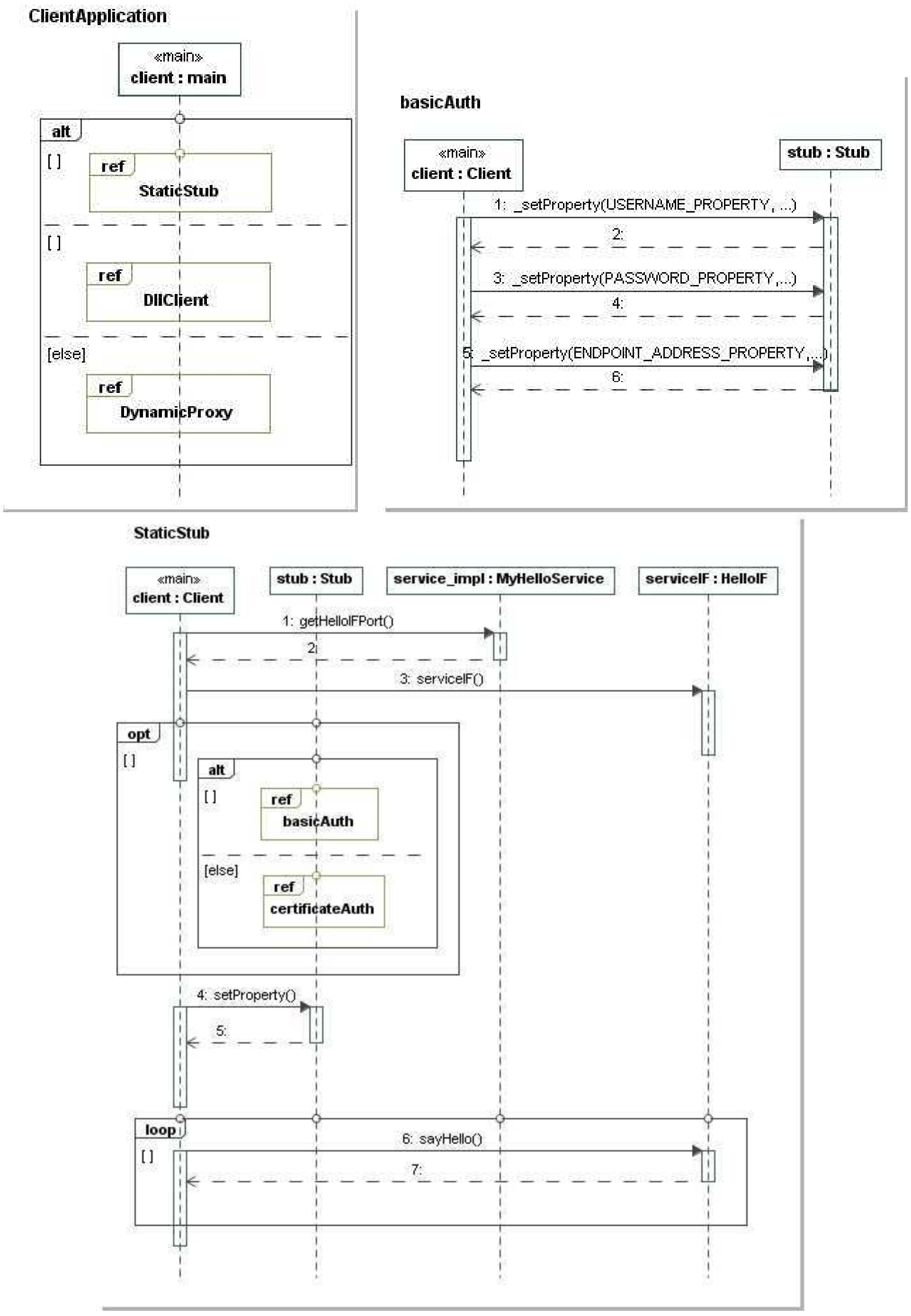


Figure 8.4: A client development scenario

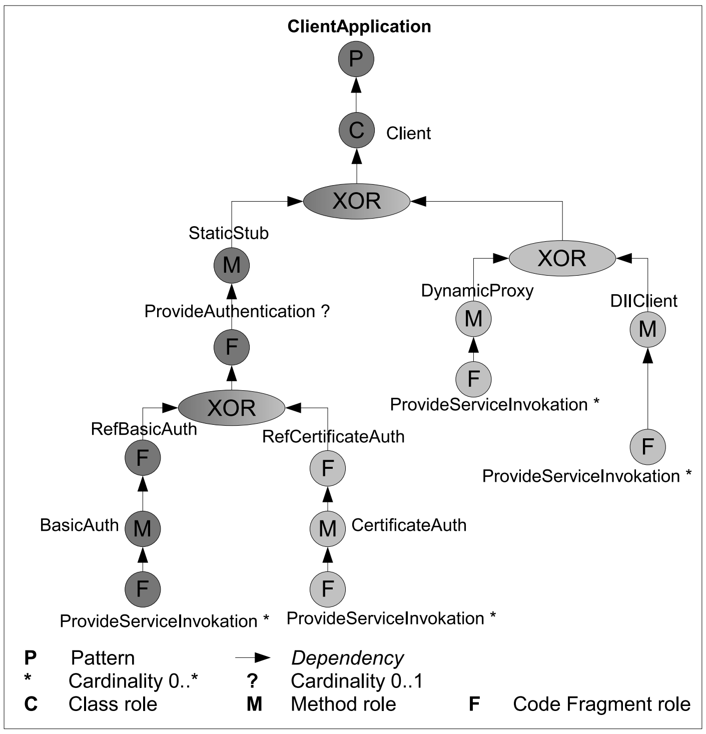


Figure 8.5: A *ClientApplication* pattern

```
II     public class MyStaticClient {
II        //#insertStaticStub
III       Stub stub;
II        public static void main(String[] args) {
III           private String endpointAddress = args[0];
II            try {

IV                stub = (Stub) (new MyHelloService_Impl().getHelloIFPort());

II                //#insertProvideAuthentication
II                //#insertBasicAuth

I,II              basicAuth(args);

II                //#insertCertificateAuth

I                 stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
I                    endpointAddress);
IV                HelloIF serviceIF = (HelloIF)stub;

II                //#insertServiceInvocation
I                 serviceIF.sayHello("Duke");

II                } catch (Exception ex) {
II                     ex.printStackTrace();
II                }
II        }
I,II      public void basicAuth(String[] args) {
 ...
      }
II   }
```

The code lines constructed can be grouped as follows:

Group I) generated directly as specified in the interaction fragment,

Group II) defined or interpreted in the pattern generation procedure,

Group III) variable definitions, specified inside <*code*> tags in documentation field of each interaction diagram, and

Group IV) defined in the documentation field of a message or added manually by the developer.

8.2.4 Case study results and conclusions

Analysis of the resulting application code is presented below.

Analysis

The composed Inari pattern captures the development of three alternative JAX-WS client application types. In this case study, the pattern was applied for creating a Static Stub client.

The resulting code is a “runnable” Java program. A major part of the application code can be generated from the scenario information together with the code generation rules implemented in the pattern creation procedure (i.e. groups I and II in the code listing). As a limitation in the current implementation, required variable declarations need to be defined explicitly in the scenario’s property sheet or added manually in the generated Java class. The same applies to object casting.

Conclusions

In the case study, scenarios were used for Java code generation. Scenarios were transformed into generative patterns associated with Java code fragments. By applying a pattern, the associated code was generated. (RQ4.1)

