Modularization and Specification of Service-Oriented Systems

Modularization and Specification of Service-Oriented Systems

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. K. C. A. M. Luyben
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 5 juli om 15:00 uur

door

Linda Iris TERLOUW
ingenieur in de technische informatica en bedrijfsinformatietechnologie
geboren te Tilburg

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. J. L. G. Dietz

Samenstelling promotiecommissie:

Rector magnificus,	Technische Universiteit Delft, voorzitter
Prof. dr. ir. J. L. G. Dietz,	Technische Universiteit Delft, promotor
Prof. dr. ir. A. Verbraeck,	Technische Universiteit Delft
Prof. dr. R. J. Wieringa,	Universiteit Twente
Prof. dr. J. Verelst,	Universiteit van Antwerpen
Prof. dr. ing. J. B. F. Mulder MBA,	Universiteit van Antwerpen
Prof. dr. H. A. Proper,	Radboud Universiteit Nijmegen
Dr. dipl.-ing. A. Albani,	Universiteit van St. Gallen
Prof. dr. Y. H. Tan,	Technische Universiteit Delft, reservelid

Verspreid door:
Linda Terlouw
Martinus Nijhoffhove 2
3437 ZR Nieuwegein
Nederland
linda.terlouw@icris.nl

ISBN: 978-94-6108-182-7

SIKS Dissertatiereeks nr. 2011-21
Het in dit proefschrift vermelde onderzoek is uitgevoerd onder de auspiciën van SIKS, de Nederlandse School voor Informatie- en KennisSystemen.

Druk: Gildeprint Drukkerijen - Enschede
Omslagontwerp: Jeroen Advocaat
Gezet in LaTeX, diagrammen in OmniGraffle

Acknowledgments

Pursuing a PhD is a painful, yet rewarding experience. Every PhD student and promotor will confirm that the only good dissertation is a done dissertation. After six years of research I am happy to present the final results. The research is conducted at the Delft University of Technology in a research group consisting of hybrids; people who work in industry with an academic interest and people who work in academia with a wish to put theory into practice. Combining research with a ‘day job’ as IT consultant is a drawback as well as a benefit. The drawback is the chronic lack of time and focus. The benefit is a continuous reality check on the relevance of the research.

Though writing a dissertation seems like a very lonely process, many people in fact contributed to it. First of all, I thank my promotor Jan Dietz, who guided me through the jungle of academic research. My daily supervisor, Antonia Albani, taught me the very specific writing style used in academic papers and gave valuable feedback on my draft papers that helped in getting the papers accepted. Also, I would like to express my gratitude to the other members of my dissertation committee for reading the dissertation and providing feedback.

The CIAO! doctoral consortia gave me an opportunity to present my work in an early phase to the CIAO! board members and the fellow doctoral students. Presenting at such an event was never easy; interruptions started at the first slide and razor-sharp comments were made by the board members. But these events contributed highly to the selection of an appropriate research topic and to building the research on strong theoretical foundations.

In this dissertation three case studies are presented. I would like to thank the case study participants at the Port of Rotterdam, De Lage Landen and Air France/KLM for giving me the opportunity to validate my findings in practice and for providing valuable new insights. Especially, I would like to thank the persons who took the time and effort to arrange the interviews and workshops: Edwin de Werk and Kees Eveleens Maarse (Port of Rotterdam), Rens Voogt (De Lage Landen), and Frank Rappange (Air France/KLM). TeamSupport supported these case studies by providing a license for group decision support software.

I would like to express my appreciation to Paul Weghorst, Reggie Maas-Schellekens, and Eric van der Ende who supported my PhD research when I was working at Ordina. Furthermore, I am grateful to all the members of the former SOA Focus Team of Ordina for all the interesting discussions. Thanks Bart Pruijn, Art Ligthart, Jan-Willem Hubbers, Marco Brattinga, Willem van Pruijssen, Hermen Grievink, and Lex Haket.

For giving tips and tricks on how to combine life as a PhD student and an entrepreneur I would like to thank Wiebe Hordijk.

My parents always supported me in my academic activities and encouraged me to bring out my best. I owe my career as an IT consultant as well as my (just-starting) academic career to them. Also, I would like to thank my brother Joeri with whom I had the pleasure of writing a brother-sister paper.

Finally, I would like to thank my husband André for standing by me at all times during this PhD process and for accepting the consequences of PhD research in terms of time and attention.

Linda Terlouw, 2011

Summary

Modularization and Specification of Service-Oriented Systems

Rationale and Objective

Modern day enterprises face a dilemma. On the one hand they want to be able to respond rapidly to market changes, i.e. they want to be agile. On the other hand they want (or actually need) to introduce complexity in their organization because of strategic choices like the mass-customization of products and services, the introduction of more complex products and services, and the participation in interorganizational networks. For example, getting an overview of their business processes and their supporting software systems is quite a challenge due to their sheer size and their interwoven structure. An even bigger problem is that 'the rules' for designing enterprises are still ill-understood or at least ill-documented.

In general systems theory modularity is proposed as a means for dealing with complexity. The objective we wanted to achieve through this dissertation is to provide practitioners with a better understanding of how to deal with modularity of enterprises and their supporting software systems by applying service-orientation. Currently, most practitioners (usually enterprise architects) make decisions based on gut feeling and experience. Our intention was to make this design knowledge more fundamental and explicit. We based our research on literature from the field of software engineering as well as from the organizational sciences.

First, we sought to find criteria for decomposing service-oriented systems into coarse-grained modules. These modules can comprise humans and/or software systems. So we did not focus on how to structure a single software system, but we focused on how to structure the complete set of services of an enterprise. Advantages of introducing this modularity are, among others, making the total enterprise more comprehensible and minimizing the effect that changes in one module may have on other modules. To minimize interdependencies between modules it is important to take into account the principle of *'maximum cohesion and minimal coupling'*. This led to the question *how* we can conform to this maximum cohesion and minimal coupling principle. And are any other criteria important for module identification? Moreover, are these criteria only software engineering principles or maybe (also) organizational criteria? We answered these questions in our 'laboratory', i.e. real-life, large organizations with complex IT environments.

Second, we saw that current service specification approaches are very immature. This poses a problem, because confusion arises when parties have different interpretations of each other's syntax, semantics, or responsibilities. For example, providers and consumers will not be able to exchange any useful information if one party only speaks English (or XML) and the other only Dutch (or EDIFACT). Also, problems can occur when the provider thinks the height of a product is in centimeters while the consumer specifies height in inches. Or maybe provider and consumer do not call the same side of a product 'height', because it can be positioned on the floor in different ways. Another type of problem comes into life when the consumer expects the provider to deliver a product at the highest possible quality level while the provider actually delivers the cheapest, low-quality product. To structure the information about a service we wanted to design a specification framework. This framework supports the provider in describing his services by stating what aspects need to be specified to enable the service consumer to find the service, to access it, and to judge whether or not the service meets his requirements. We derived the framework from the Ψ-theory, in order to base it on appropriate scientific foundations.

Summarizing, we aimed at two things: (i) to formulate criteria for delimiting coarse-grained modules of a service-oriented system and (ii) to design a framework for specifying services.

Research Approach

For the design of the service specification framework we followed the design science research methodology. The theoretical part of our research is based on the notions of Enterprise Ontology and Enterprise Architecture, as adopted by the Enterprise Engineering community (www.ciaonetwork.org). We designed a service specification framework based on the Ψ-theory, the theory that underlies the notion of Enterprise Ontology. The Generic System Development Process (GSDP), which clarifies the notion of architecture, gives guidance in the design process by proposing a clear and consistent terminology. We specialized the GSDP for service-orientation, the most recent paradigm for information system modularity. Subsequently, we verified whether practitioners agree that the service specification framework is complete and that it does not contain irrelevant aspects. In our case studies we used semi-structured interviews, because written questionnaires about complex matters are often not filled in very thoughtfully ('just get it done with'). Also, we do not want to structure the interviews completely to give the interviewees the chance to elaborate on what they think is important. A last step in conducting the case studies was to organize a workshop. During this workshop the interviewees could respond to each other's ideas.

We did not only use our case studies for evaluating the service specification framework, but also for deriving criteria for modularization from practice. Our goal was to get an overview of criteria that are important for the identification of coarse-grained modules of service-oriented systems. Though we used BCI-3D as a starting point, our goal was not to validate BCI-3D as an identification method. Instead, we tried to discover criteria that can be used as an input for BCI-3D (for determining the weights). So we did not only use our case studies for the evaluation step in the design science research methodology; they were also exploratory case studies (Yin, 2002).

Results

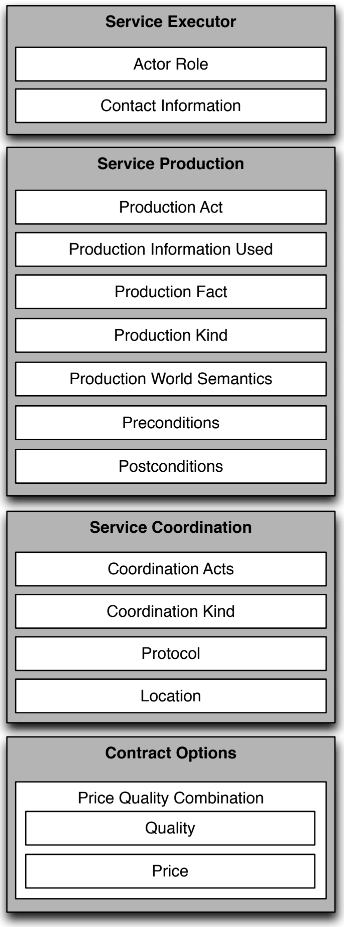
Because the Ψ-theory describes the interaction between a requesting party and an offering party in a formal way, it provides a basis for formalizing the notion of service. We defined a service using the complete transaction pattern as a basis. Though a service has many similarities with a transaction in the Ψ-theory, they are not equal. While the transaction includes all acts of the initiator and the executor, the service concept only regards the executor side. We therefore defined a service as a part of a transaction rather than a whole transaction. So a service is a pattern of coordination and production acts, performed by the executor of a transaction for the benefit of its initiator, in the order as stated in the complete, universal pattern of a transaction. When implemented it has the ability:

- to get to know the coordination facts produced by the initiator and
- to make available to the initiator the coordination facts produced by itself.

We made a distinction between the following types of services: ontological human services, infological human services, datalogical human services, ontological IT services, infological IT services, and datalogical IT services. In two case studies we asked business architects and technical architects what criteria they value for the delimitation of coarse-grained modules of service-oriented systems. In both case studies architects aimed to find modules with *maximum cohesion and minimal coupling*. In the first case study, at De Lage Landen, these modules were coarse-grained IT modules called *autonomous environments*. In the second case study, at Air France/KLM, these modules were coarse-grained business modules (consisting of people and IT systems) called *domains*. Both companies agreed that coarse-grained IT modules should not be defined based on the boundaries of existing IT systems or COTS systems as this limits the possibility to replace systems. Also, they agreed that the organizational structure is not a good starting point as these structures tend to be unstable and highly influenced by organizational politics. BCI-3D can be used to define coarse-grained business modules as well as coarse-grained IT modules independent of current IT systems and organization structures by applying the principle of maximum cohesion and minimal coupling. However, BCI-3D needs to be extended with design principles for determining the weights of different dependencies. These weights are inputs for the algorithms that determine the modules. In the case studies we found examples of criteria that can determine these weights. For instance, the criterion 'autonomous environments are built around business objects (base administrations)' leads to a relatively high weight for relations between business objects. The criterion 'life cycle decoupling' entails that it should be possible to deliver services independently of the products in which they are used, i.e. services should not be tightly coupled to the product as a whole. This implies that child transactions should not be allocated to the same module as their parent transactions.

Instead, a decomposition structure of modules similar to the transaction tree should be defined. This can be realized by giving a very low weight to relations between initiations among transactions. Especially if one transaction is initiated by multiple other transactions ('reuse of the transaction') the weights should be set low.

The figure below depicts our generic service specification framework, which is derived from the Ψ-theory.



We evaluated our service specification framework in three case studies (the Port of Rotterdam, De Lage Landen, and Air France/KLM). The service specification framework covers the vast majority of aspects that are required in practice, but according to practitioners it needs to be extended with some aspects that cannot be derived from the Ψ-theory. First of all, not only one location is needed, but multiple locations should be specified. If an enterprise develops its own software, either with or without the assistance of an external IT service provider, it generally requires four types of locations: the development, test, acceptance, and production environment. Some of the interviewees referred to this type of information as 'the lifecycle of a service'. Next to this, according to several interviewees our framework should include a versioning aspect. In one of the case studies also the way of interacting with the service (request/reply or pub/sub) was seen as a necessary specification aspect.

Future Research

In this dissertation we did not see service-orientation as a technical paradigm, but as a paradigm for structuring the complete enterprise. Modules of service-oriented systems can be defined using BCI-3D as a method. We gathered criteria for the delimitation of coarse-grained modules in two case studies. Some of these criteria can be used for formulating design principles that define the weights between relationships. These weights are used by the algorithms of BCI-3D. This means that these criteria can be used to tune BCI-3D to the specific wishes of the enterprise. However, we cannot be sure whether the criteria we found are specific to the two enterprises we studied or whether they can be applied to a broader range of organizations. We need to study a large number of companies and see whether or not applying BCI-3D combined with the proposed criteria results in more flexibility for the enterprise.

Another important topic to study in future research is the sensitivity of the weights, i.e. do small changes in the weights lead to large changes in the outcome of the delimitation of the coarse-grained modules. This can be done by applying BCI-3D a large number of times to the same model using small differences in weights and comparing the results.

Furthermore, we designed a generic service specification framework. This framework can be used for specifying human services as well as IT services. It was our intention to determine *what* aspects should be described in a service specification. Future research should focus on *how* these aspects should be specified. How an aspect is specified will be different for human services and IT services. For instance, the location of a human service is usually a physical location or a telephone number, while the location of an IT service is usually a URL. The input of an IT service is usually specified in fields with a certain type (e.g. 'string' or 'integer') and a certain length, while a human service can interpret more free format input descriptions. Sometimes industry standards for human service specification exist. For example, in the Dutch healthcare sector Diagnose Behandel Combinaties (DBC's) are used to define a certain medical treatment and to allocate a price to it. Usually human services are specified using natural language. For IT services more formal descriptions are used. It does not make sense to define a complete new standard for specifying all aspects of the specification framework, because this would be a massive effort and already many standards are available. Instead, it is advisable to look into what existing standards can be used and how they can be combined (if possible). Some of the standards worth investigating are (non-limitative list): UML-OCL and Rule-ML for pre- and postconditions, OWL and ISO/IEC 11179 for semantics, WSDL-S for annotation of input/output structures with semantics, and WSLA and WS-agreement for service level agreements.

When a standardized way of specifying the aspects of the service specification framework (i.e. the 'how part') is available, quantitative research can be conducted. We would like to compare the time required to build services using our service specification framework and using other approaches to service specification. Also, we would like to measure the discovery time for potential consumers of our framework compared to others.

Contents

Part I
Introduction

Chapter 1

Background

In this dissertation we focus on the delimitation of coarse-grained modules and the specification of services of service-oriented systems. In this chapter we explain the central notions of this dissertation, viz. Enterprise Engineering (1.1), modularization (1.2), and Service-Oriented Architecture (SOA) (1.3). These notions form the basis of the central research question posed in chapter 2.

1.1 Enterprise Engineering

Modern day enterprises face a dilemma. On the one hand they want to be able to respond rapidly to market changes, i.e. they want to be agile. On the other hand they want (or actually need) to introduce complexity in their organization because of strategic choices like the mass-customization of products and services, the introduction of more complex products and services, and the participation in interorganizational networks. For example, getting insight into their business processes and their supporting software systems is quite a challenge due to their sheer size and their interwoven structure. An even bigger problem is that 'the rules' for designing enterprises are still ill-understood or at least ill-documented.

Weinberg (2001) explains why we need systems thinking for understanding and (re)designing enterprises and software systems by means of Figure 1.1. Region I, 'organized simplicity' comprises machines, or mechanisms. In this area complexity as well as randomness is low and the behavior of every element can be predicted using an analytical approach. Region II, 'unorganized complexity', comprises populations, or aggregates. In this region the

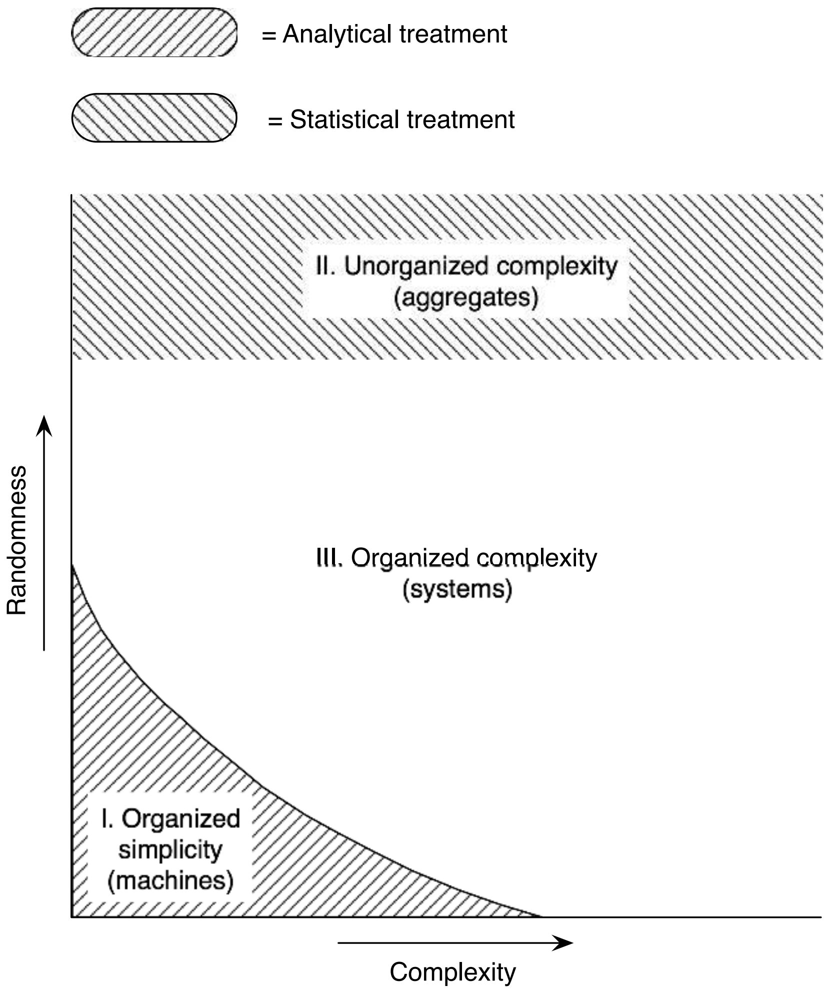

= Analytical treatment
= Statistical treatment
II. Unorganized complexity (aggregates)
III. Organized complexity (systems)
I. Organized simplicity (machines)
Randomness
Complexity

Figure 1.1: Medium numbers of organized complexity (Weinberg, 2001)

amount of randomness is high enough to rely on averages. This justifies taking a statistical approach for making predictions. In region III, 'organized complexity', we face a problem with both the analytical and the statistical approach, because we are dealing with *medium numbers*. In this region we find too much complexity for analysis and too much organization for statistics. Weinberg calls this 'yawning gap in the middle' the region of *systems*. He proposes to take a systems thinking approach for dealing with the organized complexity as found in modern enterprises.