Generative patterns can be used to generate stub code from the scenarios. The code generation can be used for generation of a simple application based on the interactions specified in a scenario. The code generation, especially, supports creation of public methods and method invocations. (RQ4.2)

Part IV

Related work and conclusions

Chapter 9

Related research

9.1 Background of the scenario-driven development

9.1.1 Scenario synthesis

State machine synthesis from sample scenarios has been widely studied, e.g. in [9, 24, 67, 76, 78]. An automatic method to generate UML statecharts from a collection of UML sequence diagrams has been applied by Whittel *et al.* [76]. Since the set of sequence diagrams is usually incomplete and does not contain enough information for synthesizing appropriate statechart diagrams, they express the additional information in a form of pre and post conditions, expressed in Object Constraint Language (OCL). This additional information is added to the sequence diagrams to guide the synthesis process. Damm *et al.* have presented Life Sequence Charts (LSCs) [15], which have been synthesized from state-based systems by Harel *et al.* [24]. In [9], Bontemps *et al.* have also used LSCs as a basis for the behavior synthesis.

The well-known problem of over-generalization can be handled in various ways. In [39,40], MAS handles this prior to or during the synthesis when run in an interactive mode. The over-generalization problem has been acknowledged (e.g. by Uchitel *et al.* [67]). In [67], Uchitel *et al.* call unexpected interactions as “implied scenarios”. They have presented an algorithm which synthesizes a behavior model as a labeled transition system (LTS), describing the closest possible implementation based on basic and high-level Message Sequence Charts (MSCs). In addition, they propose a technique for detecting the implied scenarios. Another way to tackle the over-generalization problem is to run the scenarios against the existing behavior models to check the correctness of either the initial scenarios or the behavior specification.

In [26], Harel *et al.* have presented an approach to execute and analyze scenario-based behavior. The approach aims at helping the modeller to refine and possibly correct the behavior specifications. Their so-called "Play-Out" algorithm executes scenarios and keeps track of user and system events. The algorithm is implemented in the Play-Engine tool [25].