The emerging field of Enterprise Engineering (Dietz and Hoogervorst, 2007) is grounded on systems thinking. It aims at combining (relevant parts from) the traditional organizational sciences and the information systems sciences, and to develop emerging theories and associated methodologies for the analysis, design, engineering, and implementation of future enterprises (Dietz, 2008). Dietz (2008) proposes two fundamental notions as a basis for this field: *Enterprise Ontology* and *Enterprise Architecture*. The notion of Enterprise Ontology and the accompanying *DEMO* methodology (Dietz, 2006b) provide a means for modeling enterprises in a very precise way at a high level of abstraction. By making the ontological model of an enterprise we can quickly get an understanding of the essence of this enterprise. Enterprise Architecture is defined as the whole set of design principles that an enterprise applies in (re-) designing itself. We need this notion to limit design freedom, which would otherwise be undesirable large.

The *Generic System Development Process* (GSDP) (Dietz, 2008) defines the relationships between architecture, functional design, and constructional design for any type of system. It therefore is applicable to enterprises as well as software systems. The Extensible Architecture Framework (xAF) (Dietz, 2008) elaborates on the concept of architecture by providing a generic framework that enables comparison and evaluation of existing frameworks. The GSDP and xAF give guidance in design by proposing a clear and consistent terminology and a way of structuring design principles.

1.2 Modularization in Software/Enterprises

In this dissertation we strive to use service-orientation as a means of dealing with complexity of enterprises and their supporting information systems. We define a service-oriented system as an enterprise, consisting of people and usually (but not necessarily) also IT systems, that offers services to its environment and that is structured in a modular way. In such an enterprise it does not matter whether a certain service is executed by a human being or by an IT system: they both deal with a certain request and provide a result that is described in a service specification (we will provide a definition of the notion of 'service' in chapter 4). To give an overview of the topic modularity we discuss some of the work done in this area the last decades.

Modularization has been proposed in the early days of software engineering by McIlroy (1968) and Parnas (1972). Its advantages are: (i) making the total software system structure more comprehensible, (ii) enabling easy replacement of components of the software system, (iii) making it possible to divide work between groups of developers without them needing to be aware of the structure of the total software system, and (iv) minimizing the effect that changes in one part of the system have on the other parts. Parnas (1972) introduced the criterion of information hiding, i.e. each module hides the manifestation of certain design decisions from the other modules. Interfaces between modules are chosen to reveal as little as possible about their internal operations. Also, he mentions some advisable specific example decompositions, e.g. "a data structure, its internal linkings, accessing procedures and modifying procedures are part of a single module". McIlroy (1968) envisioned a mature software industry in which a purchaser can consult a catalog to search for software components that he can regard as black boxes.

During the '90s the focus of modularization shifted towards finding the right objects (Rumbaugh et al., 1991; Booch, 1994; Jacobson, 1995). Nowadays, we are also dealing with identifying the right components (Vitharana et al., 2003; Levi and Arsanjani, 2002; Albani and Dietz, 2006) and services (McGovern et al., 2006; Zhang et al., 2005; Henkel et al., 2004; Erradi et al., 2006a). Mannaert and Verelst (2009) propose the concept of normalized systems, which is defined as software systems that are stable with respect to a defined set of anticipated changes. By the identification of the 'atoms' of software systems they aim at achieving a better evolvability of software systems. These atoms can be used as building blocks for more coarse-grained modules.

Simon (1969) states that the concept of modularity does not only apply to software systems, but in fact to all man-made artifacts, including enterprises. Te Winkel et al. (2008) elaborate on applying modularity to organizations as a strategy to reduce bureaucracy and thereby enable innovation. The concept of the described cell-based structure was first (and successfully) applied by the Dutch IT services provider BSO, which was later acquired by Philips in 1996 (Wintzen, 2007). Te Winkel et al. (2008) see three advantages of this modularized structure for Topicus, a Dutch IT company. First, it reduces management complexity. Second, the organizational structure becomes flexible as modularity accommodates uncertainty and spin-offs are easier to write off than business units. Third, by creating spin-offs for specific networks, Topicus finds new opportunities. The fourth advantage of organizational modularity the authors mention is being able to work on different parts concurrently. Baldwin and Clark (2000) propose six operators that can be applied to modular systems: splitting, substituting, augmenting, excluding, inverting, and porting. These operators are actions that change existing structures in new structures in well-defined ways. Op 't Land (2008) combined the field of enterprise engineering and modularity and studied the splitting operator. He defines eleven criteria for keeping actors together and formalizes these criteria as organizational construction rules. These actors are subjects fulfilling actor roles. Actor roles are elementary chunks of authority and responsibility Dietz (2006b). The criteria include, for instance, keeping them together if they use the same language/culture or if they need comparable competencies.

1.3 Service-Oriented Architecture

The latest widely used paradigm in information system development is Service-Oriented Architecture (SOA). Many ideas in SOA are not really new and have an origin in component-based design, message-oriented middleware, object-oriented middleware, and workflow. However, two new things attribute to the popularity of SOA. First, the widely accepted web service standards (W3C, 2004) - even though they are still under development - enable specifying the interface of services in a standardized way. Secondly, the concept of service-orientation has a great appeal on business people. The high level of granularity of services and their non-technical description have a better fit to their mindset of business processes than traditional Enterprise Application Integration (EAI) concepts that follow a more technical approach. In SOA the question is not so much how to structure a single software system, but how to structure the complete enterprise and its supporting information systems. To minimize interdependencies among modules it is important to take into account the principle of *'maximal cohesion and minimal coupling'* (Parnas, 1972). This leads to the question *how* we can conform to this maximum cohesion and minimal coupling principle. And are any other criteria important for module identification? Moreover, are these criteria only software engineering principles or maybe (also) organizational criteria?

Albani and Dietz (2006) combine the ideas of Enterprise Ontology, modularity and SOA by proposing Business Component Identification 3D (BCI-3D), a method for identifying business components, i.e. coarse-grained modules of a service-oriented system, and the services required for interaction between business components. In this dissertation we build on this foundation.

After an enterprise has identified the services it wants to offer to the market and the services for interaction between its coarse-grained modules, the enterprise faces another question, i.e. “How can a service be described in such a way that one is able to find a service and to know what the service does?”. A service catalog is in fact quite similar to the idea proposed by McIlroy (1968) (the catalog to search for software components). The specification of a service has three different goals. First, the specification acts as requirements to which the design of the internals must conform. Secondly, the specification enables potential service consumers to find the services they need by using aspects from the service as search items. Thirdly, the service consumer can use the specification to check whether or not the service will act according to the behavior he expects.

Chapter 2

Research Questions and Approach

In this chapter we elaborate on how we achieve our results. We discuss the research objective and the research questions in section 2.1. Section 2.2 presents the research approach. Section 2.3 provides the outline of this dissertation.

2.1 Research Objective and Questions

The objective we want to achieve in this dissertation is to provide practitioners with a better understanding of how to construct and specify service-oriented systems. As stated in the previous chapter, we define a service-oriented system as an enterprise, consisting of people and usually (but not necessarily) also IT systems, that offers services to its environment and that is structured in a modular way. In such an enterprise it does not matter whether a certain service is executed by a human being or by an IT system. Currently, most practitioners (usually enterprise architects) make decisions about modularization based on gut feeling and experience. This dissertation shows how coarse-grained modules of a service-oriented system can be delimited in an objective way. Next to this, it presents a service specification framework that prescribes the aspects that need to be specified for getting an understanding of the external behavior of a service. The service specification framework can be used for specifying the services delivered to the environment as well for the services for interaction among the modules of the service-oriented system.
The notions of *Enterprise Ontology* and *Enterprise Architecture* provide a firm theoretical basis for reaching our objective. The first notion encom-

passes that the organization of an enterprise is the layered integration of three aspect organizations: the B-organization, the I-organization, and the D-organization (Dietz, 2006b). By the realization of an enterprise is understood the thorough integration of these aspect organizations (Dietz, 2006b). By implementation is understood the making operational of the organization's realization by means of technology (Dietz, 2006b). Architecture is defined as the normative restriction of design freedom (Dietz, 2008) as explained earlier in chapter 1.
We start out with the basic understanding of a service that it has two key properties: (i) a service is always offered by a provider to a consumer and (ii) it hides the internals of a system from the consumer. We wish to clarify this service concept further using the notions of Enterprise Ontology and Enterprise Architecture.

This leads to our central research question: *how can service-oriented systems be constructed and specified in practice by founding service-orientation on the notions of Enterprise Ontology and Enterprise Architecture?* In order to answer this central research question we have formulated the following subquestions:

- RQ1: How can service-orientation be founded on the notions of Enterprise Ontology and Enterprise Architecture?
  (a) How can the Ψ-theory which underlies the notion of Enterprise Ontology be used to define the concept of service?
  (b) How can the notions of Service-Oriented Design (SoD) and Service-Oriented Architecture (SOA) be defined on the basis of the Generic System Development Process (GSDP)?
- RQ2: How can coarse-grained modules of service-oriented systems be delimited based on notions of Enterprise Ontology and Enterprise Architecture and on the needs of practitioners?
  (a) How can coarse-grained modules of a service-oriented system be delimited by applying the principle of maximum cohesion and minimal coupling on its ontological model?
  (b) What criteria do practitioners regard as important for delimiting coarse-grained modules of service-oriented systems?

   (c) How can the criteria proposed by practitioners be included in the proposed approach for delimiting coarse-grained modules of service-oriented systems?

- RQ3: How can the external behavior of a service be specified based on the Ψ-theory and on the needs of practitioners?