9.1.2 Scenario-based approaches for constructing service compositions

Some scenario-based approaches for defining simple Web service communication have been presented in the literature. In [11], Casella *et al.* have used UML sequence diagrams to define agent communication. This approach only supports simple message exchange. Messages produce only invoke and receive activities in a WS-BPEL description. The approach does not aim at constructing any complicated WS-BPEL structures.

A template-based methodology for Web service compositions construction has also been proposed by Karastoyanova *et al.* in [33]. They use coordination protocols to define Web service communications. A protocol contains one coordination controller role, which is responsible for communication with the workflow engine. Interactions are marked with a keyword, which defines the type of activity. A coordination protocol defines a possible message sequence and thus it is quite close to sequence diagram notation, which is also used in our approach.

The approaches presented in [11] and [33] do not include synthesis of different scenarios nor provide support for use of any control structures. Instead of business process models, the approaches can be used to express simple interaction and service compositions.

9.2 Business process development

9.2.1 Process modeling using UML state and activity models

In [19], Evans *et al.* propose a unified superstructure for UML state machines and activity diagrams. This means that they provide mapping between UML state machine and activity constructs. The proposed approach is general purpose and independent of any domain or application context. In our approach, a direct transformation from a state machine into an activity model does not provide a result compatible with WS-BPEL profile. Thus, in our approach domain-specific mappings are defined, meaning that the resulting activity diagram is created according to WS-BPEL profile rules and it is customized with WS-BPEL specific stereotypes and structures. The resulting activity model thus supports generation of WS-BPEL descriptions.

In some approaches, state machines have been used for modeling business processes. State-oriented approaches have proven to be especially successful for modeling high level human-driven workflows as described in [59] by Shi *et al.*. They have also implemented a model transformation from state diagrams to WS-BPEL. However, UML activity diagram notation provides many of the workflow modeling constructs and is closer to BPMN. It is often used for process modeling by IT professionals. Mainly for these reasons, activity models are used for workflow modeling in this study.

9.2.2 Process and decision mining

As explained in [69, 71] by van der Aalst *et al.*, business process mining aims at the automatic construction of process models based on monitoring existing systems. In process mining, the process behavior is identified from the observed event log. The event log, i.e. the trace, defines the possible execution paths, which are merged into a state machine presentation. In the scenario-driven business process modeling method, it is not expected to have existing and running services or processes already available. Still, process mining has similar motivation to the scenario-driven method and similar techniques are used.

In scenario mining, invisible tasks, which belong to a process model but which cannot be found from the event log, must be usually constructed manually. Similarly, in this study copying of process variables and evaluation criteria for decision nodes must, for example, be created manually.

Decision mining aims at automatically detecting decision points, i.e. how data dependencies affect the process flow. Decision mining approaches are usually applied to initial process model, which might be a result of applying some process mining algorithm. The effect of the data attributes values on the process routing is studied in [52] by Rozinat *et al.*. In the study, decision points are turned into a learning problem and machine learning techniques, such as decision tree, have been applied to Petri net based models.

In [70], van der Aalst *et al.* decision mining techniques have been applied for restructuring of WS-BPEL specifications into more structured and readable form. In the case of Sketch, detection of WS-BPEL specific patterns can also be considered a type of restructuring.

Rozinat *et al.* have also applied process and decision mining techniques for conformance checking as described in [53]. The approach aims at finding inconsistencies between the event log and the corresponding process model.

9.2.3 WS-BPEL generation from graph-based models

Several transformation approaches to generate WS-BPEL code from graphical models like BPMN and UML activity models exist [10, 27, 31, 32, 38]. In this thesis, the BPELGen tool to generate WS-BPEL descriptions from UML activity and class models is presented. Also, a UML profile for WS-BPEL is defined in detail in [51]. By following the profile, a UML model can be used as a source for transformation into WS-BPEL. To enable the scenario-driven approach, the BPELGen tool is integrated with the Sketch tool.

However, graph-based business process models are not automatically compatible with XML-based block-oriented WS-BPEL. The issue of handling unstructured models has been studied earlier. In [7], L. Garciano-Banuelos uses SQPR-tree decomposition techniques for translating BPMN models to WS-BPEL code. In [58], C. Sandberg presents a tree-based calculation approach for eliminating unstructured loops. Tree-based calculation produces several distinct parallel branches in the process model. A drawback to this is that some information might get lost or the result is not very intuitive. Especially, if some manual process refinement is needed readable process models are highly valued. Thus, Sketch tool does not further transform unstructured state machines.

9.3 Service-oriented software development

IBM has proposed a Service-Oriented Modeling and Architecture (SOMA) [4, 5] framework for service-oriented development. It includes identification, specialization, and realization of services to be used to form composite services and business processes. However, SOMA describes an abstract framework rather than a concrete method or tool support.

OMG has proposed a specification of Service oriented architecture Modeling Language (SoaML) [50]. SoaML consists of an extension to UML to support model-driven development of SOA. The specification does not propose any particular development method, but instead proposes a consistent way of describing the service consumer and the provider concepts, as well their interaction and the agreements between them. For specification, several UML diagram types are used, including collaboration and sequence diagrams. SoaML supports modeling requirements for service-oriented architectures, including the system specification, the specification of individual service interfaces, and the specification of service implementations. The approach supports the generation of derived artifacts based on a common metamodel and a UML profile. Realization of this approach still remains to be achieved by tool vendors.

In comparison to above mentioned approaches, the scenario-driven approach describes a concrete method and implementation. In addition to tool support, it includes description of the conducted case studies.

A Domain Specific Language (DSL) is a language designed specifically to express concepts in a specific domain. Applying DSLs in the context of SOA have been studied [44]. In the scenario-driven approach domain concepts are presented as stereotyped UML elements (process, WSDL, etc.). Compared to DSLs, the scenario-driven approach is a lightweight approach utilized using existing UML editors, UML2, and UML profiling mechanism. Thus, the approach is not dependent on a specific DSL tool, but the modeling can be done with any UML editor supporting the UML2 metamodel.

Chapter 10

Summary of the included publications

This thesis includes five publications. In this chapter, the publications are summarized and the author's contribution to each of the publications is defined.

[I] In the paper *Variation Needs in Service-Based Systems*, a development process for a provider, called SPDP, is presented. SPDP provides a framework for a new service-based products. In addition, variation needs in the service-oriented development are discussed in the context of the SPDP phases. As an example application, development for an eCard product is presented. The author of this thesis is one of the main authors of this paper. In particular, the author was responsible for constructing the eCard service example.

[II] In the paper *Scenario-Driven Approach for Business Process Development*, a scenario-driven approach for WS-BPEL based business process development is presented. Here, simple scenarios are used to present functional business requirements. They are synthesized into a state machine, which is further transformed into a WS-BPEL specific workflow model. The resulting workflow model, given as a UML activity model, enables mapping into WS-BPEL code. In the paper, a prototype tool, called Sketch, is presented. The Sketch tool supports the scenario synthesis and construction of WS-BPEL compatible models. The paper includes an industrial case study. The author of this thesis is the main author of this paper, and she has developed the method introduced in the paper, as well as designed and implemented the Sketch tool, and conducted the case study. The second author, Timo Kokko, has participated in planning the case study and analyzing the results.

[III] In the paper *Modeling and Generating Mobile Business Processes*, a model-driven approach for modeling and generating WS-BPEL descriptions for mobile business processes is presented. As a modeling notation, UML class and activity diagrams are used. The paper defines rules for mapping UML models into WS-BPEL code, as well as developed tool support for generating WS-BPEL and WSDL descriptions from UML models. The proposed method was developed in collaboration by the author of this thesis and Lasse Pajunen. The author has implemented a prototype tool called BPELGen. She is also the main author of the paper. The work has been carried out at NRC. Lasse Pajunen was the author's supervisor and project manager at NRC.

[IV] In the paper *Constructing Usage Scenarios for API Redocumentation*, an automatic approach for scenario mining from application traces is presented. To focus on a particular type of interaction, a filtering model is introduced. It is used to filter out uninteresting message exchange. The author of this thesis has developed the proposed method with Johannes Koskinen and Juanjuan Jiang. The author was also responsible for conducting the case study. In addition, she has developed part of the tool support presented in the paper.

[V] In the paper *A Pattern-Based Approach to Generate Code from API Usage Scenarios*, a scenario-based approach for code generation is presented. API rules, presented with usage scenarios, are transformed into reusable generative patterns, which can be applied to generate applications stub code. The author has developed the approach presented together with Johannes Koskinen. The author has implemented the tool support presented in this paper, that is the transformation from usage scenarios into generative Java patterns.

Chapter 11

Conclusions

In this chapter, a summary of the thesis contents is presented. In addition, answers to the research questions are provided.

11.1 Thesis summary

One of the motivating factors for the study was to apply a simple method of sketching and applying functional system requirements to the development of service-based systems. The research interests in particular include a better understanding and integration of the development phases by constructing systematic development methods. As a solution, the scenario-driven approach for development of service-based systems has been proposed. Applicability and benefits of the approach has been studied by applying constructive research methods and conducting case studies.

In this thesis, the development process for service-based systems has been studied. The results are summarized in a description of Service Product Development Process (SPDP). Furthermore, the scenario-driven development approach for service-based systems, called SceDA, is developed. SceDA includes three independent scenario-based methods to support development of service-based systems in different development phases. Each method is presented with a practical application; a case study and description of the developed tool support.