   (a) How can a service specification framework be derived from the Ψ-theory?

   (b) Which aspects of the service specification framework derived from the Ψ-theory do practitioners specify in their own organization and/or regard as useful for getting an understanding of the external behavior of a service?

   (c) What aspects is the service specification framework derived from the Ψ-theory lacking according to practitioners?

In RQ2 we choose BCI-3D as a starting point. The reason behind this choice is that BCI-3D is the only modularization approach (that we know of) that uses the Enterprise Ontology as a basis.

2.2 Research Approach

The goal of *Information Systems* (IS) research is to produce knowledge that enables the application of information technology for managerial and organizational purposes (Hevner and March, 2003). Looking at the IS research field, we see a distinction between (explanatory) *behavioral science* research ('problem understanding paradigm') and *design science* research ('problem solving paradigm') (Becker et al., 2008). Figure 2.1 shows the process model of the *Design Science Research Methodology* (DSRM) of Peffers et al. (2007). The DSRM is based on seven papers about design science research including the paper describing the most widely accepted framework for design science research proposed by Hevner et al. (2004). The process model shows the steps involved in design science research, viz. identify problem and motivate, define objectives of a solution, design and development, demonstration, evaluation, and communication. Also, the figure shows that we have four possible entry points: problem-centered initiation, objective-centered solution, design and development centered initiation, and client/context initiated.

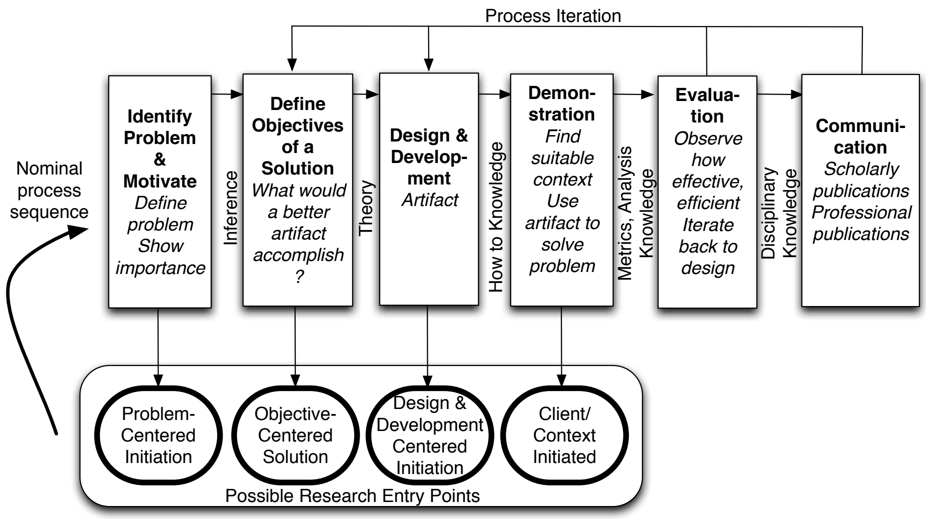

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of a Solution
What would a better artifact accomplish ?
Theory
Design & Development
Artifact
How to Knowledge
Demonstration
Find suitable context
Use artifact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points

Figure 2.1: DSRM process model Peffers et al. (2007)

For the design of the service specification framework we take a problem-centered initiation as our research entry point and we follow the nominal sequence. Our theoretical basis for designing the service specification framework is the Ψ-theory. In the evaluation step we verify in case studies whether practitioners agree that the service specification framework is complete and that it does not contain irrelevant aspects. In the case studies we use semi-structured interviews, because written questionnaires about complex matters are often not filled in very thoughtfully ('just get it done with'). Also, we do not want to structure the interviews completely to give the interviewees the chance to elaborate on what they think is important. A last step in conducting the case studies is to organize a workshop. During this workshop the interviewees can respond to each other's ideas. Multiple ways of organizing a workshop are available. One way to prevent that certain people dominate the workshop by talking too much and to enable people to express controversial ideas by guaranteeing their anonymity is the use of Group Decision Software. TeamSupport, a supplier of Group Decision Software, is willing to support our research by providing their software for free. Because this software provides a structured way to assemble data and to vote on the importance of things, this software can assist us in achieving our workshop goals.

We do not only use our case studies for evaluating the service specification framework, but also for deriving criteria for modularization from practice. Our goal is to get an overview of those criteria that are important for the identification of coarse-grained modules of service-oriented systems. Though we use BCI-3D as a starting point, our goal is not to validate BCI-3D as an identification method. Instead, we try to discover criteria that can be used as an input for BCI-3D (for determining the weights). So we do not only use our case studies for the evaluation step in the design science research methodology; they are also exploratory case studies (Yin, 2002). While many articles have been written about combining design science research with case study research (e.g. by Becker et al. (2008) and Pries-Heje et al. (2008)), no final broadly accepted answer on how to achieve this is available. We choose to use the guidelines of Hevner et al. (2004) to provide an overview of our research (see Figure 2.2).

All in all, our research aims at producing two design artifacts, viz. (i) criteria to delimit coarse-grained modules of a service-oriented system and (ii) a service specification framework that prescribes the aspects that need to be specified for getting an understanding of the external behavior of a service. We derive our service specification framework from theory and evaluate it in practice (deductive approach). The criteria for modularization are derived from practice (inductive approach).

1 *Design as an Artifact:* Our artifacts are criteria for the modularization of service-oriented systems and a service specification framework.

2 *Problem Relevance:* First, enterprises are better manageable when they are structured in modules. However, currently we lack insight into the criteria that need to be applied for this modularization. Second, often only the interface (in terms of input, output and errors) of the services used for interaction between modules is specified. This can lead to serious misinterpretations on what the service actually does.

3 *Design Evaluation:* We used an observational design evaluation method for evaluating the service specification framework, i.e. case studies.

4 *Research Contributions:* The contributions of this research are a theoretical foundation of service-orientation on the notions of Enterprise Architecture and Enterprise Ontology, criteria for the modularization of service-oriented systems derived from practice, and a standardized way of specifying services based on theory and validated in practice.

5 *Research Rigor:* We used the GSDP (Dietz, 2008) and the Ψ-theory (Dietz, 2006b) as a theoretical basis. Both theories are published in multiple academic papers.

6 *Design as Search Process:* We performed three case studies in which we aimed at acquiring new insights.

7 *Communication:* The artifacts have been presented to six large organizations. Scientific and professional articles are published.

Figure 2.2: Applying the guidelines of Hevner et al. (2004)

2.3 Outline

Figure 2.3 represents the structure of this dissertation graphically. Background information about Enterprise Engineering, modularization, and SOA was provided in chapter 1. This chapter (chapter 2) explained our research questions and approach. After this introductory part, three more parts follow: Theoretical Foundations, Theory Meets Practice, and Conclusions. The first chapter of the theoretical foundations, chapter 3, provides a literature overview of the notion of modularity and of different types of modularity. This chapter only contains existing theory. In chapters 4 to 7 we present new theory. First of all, the notion of 'service' is defined in chapter 4 (answer to RQ1.a). Chapter 5 explains the GSDP and founds the notions of SoD and Service-Oriented Architecture (SOA) on the GSDP (answer to RQ1.b). Also, it explains how BCI-3D can be applied to delimit coarse-grained modules of service-oriented systems (answer to RQ2.a and RQ2.c). We further explain our terminology by comparing several approaches to service-orientation in chapter 6. After this, chapter 7 elaborates on how we derive the service specification framework from the Ψ-theory (answer to RQ3.a). The next part of this dissertation presents the practical part of the PhD research. We elaborate on the results of our case studies in chapters 8, 9, and 10 (answer to RQ2.b, RQ3.b, and RQ3.c). We conclude this dissertation by reflecting on the case studies in chapter 11 and by answering the research questions in chapter 12.

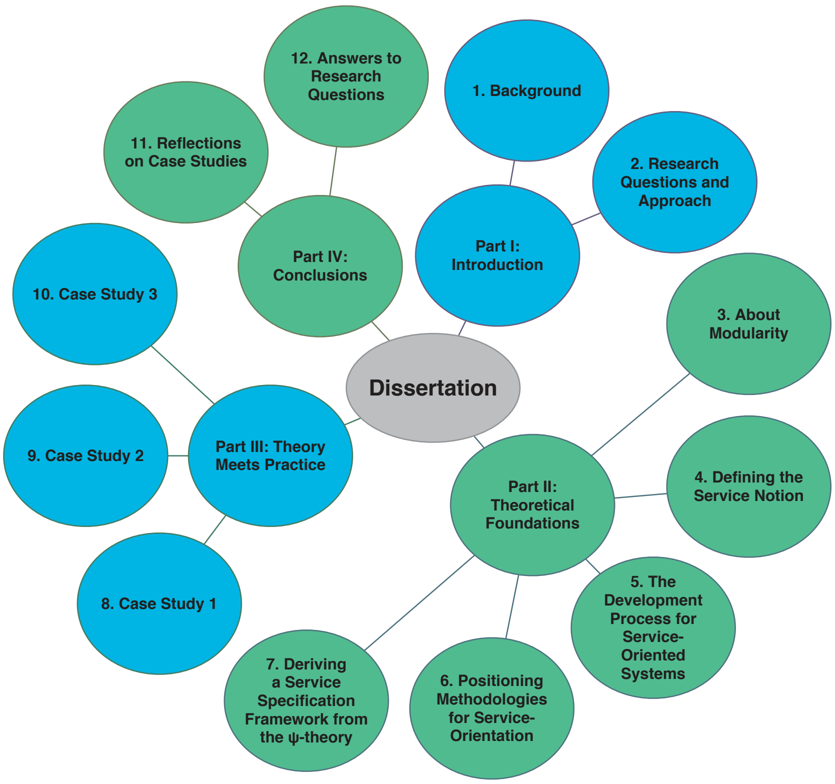

12. Answers to Research Questions
1. Background
11. Reflections on Case Studies
2. Research Questions and Approach
Part IV: Conclusions
Part I: Introduction
10. Case Study 3
3. About Modularity
Dissertation
9. Case Study 2
Part III: Theory Meets Practice
Part II: Theoretical Foundations
4. Defining the Service Notion
8. Case Study 1
5. The Development Process for Service-Oriented Systems
7. Deriving a Service Specification Framework from the ψ-theory
6. Positioning Methodologies for Service-Orientation

Figure 2.3: Structure of this dissertation

Part II

Theoretical Foundations

Chapter 3

About Modularity

Abstract Modularity is an important concept in system design. This chapter presents a formal definition of the notion of subsystem and shows that a module is subsystem with strong cohesion. Next, it explains the advantages and disadvantages of modular systems. As a service-oriented system is an enterprise (consisting of human services as well as IT services), we did not only look into literature from the field of computer science, but also into literature from the organizational sciences. From this we have learned that the organizational sciences distinguish between product modularity, organizational modularity, process modularity, and knowledge modularity (knowledge modularity is encountered less often in literature than the other types of modularity and the term is not well-defined).