SceDA is presented in Figure 11.1. It includes the development process and the scenario-based methods developed. The application of the methods cover development of business processes as well as individual services. Business analysis is considered as a prerequisite, that is essential for defining the need for the building of a system or a product. Thus, business requirements are considered as an input for SceDA. The output from the framework is a service-based product or system.

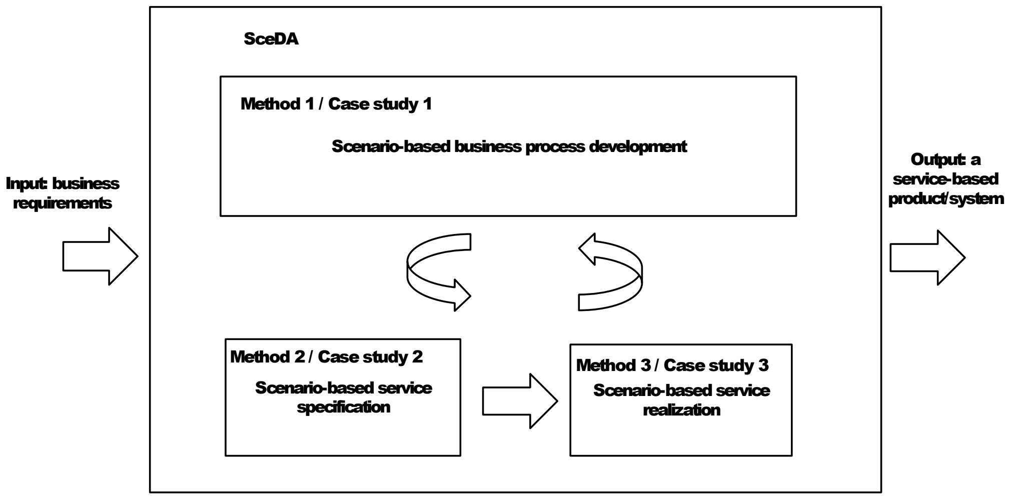

SceDA
Method 1 / Case study 1
Scenario-based business process development
Input: business requirements
Output: a service-based product/system
Method 2 / Case study 2
Scenario-based service specification
Method 3 / Case study 3
Scenario-based service realization

Figure 11.1: SceDA

In this thesis it has been shown that the scenario-driven approach can be applied to different development phases. Furthermore, the approach not only supports the particular development activity, but also provides better integration of the development activities and propagation of the requirements.

11.2 Research questions revisited

The research questions presented in Section 1.2 are revisited below. Answers to these and a summary of the thesis contributions are included.

RQ1 What activities are involved in a typical development process for service-based systems, and what kind of methods and tool support can be built to support the development activities?

A1: A typical development process for service-based systems is identified. The process consists of six service-level phases summarized as a description of Service Product Development Process (SPDP).

Due to the separate needs and challenges of different development phases, we have described scenario-based methods M1, M2, and M3, which address business processes development, code generation, and mining of development rules and service interfaces from existing systems. The methods presented cover the different development phases, namely, product and service specification, service identification, and realization of services and processes. Each of the methods have been applied in a case study, CS1, CS2, and CS3 respectively. In addition, tool support for each method has been developed.

Through application of SceDA, better integration and consistency of the development process can be achieved.

RQ2 How can the scenario-driven approach be applied to business process development?

A2: In case study CS1, scenario-driven business process development has been applied. The method developed, M1, enables easy sketching of business process requiring no knowledge of state-based design and control structures. Initially, functional business process requirements are presented as simple scenarios. Then, the scenarios are synthesized and transformed into an initial business process model. Finally, the initial model can be refined and transformed into a WS-BPEL description.

RQ3 How can the scenario-driven approach be applied to services specification and re-documentation?

A3: Case study CS2 shows that scenarios can be used to capture development rules for services. The method developed, M2, presents an automated method for mining development and design rules from application traces.

To identify a certain type of interaction, the filtering model introduced can be used to automatically remove unnecessary information from application traces. Possible applications of M2 include mining of service interfaces in order to support service identification.

RQ4 How can the scenario-driven approach be applied to realization of individual services, i.e. service implementation?

A4: The research question is studied in the context of case study CS3. The method developed, M3, can be used to transform the system development rules, given as scenarios, into reusable and generative patterns. The patterns can be applied in the service realization phase for generating stub code for the application, or, alternatively, for service interfaces.

11.3 Future work

In the future, the author would like apply the scenario-driven approach in the context of a particular development environment, for example to integrate Sketch with some existing integrated development environment. The author is especially interested in applying the scenario-driven approach in cloud computing. In addition to end-user services, cloud computing involves development and utilization of platforms and tools as service-based systems. It is thus assumed that the typical development activities and challenges differ from a traditional SOA project as well as the development methods.

The author is interested in conducting case studies in the cloud computing environment to gather requirements for further development of the Sketch tool and to acquire more information on the possible applications of the scenario-driven approach. Further development of the approach and tools would benefit on usability studies, better integration of the tools, and further evaluation on industrial SOA projects.

Bibliography

[1] D. Alur, D. Malks, and J. Crupi. *Core J2EE Patterns: Best Practices and Design Strategies.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[2] J. Amsden, T. Gardner, C. Griffin, and S. Iyengar. Draft UML 1.4 Profile for Automated Business Processes with a mapping to BPEL 1.0. Version 1.1, April 2004. http://www128.ibm.com/developerworks/rational/library/. Last visited December 2011.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, November 1987.

[4] A. Arsanjani. Service-Oriented Modeling and Architecture: How to Identify, Specify and Realize Services for your SOA, 2004. On-line at http://www.ibm.com/developerworks/. Last visited December 2011.

[5] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah. S3: A Service-Oriented Reference Architecture. *It Professional*, 9(3):10–17, 2007.

[6] L. G. Azevedo, F. Santoro, F. Baião, J. Souza, K. Revoredo, V. Pereira, and I. Herlain. A method for service identification from business process models in a SOA approach. In *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 99–112. Springer Berlin Heidelberg, 2009.

[7] L. G. Banuelos. Translating BPMN models to BPEL code. In *GraBaTs 2009: Graph-Based Tool Contest - Solutions To Case Studies*, 2009.

[8] N. Bieberstein, R. G. Laird, K. Jones, and T. Mitra. *Executing SOA: A Practical Guide for the Service-Oriented Architect.* IBM Press, 1st edition, 2008.

[9] Y. Bontemps. *Relating Inter-Agent and Intra-Agent Specifications (The Case of Live Sequence Charts)*. 2005. PhD Thesis.

[10] Business Process Modeling Initiative, http://www.bpmi.org/. *Business Process Modeling Language*, 2002. Last visited December 2011.

[11] G. Casella and V. Mascardi. From AUML to WS-BPEL. In *Tech. Rep. DISI-TR-06-01*, 2006. http://www.disi.unige.it/person/MascardiV/Download/DISI-TR-06-01.pdf, last visited December 2011.

[12] J. Castro, M. Kolp, and J. Mylopoulos. A Requirements-Driven Development Methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE '01)*, pages 108–123, London, UK, 2001. Springer-Verlag.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[14] G. Coticchia. Seven Steps to a Successful SOA Implementation. *Business Integration Journal*, 10(5):10–13, 2006.

[15] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[16] Eclipse, http://www.eclipse.org. *Eclipse WWW site*, 2011. Last visited December 2011.

[17] The Eclipse Foundation, http://www.eclipse.org/m2m/atl/. *Atlas Transformation Language*, 2007. Last visited December 2011.

[18] The Eclipse Foundation, http://www.eclipse.org/tptp/. *The Eclipse Test & Performance Tools Platform*, 2011. Last visited December 2011.

[19] A. Evans, P. Sammut, J. S. Willans, A. Moore, and G. M. Rama. A Unified Superstructure for UML. *Journal of Object Technology*, 4(1):165–182, 2005.

[20] N. Fareghzadeh. Service identification approach to SOA development. In *World Academy of Science, Engineering and Technology*, volume 45, pages 258–266. Springer Berlin Heidelberg, 2008.

[21] Google, http://code.google.com. *Google APIs*, 2011. Last visited December 2011.

[22] I. Hammouda. *Model-driven Software Development - Volume II of Research and Practice in Software Engineering*, chapter A Tool Infrastructure for Model-Driven Development Using Aspectual Patterns, pages 139–178. Springer, 2005.

[23] I. Hammouda, A. Ruokonen, M. Siikarla, A. L. Santos, K. Koskimies, and T. Systä. Design profiles: toward unified tool support for design patterns and UML profiles. *Softw. Pract. Exper.*, 39(4):331–354, March 2009.

[24] D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. *International Journal of Foundations of Computer Science*, 13(1):5–51, Febuary 2002.

[25] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[26] D. Harel and R. Marelly. Specifying and executing behavioral requirements: the play-in/play-out approach. *Software and System Modeling*, 2(2):82–107, 2003.

[27] IBM, alphaWorks, http://www.alphaworks.ibm.com/. *Emerging Technologies Toolkit (ETTK)*, 2006. Last visited December 2011.

[28] P. Järvinen. *On Research Methods*. Tampereen yliopistopaino Oy, Tampere, Finland, 2004.

[29] J. Jiang, A. Ruokonen, and T. Systä. Pattern-based Variability Management in Web Service Development. In *Proceedings of the Third European Conference on Web Services*, ECOWS '05, pages 83–94, Washington, DC, USA, 2005. IEEE Computer Society.

[30] J. Jiang and T. Systä. UML-Based Modeling and Validity Checking of Web Service Descriptions. In *Proceedings of 2005 IEEE International Conference on Web Services, ICWS 2005*, pages 453–460, July 2005.