3.1 Introduction

Modularity is an important notion in the field of general systems theory. Two of the most early, influential works on this subject are those of Alexander (1964) and Simon (1969). Although they do not use the word modularity, they both speak about dealing with complex systems by decomposing them into smaller parts. They do not limit themselves to a particular field. Instead, they discuss a broad range of system types that have a modular structure, e.g. biological systems, buildings, and enterprises.

In the same period McIlroy (1968) and Parnas (1972) proposed modularization as a means for dealing with complexity in software systems. Parnas (1972) introduced the criterion of information hiding, i.e. each module hides the manifestation of certain design decisions from the other modules. Interfaces between modules are chosen to reveal as little as possible about their internal operations. Also, he mentions some advisable specific example decompositions. McIlroy (1968) envisioned a mature software industry in which a purchaser can consult a catalog to search for software components that he can regard as black boxes: service-orientation avant la lettre.

One of the main premises of service-orientation is organizational flexibility. By organizational flexibility we refer to flexibility of the construction of the enterprise. In practice we often see the terms 'business flexibility' and 'business agility' (Bloomberg, 2003; Marks and Bell, 2006) used more or less as synonyms. As should be clear, we cannot limit ourselves to literature from the software engineering field to draw conclusions about organizational flexibility. That is why we also include literature from the organizational sciences.

The remainder of this chapter is structured as follows. In section 3.2 we define the notion of modularity. In section 3.3 we study the reasons that drive towards modular systems, and, equally important, those that prevent one from designing systems in a modular way. After that, in section 3.4, we have a look at different types of module cohesion and coupling identified in the field of software engineering. To see how modularity and business flexibility are related we analyze different types of modularity from the organizational studies in section 3.5. The conclusions are contained in section 3.6.

3.2 Defining Modularity

Ulrich (1995) defines a modular product design as one that "includes a one-to-one mapping from functional elements in the function structure to the physical components of the product and specified de-coupled interfaces between components". Watson and Pollack (2005), who build on the work of Simon and focus on evolutionary biology, disagree with this definition. They emphasize the difference between the structural modularity and the functional behavior of the system. Baldwin and Clark (2000) also state that modularity has to do with relationships among structures, and not functions and therefore do not adopt the definition of Ulrich. We agree with the last two authors and believe it is important not to intertwine functional and structural aspects of a system. We refer to the first as the teleological system notion and to the second as the ontological system notion (Dietz, 2006b). Baldwin and Clark (2000) provide us with a better definition, which they adapted from McClelland and Rumelhart (1995). They say a module is "a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units" (they use powerfully as a synonym for strongly). They add to this definition that there clearly are degrees of connection, thus there are gradations of modularity. Let us try to rephrase this definition in a more formal way.

Definition 1 shows us the definition of the construction of a system provided by Dietz (2006b) (following Bunge (1979)). The following two special symbols are used:

- $\prec$ means "is part of":
- $\rhd$ means "acts upon"; x acts upon y if and only if x influences the behavior of y; if both x $\rhd$ y and y $\rhd$ x hold, we say that x and y interact.

Definition 1. *Let σ be a* **system** *and Γ a class of things, called the* **category** *of σ. Then, the* **composition** *C of σ is defined as*
$C(\sigma) = \{ x \in \Gamma \mid x \prec \sigma \}$
the **environment** *E of σ is defined as*
$E(\sigma) = \{ x \in \Gamma \mid x \notin C(\sigma) \wedge \exists y\colon y \in C(\sigma) \wedge (x \rhd y \vee y \rhd x)\}$*, and*
the **structure** *S of σ is defined as*
$S(\sigma) = \{ < x,y > \mid (x \rhd y \vee y \rhd x) \wedge (x,y \in C(\sigma) \vee (x \in C(\sigma) \wedge y \in E(\sigma)))\}$
The composition, environment, and the structure are collectively called the **construction** *of the system. Thus the construction consists of the elements in the composition and the environment, as well as the relationships in the structure. The construction is a* **connected graph***, i.e. for every element (of the composition or environment) there is a walk to any other element.*

Dietz also provides us with a formal definition of a subsystem, i.e.:

Definition 2. *Let there be a system σ_1 with the construction $< C(\sigma_1), E(\sigma_1), S(\sigma_1) >$ and a system σ_2 with the construction $< C(\sigma_2), E(\sigma_2), S(\sigma_2) >$. Then system σ_2 is a* **subsystem** *of σ_1, if and only if*

- $C(\sigma_2) \subseteqq C(\sigma_1)$
- $E(\sigma_2) \subseteqq (C(\sigma_1) \setminus C(\sigma_2) \cup E(\sigma_1))$
- $S(\sigma_2) \subseteqq S(\sigma_1)$

Now let us revisit the definition of Baldwin and Clark. They state that a module is a unit whose structural elements are strongly connected among themselves and relatively weakly connected to elements in other units. We define a module (unit) of a system σ_1 as a subsystem σ_2 of this system. When we try to formalize their definition we face a problem, because 'strongly' and 'weakly' are not absolute criteria. For instance, we can define a strong connection as the condition that every element in a module interacts with more elements in the module than outside the module during a certain period. Or, more formally:

> Let σ_1 be a system and let σ_2 be a subsystem of σ_1. Let N(x,y,τ) be the number of times element $x \in$ C(σ_1) influences the behavior of element $y \in$ C(σ_1) or vice versa during a period τ and let sum(L) be a function for calculating the sum of the numbers in list L.
>
> σ_2 has strong cohesion during period τ if:
> $\forall\ x \in \mathrm{C}(\sigma_2) : (\mathrm{sum}\{\mathrm{N}(x,y,\tau) \mid y \in \mathrm{C}(\sigma_2) \land x \neq y\ \} > \mathrm{sum}\{\mathrm{N}(x,z,\tau)$
> $\mid z \in (\mathrm{C}(\sigma_1 \setminus \mathrm{C}(\sigma_2)\}\ )$

But we could also define it in a weaker way, e.g. that the total number of acts upon relations in period τ among elements in the module is higher than the number of connections between elements in the module and elements in the environment of the module during τ, viz.:

> σ_2 has strong cohesion during period τ if:
> $\mathrm{sum}\{\mathrm{N}(x,y,\tau) \mid x \in \mathrm{C}(\sigma_2) \land y \in \mathrm{C}(\sigma_2) \land x \neq y\ \} > \mathrm{sum}\{\mathrm{N}(x,z,\tau) \mid$
> $x \in \mathrm{C}(\sigma_2) \land \mathrm{y} \in (\mathrm{C}(\sigma_1 \setminus \mathrm{C}(\sigma_2)\}$

We see that it is not possible to create hard criteria on the degree of modularity from this definition.

Next, we face a second problem. Not only the number of connections are relevant, but also the nature of the connections. When two modules in a system have a relation, this relation itself can be stronger or weaker. The acts upon relation can lead to very small or very large changes in the behavior of another module. All in all, we lack the general criteria to determine whether or not a system is structured in 'the best modular way'. Nevertheless, we can define a module as a subsystem that has high cohesion and low coupling with other subsystems of the same system. But for getting a better understanding we need to dive further into literature on specific system types. In our study on modularity of service-oriented systems we are interested in modularity of

IT systems as well as modularity of enterprises. We give a literature overview on the first subject in section 3.4 and on the second in section 3.5. But first, we have a look at what why one should or should not choose for a modular design in section 3.3.

3.3 Reasons for (not) Applying Modularity

Many researchers, e.g. Parnas (1972), Baldwin and Clark (2000), Sanchez and Mahoney (1996), bring up one or more of the following advantages of modularization: (i) making the total system structure more comprehensible, (ii) enabling easy replacement of components, (iii) making it possible to divide work between several groups of designers without them needing to be aware of the structure of the total system, and (iv) minimizing the effect that changes in one part of the system have on the other parts of the system. However, a great degree of modularity does not only pose benefits in system design. Schilling (2000) presents a general theory of modular systems from a market perspective. She draws on systems theory from many disciplines. In this theory she answers the question of what drives some systems toward increasing modularity and others toward increasing integration. According to her, the balance between the gains achievable through recombination (of modules) and the gains achievable through specificity (of the system as a whole) determines the pressure for or against the decomposition of a system (Schilling, 2000).