[31] A. Kalnins, J. Barzdins, and E. Celms. Model Transformation Language MOLA. In U. Aßmann, M. Aksit, and A. Rensink, editors, *MDAFA*, volume 3599 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2004.

[32] A. Kalnins and V. Vitolins. Use of UML and model transformations for workflow process definitions. In *Proceedings of Baltic DBIS 2006*, pages 3–14, 2006.

[33] D. Karastoyanova and A. Buchmann. A Methodology for Development and Execution of Web Service-based Business Processess. In *Proceedings of 1st Australian Workshop on Engineering Service-Oriented Systems, Melbourne*, 2004.

[34] K. Klose, R. Knackstedt, and D. Beverungen. Identification of services - a stakeholder-based approach to SOA development and its application in the area of production planning. In *European Conference on Information Systems (ECIS)*, pages 1802–1814. AIS, 2007.

[35] T. Kokko, J. Antikainen, and T. Systä. Adopting SOA - Experiences from nine Finnish organizations. In *CSMR'09: Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, pages 129 – 138, London, UK, 2009. IEEE Computer Society.

[36] N. Kulkarni and V. Dwivedi. The role of service granularity in a successful SOA realization - a case study. In *Proceedings of the 2008 IEEE Congress on Services - Part I*, SERVICES '08, pages 423–430, Washington, DC, USA, 2008. IEEE Computer Society.

[37] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage. SMART: The Service-Oriented Migration and Reuse Technique. 2005. Technical report CMU/SEI-2005-TN-029.

[38] C. Lohmann, J. Greenyer, J. Jiang, and T. Systä. Applying Triple Graph Grammars for Pattern-Based Workflow Model Transformations. In *Proceedings of TOOLS 2007, JOT*, pages 253–273, 2007.

[39] E. Mäkinen and T. Systä. *Minimally adequate teacher designs software.* April 2000. Report A-2000-7.

[40] E. Mäkinen and T. Systä. *Implementing minimally adequate synthesizer.* June 2000. Report A-2000-9.

[41] OASIS, http://www.oasis-open.org/. *Web Services Business Process Execution Language Version 2.0*, 2005.

[42] OASIS, http://www.oasis-open.org/. *Reference Model for Service Oriented Architecture 1.0*, 2006. Last visited December 2011.

[43] OASIS, http://www.oasis-open.org/. *Organization for the Advancement of Structured Information Standards*, 2011. Last visited December 2011.

[44] E. Oberortner, U. Zdun, and S. Dustdar. Domain-specific languages for service-oriented architectures: An explorative study. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet*, ServiceWave '08, pages 159–170, Berlin, Heidelberg, 2008. Springer-Verlag.

[45] Object Management Group, Inc. *OMG Model Driven Architecture, Version 1.0.1*, 2003.

[46] Object Management Group, Inc. *Unified Modeling Language: Superstructure version 2.0 Final Adopted Specification ptc/03-08-02*, August 2003.

[47] Object Management Group, Inc. *Unified Modeling Language Specification, Version 2.0.*, May 2005.

[48] Object Management Group, Inc., http://www.bpmn.org/. *Business Process Modeling Notation (BPMN) Specification Final Adopted Specification 06-02-01*, 2006.

[49] Object Management Group, Inc. *Meta Object Facility (MOF), Version 2.0.*, January 2006.

[50] Object Management Group, Inc. *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*, 2009. OMG Adopted Specification, Finalisation Task Force Beta 2 (FTF Beta 2), ptc/2009-12-09.

[51] L. Pajunen and A. Ruokonen. Modeling and Generating Mobile Business Processes. In *IEEE International Conference on Web Services, 2007. ICWS 2007*, pages 920–927, July 2007.

[52] A. Rozinat and W. M. P. van der Aalst. Decision Mining in ProM. *Lecture Notes in Computer Science : Business Process Management, Volume 4102, 2006*, pages 420–425, 2006.

[53] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[54] J. E. Rumbaugh. Getting Started: Using Use Cases to Capture Requirements. *JOOP*, 7(5):8–12, 1994.

[55] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. 10.1007/s10664-008-9102-8.

[56] A. Ruokonen, I. Hammouda, and T. Mikkonen. Enforcing Consistency of Model-Driven Architecture Using Meta-Designs. In *European Conference on MDA - Foundations and Applications: Workshop on Consistency in Model Driven Engineering (C@MoDE 2005)*, pages 127–141, Nuremberg, Germany, 2005.

[57] A. Ruokonen, L. Pajunen, and T. Systä. On Model-Driven Development of Mobile Business Processes. In *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications, SERA*, pages 59–66, Washington, DC, USA, 2008. IEEE Computer Society.