Schilling defines different forces that drive toward or away from modularity. First, the *heterogeneity of inputs* drives towards modular systems. With input the author does not refer to the input of an interface, as computer scientists usually do. But she refers to the number of components (modules), that are available to compose a system. The more heterogeneous the inputs are that may be used to compose a system, the more possible configurations are attainable through recombinability enabled by modularity (Schilling, 2000). When many components of a certain type are available vendor lock-in can easily be prevented. Another positive force for modularity is *heterogeneity of demands*. This means when customers of a certain product have very different needs, then it is difficult to find a 'one size fits all' integrated solution. The forces of heterogeneity of inputs and heterogeneity of demands reinforce each other. A factor that negatively impacts the choice for modularity is *synergistic specificity*. She states that if customers need very specific components, the costs of making specialized interfaces would be very high. Also, the degree of difficulty customers face in assessing the quality and interaction of components and in assembling the components will be negatively related to increasing (interfirm) product modularity. Schilling (2000) mentions technological change and competitive intensity as some of the primary factors that create urgency. In both cases suppliers need to be able to rapidly reconfigure their products. So urgency directly contributes to a need for modularity. Also, urgency reinforces the heterogeneity of inputs and the heterogeneity of demands.

When we apply this framework to an information system as a product, we can easily explain the popularity of the modular service-orientation approach. For this type of product, the end user organization of the enterprise is the customer. The supplier(s) can be external companies and/or the internal IT department(s). We see a heterogeneity of inputs in the many suppliers of different parts of an information system, e.g. for CRM systems we have different solutions like Siebel, SAP CRM, Microsoft CRM, and custom developed solutions. Also, we see a heterogeneity of demands. There are no 'one size fits all' information systems for (large) enterprises. Even ERP systems, of which the vendors make such a statement, require a lot of tweaking and tuning. Often even so much that these ERP systems end up more like custom developed systems, than COTS systems. We recognize urgency as a driver too. Technology evolves fast in the IT world and organizations want to extend or replace parts of their information system with subsystems written in new programming languages and/or based on new software paradigms. IT suppliers feel a competitive intensity, because their clients demand more business flexibility. They do not have any problem with moving to another supplier or to outsource their IT department. All in all, we see many forces that drive toward a modular approach in information system design. However, we also recognize the negative forces of synergistic specificity. Sometimes organizations demand very specific information systems. Usually, this results in custom made software development. Assessing the quality of different subsystems can be hard. At the moment we see many organizations struggling in their SOA projects to make everything 'fit together'.

Arnheiter and Harren (2006) add some other negative aspects of modularity to the list: the limitation of creativity of design because of the need for well-defined interfaces, the overuse of the same module across too many product lines, less than optimal system performance due to use of generic modules, unnecessary time and expenses of replacing an entire module when only a single component within the module is faulty. We do not agree with the last statement as being a disadvantage of modularity, since in our eyes this ‘component’, i.e. submodule, is just a module at a lower level and there is no reason why submodules cannot be available on the market.

Concluding, we see that modularity has advantages as well as disadvantages. Table 3.1 exhibits the main advantages and disadvantages of modular systems as found in literature. The advantages are that the total structure is more comprehensible: modularity contributes to making a system intellectually manageable. Because interfaces of modules are specified, modules can be easily replaced by other ones. Modularity also contributes to the process of building/changing a system. People working on one part of the system do not have an overview of the complete system. When making changes to the system one knows what changes will and will not affect other parts as the system, as only changes of a module that affect the interface can affect other parts. Because often several different implementations of a module conforming to a certain interface are available, one can choose between different configurations of a system. This standardization also prevents vendor lock-in. Disadvantages are the higher costs involved in making a modular system in comparison to making a non-modular one. Next to this, the task of integrating the different modules can be complex. One has to assess the quality and interaction of different modules and letting everything work together can be hard. Designers can be limited in their creativity as they have to conform to the interface and it can lead to less variation in products as all products use the same modules. Finally, the total system performance may be suboptimal.

3.4 Modularity of IT Systems

In this section we describe the research on modularity performed in the field of software engineering. We start by looking at imperative systems in subsection 3.4.1. Then we move to object-oriented systems in subsection 3.4.2. Lastly, we look at service-oriented systems in subsection 3.4.1.

3.4.1 Imperative Systems

In the field of software engineering McIlroy (1968) and Parnas (1972) were the first authors to propose modularization as a means for dealing with complexity. Several years later Myers (1978) introduced the concept of *clustering.* As depicted in Figure 3.1 he plots the coherence of program statements of an

Advantages
Total structure is more comprehensible (Sanchez and Mahoney, 1996; Parnas, 1972; Baldwin and Clark, 2000; Arnheiter and Harren, 2006).
Modules can be easily replaced (Sanchez and Mahoney, 1996; Parnas, 1972; Baldwin and Clark, 2000; Arnheiter and Harren, 2006).
Work division possible without everyone having overview of complete system (Parnas, 1972; Baldwin and Clark, 2000; Arnheiter and Harren, 2006).
Effect of changes of one part of system to other parts are minimized (Sanchez and Mahoney, 1996; Parnas, 1972; Baldwin and Clark, 2000; Arnheiter and Harren, 2006).
Many different configurations of the system are possible (Schilling, 2000; Sanchez and Mahoney, 1996; Baldwin and Clark, 2000; Arnheiter and Harren, 2006).
Vendor lock-in is prevented due to standardization (Schilling, 2000).
Disadvantages
For very specific modules the cost of making interfaces can be high (Schilling, 2000).
For assemblers (integrators) it can be difficult to assess the quality and interaction of different modules (Schilling, 2000; Arnheiter and Harren, 2006).
It can be difficult to assemble (integrate) the modules (Schilling, 2000; Arnheiter and Harren, 2006).
The design creativity of a module designer can be limited because he needs to conform to the interface (Arnheiter and Harren, 2006).
Less variation in products because of overuse of the same modules (Arnheiter and Harren, 2006).
Total system performance may be suboptimal (Arnheiter and Harren, 2006).

Table 3.1: Advantages and disadvantages of modular systems

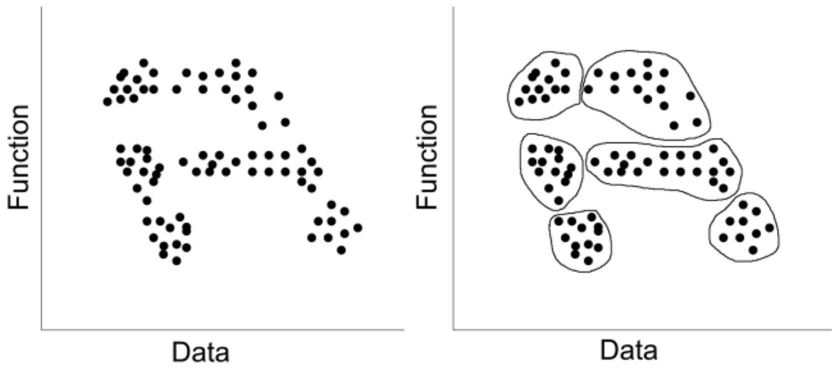


Figure 3.1: The ideal module boundaries according to Myers (redrawn version of figure by Myers (1978))

existing program based on their functional relationship (i.e. are they part of the same function or different ones) and on their data relationship (i.e. do they reference the same variables, different fields in the same structure or completely unrelated data). When these points representing the statements are plotted in a graph, according to Meyers one can define the ideal module boundaries by a clustering approach. Myers proposed the methodology of composite design as a means to get foresight on how to define these clusters.

Myers defines different categories of module strength, i.e. the cohesion of a module. These categories are, from highest to lowest strength: informational strength (equal level as functional strength), functional strength (equal level as informational strength), communicational strength, procedural strength, classical strength, logical strength, and coincidental strength. Myers stresses that the scale does not imply that a program with a lower-type of strength than functional strength is undesirable. He does, however, say that this classification provides a designer with a means to identify the type of module strength. It allows him to make a tradeoff between module strength and other considerations.

Yourdon and Constantine (1979), two other researchers from the structured design paradigm, define 'initialization' and 'termination' modules as another type of cohesion: *temporal cohesion.* This type of cohesion entails that statements that are always executed at the same time are grouped in a module. Besides this, they introduce *sequential cohesion.* In this type of module output data from one function serves as input data for the next one.

3.4.2 Object-Oriented Systems

Later, as the paradigm of object-orientation gained territory, several researchers continued the work of Myers, Yourdon and Constantine. Briand et al. (1998) present a literature survey on different studies on cohesion for object-oriented systems. They compared six different frameworks: the frameworks by Eder et al. (1994), Chidamber and Kemerer (1991, 1994), Hitz and Montazeri (1995), Bieman and Kang (1995), Henderson-Sellers (1996), Lee et al. (1995), Briand et al. (1993), and Briand et al. (1994). Eder et al. distinguish between three types of cohesion in an object-oriented system: method, class and inheritance cohesion. For method cohesion Eder et al. define the same degrees of cohesion as Myers. Also, they define degrees of cohesion specific to classes. An example of the application of one of their principles is that a class Employee having the attributes DayOfBirth, MonthOfBirth, YearOfBirth, DayOfHire, MonthOfHire and YearOfHire would be better off with making the concealed data abstraction 'date' a separate 'Date' class.

Chimdamber and Kemerer base their work on that of Bunge (1979), which we already used in this chapter to define the notion of system. They define the cohesion of a class as the degree of similarity of its methods, i.e. the number of attributes used in common by all methods. The approaches of Hitz and Montazeri and of Bieman and Kang are based on that of Chidamber and Kemerer.

Henderson-Sellers defines cohesion measures based on how many attributes of a class each method within the class references as attributes.

Lee proposes a set of cohesion measures based on information flow through method invocations within a class. This means that for a method in a class, its cohesion is the number of invocations to other methods implemented in this class, weighted by the number of parameters of the invoked methods.

Briand et al. measure cohesion based on the interactions of data structures, which can be either attributes within the class or types of a different class. By interaction they mean that a change in one data declaration may cause a change in another one. Next to this, they measure data/method interaction; such an interaction exists if a data interaction interacts with at least one data declaration of the method.

3.4.3 Service-Oriented Systems

Now let us move on to the more recent field of SOA in which the notions of cohesion and coupling play an important role. Table 3.2 exhibits an overview of tight versus loose coupling different levels of a distributed software system according to Krafzig et al. (2004). This work does not so much give us insight into how to define services, but it shows how IT systems that use each other's services can be as independent from each other as possible. First, service consumer and provider need to be decoupled at a physical level, usually message queues are used for this purpose. Next, to achieve loose coupling communication between consumer and provider should be asynchronous, i.e. they must be able to communicate while not being available at the same time. In a strong type system input and output arguments of the services (or functions) are directly coupled and have a compile-time dependency. In weak type system the services receive input and produce output in messages. The structure of these messages can be changed without having to recompile the services. Of course one still must know how the changes in the message affect the services. But one has more flexibility in changing messages (e.g. optional data structures) that are used by multiple services. On the interaction pattern level there is a difference between tight object-oriented style navigation and data-centric, self-contained messages. By the first style the authors mean that a consumer has to understand not only the logic of each individual object, but also the way to navigate across objects. In message-based systems navigation is much simpler. Next, in tightly coupled systems (e.g. monolithic ERP systems) process logic is centralized, i.e. all processes, subprocesses, and transactions are stored in one location. In loosely coupled systems this can be distributed. Finally, in really loosely coupled systems binding of a service should be dynamical. Finally, of course, services should not have any dependencies on the operating system or programming language.

3.5 Modularity from Enterprise Perspective

The phrase 'loose coupling' that we encounter so often in the field of service-orientation was introduced in the organizational sciences by Weick (1976). Weick read about this loose coupling in works of Glassman (1973) and March and Olsen (1975) and defines it as "the image that coupled events are responsive, but that each event also preserves its own identity and some evidence of its physical or logical separateness". He applies this notion to the context of

Level	Tight coupling	Loose coupling
Physical coupling	Direct physical link required	Physical intermediary
Communication style	Synchronous	Asynchronous
Type system	Strong type system (e.g., interface semantics)	Weak type systems (e.g. payload semantics)
Interaction pattern	OO-style of navigation of complex object trees	Data-centric, self-contained messages
Control of process logic	Central control of processing logic	Distributed logic components
Service discovery and binding	Statically bound services	Dynamically bound services
Platform dependencies	Strong OS and programming language dependencies	OS and programming language independent

Table 3.2: Tight versus loose coupling in service-orientation

educational organizations. In the literature of the organizational sciences, we also find the notions of loosely coupled, autonomous, and reconfigurable in the work of other authors (Hoetker, 2002; Adamides et al., 2005; Sanchez and Mahoney, 1996; Brusoni and Prencipe, 2001). The concept of interface is mentioned by multiple authors, e.g. Brusoni and Prencipe (2001) and Salvador et al. (2002).

In literature from the organizational sciences a distinction is made between different kinds of modularity:

- product modularity
- organizational modularity
- process modularity
- knowledge modularity

Product modularity refers to the type of modularity we most often speak of. Arnheiter and Harren (2005) distinguish between *hard* and *soft* modules, based on the work of O'Grady (1999). Hard modules have a physical appearance, whereas soft modules have a limited physical presence. Arnheiter and

Harren mention software, financial products or insurance policies as examples of soft modules. They define a modularity typology containing four different types that holds for hard as well as soft modules:

1. manufacturing modularity
   In this type of modularity suppliers are able to produce products by using only a handful of pre-manufactured subassemblies (modules).

2. product use modularity
   In this type of modularity the user is able to use modules for product customization.

3. limited life modularity
   In this type of modularity a distinction between modules is made based on their life span. Modules that have a limited life time (e.g. batteries) should be easily replaceable.

4. data access modularity
   In this type of modularity the data storage is realized in data access modules, e.g. CDs, USB sticks etc. These data access modules can be used in different system types, e.g. a PC or a camera.

Todorova and Durisin (2008) distinguish between *internal* and *external* product modularity depending on whether design and integration activities take place within the same organization or whether they are all distributed in a network of different organizations. Other authors call this external product modularity *interfirm modularity* (Baldwin and Clark, 2000; Schilling, 2000; Langlois and Robertson, 1992). Ulrich (1995) defines three different types of product modularity: slot, bus, and sectional. Each of the interfaces between components in a slot architecture is of a different type than the others, so that the various components in the product cannot be interchanged. An automobile radio is an example of a component in a slot architecture. The radio implements exactly one function and is de-coupled from surrounding components, but its interface is different from any of the other components in the vehicle (e.g. radios and speedometers have different types of interfaces to the instrument panel). In a bus structure the connections are the same for each component. In a sectional structure component can be added or removed freely, in a Lego-like manner.

Organizational modularity deals with modularity in the organizational structure. In modular organizational structures, small component development and manufacturing units or business units can function autonomously and innovate concurrently under the coordination of the product blueprint (Todorova and Durisin, 2008). Organizational modularity can be applied to one company as well as to an organizational network or supply chain. Topics in this area include the organization of project teams, outsourcing, alliances etc.

Process modularity (Kusiak, 2002; Bask et al., 2009; Ding and Jie, 2008) refers to breaking down a business process into standardized subprocesses (i.e. process modules). So process modularity is about standardizing process to enable the rearrangement of different process configurations (Todorova and Durisin, 2008). These subprocesses may be performed at different locations and by different companies. This type of modularity can, for instance, be applied to assemble the complete process at the last moment (Just in Time) to enable very efficient production. Usually, process modularity is accompanied by product modularity.

Though we encountered the term *knowledge modularity* several times (e.g. Brusoni and Prencipe (2001) and Bohn (2005)), we could not find a good definition. It somehow deals with knowledge structures of enterprises, i.e. who has what knowledge about the product and its components. But we have some difficulties in seeing this as a separate type of modularity, since it is very closely related to process modularity; to perform certain activities one has to have certain knowledge.

Sanchez and Mahoney (1996) state that "standardized component interfaces in a modular product architecture provide a form of embedded coordination that greatly reduces the need for overt exercise of managerial authority to achieve coordination of development processes, thereby making possible the concurrent and autonomous development of components by *loosely coupled organization structures* (Orton and Weick, 1990)". They call these organization structures modular organization designs. Hoetker (2002) argues that product modularity may not lead firms to move activities from hierarchy to more loosely coupled organizations to the degree that the literature (including the article of Sanchez and Mohoney) has assumed. Brusoni and Prencipe (2001) study how product modularity, organizational modularity, and knowledge modularity correlate. Regarding knowledge modularity, they build on the work of (Arora and Gambardella, 1994), who contend not only that the production of new products can be conceived in terms of the production and combination of modules, but also that the knowledge underlying such products is a matter of 'mixing and matching' modules.

One of the main premises of SOA is to achieve organizational flexibility. However, many approaches to Service-Oriented Architecture (SOA) are very much focused on technology and only spend little attention on services executed by human beings. The main error in reasoning is that product modularity of a supporting product alone (the IT system) is sufficient to enable flexibility of the complete enterprise. Instead, it is only conditional. The latter means that business decisions, e.g. outsourcing, mergers and acquisitions, selling new product types, are no longer withheld by limitations of IT systems. Also, software engineering principles are not the only thing to consider when structuring information systems. Consider, for instance, the principle 'customer data must be stored only once'. This makes perfect sense from a software engineering perspective, because this is efficient and prevents data inconsistencies. However, there can be legal restrictions that prohibits the reuse of customer data from one department (e.g. pension management department) by another department (e.g. insurance selling department). Concluding, we can say that it is too simple to say that more flexible IT systems lead to more flexible enterprises.

3.6 Conclusions

In this chapter we studied the concept of modularity. The most widely accepted definition in general systems theory is that of Baldwin and Clark. They define a module as a unit whose structural elements are strongly connected among themselves and relatively weakly connected to elements in other units. Unfortunately, the terms 'strongly' and 'weakly' are open to interpretation. Therefore this definition only provides guidance instead of clear criteria to determine the degree of modularity of a system. In software engineering literature more precise classifications of modularity are proposed in the '70s by Myers and Constantine. Specifically for service-oriented systems Krafzig et al. give an overview of the difference between tight and loose coupling. In our research we want to find out how we can define coarse-grained modules of service-oriented systems to profit from the advantages of modularization. After studying literature about modularization from the organizational sciences we can draw two conclusions. Our conclusion is that modularization of IT systems is not sufficient to create more flexible enterprises. Many approaches on SOA only focus on IT systems. As we have seen in literature from the organizational sciences, this is insufficient for achieving business flexibility. In the remainder of this thesis we aim to get a better understanding of the principles that play a role in the structuring of service-oriented systems by looking at human as well as IT services.

Chapter 4

Defining the Service Notion

Abstract The contribution of this chapter is a precise and unambiguous definition of the notion of 'service'. We base this definition on the Ψ-theory. This theory originates from the scientific fields of Language Philosophy and Systemic Ontology and underlies the notion of Enterprise Ontology. According to this theory, the operation of organizations is all about communication between and production by social actors. The service definition presented in this chapter is based on the complete transaction pattern in the Ψ-theory. Though a service has many similarities with a transaction, they are not equal. While the transaction includes all acts of the initiator and the executor, the service concept only regards the executor side. We therefore define a service as a part of a transaction rather than a whole transaction. We can classify services based on two criteria. The first criterion is the type of production fact delivered. This production fact can be either ontological, infological, or datalogical. The second criterion is the type of implementation; a service can be implemented by a human being or an IT system. We therefore distinguish between the following six types of services: ontological human services, infological human services, datalogical human services, ontological IT services, infological IT services, and datalogical IT services.