[58] C. Sandberg. Elimination of Unstructured Loops in Flow Analysis. In *WCET 2003 Workshop*, pages 51–55, July 2003. Last visited December 2011.

[59] W. Shi, J. Wu, S. Zhou, L. Zhang, Y. Yin, and Z. Wu. Facilitating the Flexible Modeling of Human-Driven Workflow in BPEL. In *Proceedings of Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008*, pages 1615 – 1624, Gino-wan, Okinawa, Japan, 2008. IEEE Computer Society.

[60] M. Siikarla. *A Light-Weight Approach to Developing Interactive Model Transformation.* PhD thesis, Tampere University of Technology, Finland, 2011.

[61] H. Sneed. Wrapping legacy software for reuse in a SOA. In *Multikonferenz Wirtschaftsinformatik (2006)*, volume 2, pages 345–360. Citeseer, 2006.

[62] Sun MicroSystems. *The J2EE 1.4 Tutorial For Sun Java System Application Server Platform Edition 8.2*, 2006.

[63] Sun MicroSystems. *Java Web Services Developer Pack (Java WSDP) Version 2.0*, 2006.

[64] R. Tarjan. Testing Flow Graph Reducibility. *Journal of Computer and System Sciences*, 9(3):355–365, 1974.

[65] The TeleManagement Forum (TM Forum). *Business Process Framework (eTOM).* Last visited December 2011.

[66] S. R. Tilley, D. B. Smith, and H. A. Müller. Migrating to SOA: approaches, challenges, and lessons learned. In J. W. Ng, C. Couturier,

H. A. Müller, and A. G. Ryman, editors, *CASCON*, pages 371–373. ACM, 2010.

[67] S. Uchitel, J. Kramer, and J. Magee. Synthesis of Behavioral Models from Scenarios. *IEEE Trans. Softw. Eng.*, 29(2):99–115, 2003.

[68] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

[69] W. M. P. van der Aalst. 06291 workshop report: Process mining, monitoring processes and services. In F. Leymann, W. Reisig, S. R. Thatte, and W. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[70] W. M. P. van der Aalst and K. Bisgaard Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Inf. Softw. Technol.*, 50(3):131–159, 2008.

[71] W. M. P. van der Aalst, H. A. Reijers, A. Weijters, B. van Dongen, A. Alves de Medeiros, M. Song, and H. Verbeek. Business process mining: An industrial application. *Inf. Syst.*, 32(5):713–732, 2007.

[72] D. Varró. Model Transformation by Example. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin / Heidelberg, 2006.

[73] W3C, http://www.w3.org/TR/wsdl. *Web Services Description Language (WSDL) 2.0*, 2001. Last visited December 2011.

[74] W3C, http://www.w3.org/. *Simple Object Access Protocol (SOAP) 1.2*, 2007. Last visited December 2011.

[75] W3C, http://www.w3.org/. *World Wide Web Consortium*, 2011. Last visited December 2011.

[76] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd international conference on Software engineering*, pages 314–323. ACM Press, 2000.

[77] R. Yin. *Case study research: design and methods.* Applied social research methods series. Sage Publications, 2003.

[78] T. Ziadi, L. Hélouët, and J.-M. Jézéquel. Revisiting Statechart Synthesis with an Algebraic Approach. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 242–251, Washington, DC, USA, 2004. IEEE Computer Society.

[I] A. Ruokonen, V. Räisänen, M. Siikarla, K. Koskimies and T. Systä, Variation Needs in Service-Based Systems, The 6th IEEE European Conference on Web Services. In *The 6th IEEE European Conference on Web Services (ECOWS 2008)*, pp. 115 - 124, Dublin, Ireland, 2008. ©2008 IEEE

[II] A. Ruokonen, T. Kokko, and T. Systä, Scenario-Driven Approach for Business Process Development. *International Journal of Business Process Integration and Management (IJBPIM)*, vol. 6(2012) No 1, pp. 77 - 96. ©2012 Inderscience Publishers

[III] L. Pajunen, and A. Ruokonen, Modeling and Generating Mobile Business Processes. In *IEEE 2007 International Conference on Web Services (ICWS 2007)*, pp. 920-927, Salt Lake City, Utah, USA, 2007. ©2007 IEEE

[IV] J. Jiang, J. Koskinen, A. Ruokonen, and T. Systä, Constructing Usage Scenarios for API Redocumentation. In *The 15th IEEE International Conference on Program Comprehension (ICPC 2007)*, pp. 259 - 264, Banff, Alberta,Canada, 2007. ©2007 IEEE

[V] J. Koskinen, A. Ruokonen, and T. Systä, A Pattern-Based Approach to Generate Code from API Usage Scenarios. In *Nordic Journal of Computing (NJC'06)*, vol. 13(2006), pp. 162 - 179. ©2006 Nordic Journal of Computing

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-2975-7
ISSN 1459-2045