4.1 Introduction

Before thinking about how we can specify services in a service specification framework, we need to define what a service is. Economists and business scientists have been debating about this *'service'* notion for more than two centuries (Gadrey, 2000). Often, the definitions in business literature limit the service notion to the delivery of immaterial goods. The adoption of the notion of 'service' by computer scientists and IT practitioners has been more recent. In both the business science field (Zeithaml et al., 1993; Gallouj and Weinstein, 1997; Hart, 1988; Goldstein et al., 2002) and the computer science field (OMG, 2006; OASIS, 2006a; The Open Group, 2006; W3C, 2006b) a service is regarded as an interaction between a requesting party (often called consumer or customer) and an offering party (often called provider or supplier). The offering party is able to produce a certain value that is requested by the other party. But even with this common notion a precise definition and mutual understanding of the term service is missing. Let us have a look at the definition given by the Open Group (The Open Group, 2006). It says that a service:

- is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data, consolidate drilling reports),
- is self-contained,
- may be composed of other services, and
- is a 'black-box' to consumers of the service.

This definition is as vague as the other definitions mentioned above and one could discuss every single statement of the definition. E.g., what is a business activity? The Open Group mentions 'check customer credit' or 'provide weather data' as business activities, but are these really business activities or are they only computational acts? What about a business activity concerning the 'manufacturing of a car'? Such a business activity has a completely different granularity as the ones mentioned in the definition. What is self-contained? If a service is composed of other services is it still self-contained? What is precisely meant by a black-box when a service is also defined to be an activity? What about communication activities e.g., to call the service or to accept/reject the requested result?

We start this chapter with further explaining the Ψ-theory in section 4.2. Subsequently, we provide our service definition and six different types of services in section 4.3. Section 4.4 provides the conclusions of this chapter.

4.2 The Ψ-theory

The Ψ-theory (Dietz, 2006b) finds its roots in the scientific fields of Language Philosophy, in particular the Language Action Perspective (LAP) (Flores and Ludlow, 1980; Goldkuhl and Lyytinen, 1982), and in Systemic Ontology (Bunge, 1979). It focuses on the use of language to achieve agreement and mutual understanding (Weigand, 2003). By applying the Ψ-theory one can disentangle the essential knowledge of the construction and the operation of the organization of an *enterprise*, by which we mean a commercial or non-profit company as well as a network of enterprises. This essential enterprise model is called the ontological model. The theory consists of several axioms and one theorem. In this section we give a short summary of the Ψ-theory. We only discuss the parts of the theory that we need for developing a service specification framework, viz.: the operation axiom, the transaction axiom, the distinction axiom, and the organization theorem. A complete overview of the theory is available in the book (Dietz, 2006b) and the papers (Dietz and Hoogervorst, 2008; Dietz and Albani, 2005; Dietz, 2006a; Dietz and Hoogervorst, 2007).

4.2.1 The Operation Axiom

The first axiom, the operation axiom, focuses on the different types of acts that actors in organizations (people, also called subjects) perform and the results of these acts. It states the following (Dietz, 2006b):

Axiom 1. *Actors perform two kinds of acts: production acts and coordination acts. These acts have definite results: production facts and coordination facts respectively. By performing production acts, actors contribute to bringing about the function of the organization. By performing coordination acts, actors enter into and comply with commitments regarding production acts. An actor is a subject fulfilling an actor role. Actor roles are elementary chunks of authority and responsibility.*

What are these so-called production and coordination acts the axiom speaks about? And why do we need to distinguish between them? First, let us look at the production acts. Production acts are acts that deal with the delivery of material or immaterial goods by actors to their environment. Their results are production facts. Examples of production acts dealing with material goods are manufacturing and transporting. Their corresponding production facts are 'Product P has been manufactured' and 'Product P has been transported'. For immaterial goods, examples are deciding and judging. Their corresponding production facts are 'Decision D has been made' and 'Judgment J has been made'. Coordination acts serve a totally different purpose than production acts, though they are executed by the same actors. They do not directly contribute to the production of goods, but they *coordinate* the execution of production acts. An example of a coordination act and its corresponding fact is the request for manufacturing a product and 'The production fact "Product P has been manufactured" has been requested'. In the next paragraph we will see that the different types of coordination acts form a limitative list.

4.2.2 The Transaction Axiom

The second axiom, the transaction axiom, further looks into the coordination acts. It states the following (Dietz, 2006b):

Axiom 2. *Coordination acts and production acts always occur in particular patterns. These patterns are paths through one universal pattern, called transaction. The result of carrying through a transaction is the creation of a production fact.*

A transaction evolves in three phases, the *order* phase (O-phase), the *execution* phase (E-phase) and the *result* phase (R-phase), see Figure 4.1. Two actor roles are involved in such a transaction, the *initiator*, who starts and completes the transaction, and the *executor*, who performs the production act. In the order phase the initiator and the executor try to reach agreement about the intended result of the transaction, i.e., the production fact that the executor is going to create as well as the intended time of creation. In the execution phase this product is created by the executor, and in the result phase both actors try to reach agreement about the fact that has been produced. The so-called *basic transaction pattern* consists of the request, promise, state, and accept coordination acts. An example of this basic pattern looks as follows.

1. person A *requests* person B to *manufacture a car*
2. person B *promises* person A to *manufacture a car*
3. <actual delivery of the manufactured car>

4. person B *states* to person A that he has *manufactured a car*

5. person A *accepts* from person B that he has *manufactured a car* (the car conforms to his expectations)

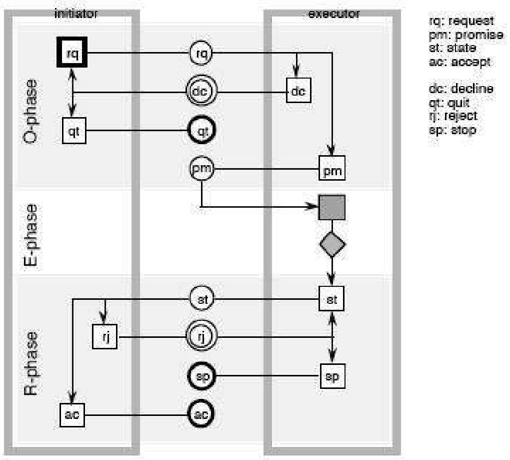


Figure 4.1: Standard transaction pattern

Transactions will conform to this basic transaction pattern in a *happy scenario*, i.e. everything goes as it should go. However, in reality the initiator and executor may dissent in two of the states; (i) the requested state and (ii) the stated state. In the first case, the executor may (instead of promising) respond to a request by declining it. In the second case, the initiator may (instead of accepting) respond to a statement by rejecting it. By allowing these acts, a transaction can end up in a *discussion state.* Dietz describes that in this situation the two actors must sit together, discuss the situation at hand, and negotiate about how to get out of it. When the basic pattern is expanded with these two dissent patterns, we get the *standard transaction pattern.* The standard transaction pattern is the pattern already introduced in Figure 4.1. The *complete transaction pattern* is constituted by the standard pattern and four *cancellation patterns.* Cancellation patterns concern the revocation of a request act, promise act, state act, or accept act.

4.2.3 The Distinction Axiom

The third axiom, the distinction axiom, is concerned with the different abilities of a human being that are involved in the activities they perform. The axiom states the following (Dietz, 2006b):

Axiom 3. *Three distinct human abilities play a role in the performance of coordination acts and production acts: the forma, informa and performa abilities.*

How are these human abilities relevant for coordination acts on the one hand and production acts on the other hand? The forma ability deals with the form aspects of communication and information. Applying this to coordination acts, this means actors should have a way to utter and perceive information. Information should be expressed in a particular language or code scheme that both the initiator and the executor of a transaction understand. This is also known as syntactic (or significational) understanding. One might think, for instance, of information written in English. The informa ability concerns the content aspects of information and communication. In order to communicate, the initiator should formulate information in a way that the executor can interpret. In other words, the initiator and the executor should semantically be in agreement with each other and share the same thoughts. This is also called intellectual understanding. The performa ability states that new information and knowledge can be created through communication between the initiator and executor. Looking at coordination acts, this means that actors can expose and evoke commitments and it indicates social understanding between the initiator and executor. For the production acts we see a similar distinction. The forma ability is concerned with the form aspects of information in terms of information transmission and storage. This type of production acts are known as *datalogical acts.* Transactions that contain a datalogical act are called *datalogical transactions* (D-transactions). The informa ability states that information can be reasoned, computed or deduced. Those activities are known as *infological acts.* Transactions are called *infological transactions* (I-transactions) if they include this type of production act. The performa ability concerns making decisions, judgments, or creating material things such as products. This is what we call *ontological acts.* Transactions that include ontological acts are known as *ontological transactions* (B-transactions).

4.2.4 The Organization Theorem

We just presented three of the axioms of the Ψ-theory. Together with the composition axiom, which we did not discuss, they provide the basis for the organization theorem. This theorem provides a concise, comprehensive, coherent, and consistent notion of enterprise, such that the (white-box) model of an enterprise may rightly be called its ontological model (Dietz, 2006b). It states the following (Dietz, 2006b):

Theorem 1. *The organization of an enterprise is the layered integration of three aspect organizations: the B-organization, the I-organization, and the D-organization.*

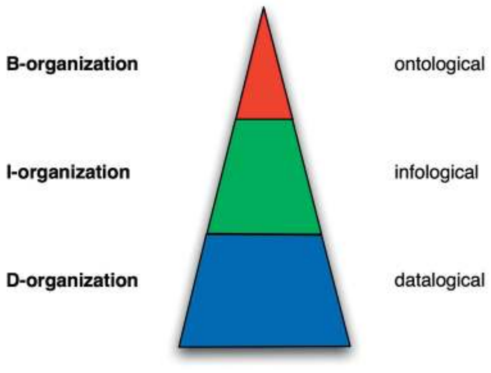


Figure 4.2: The three aspect organizations

Figure 4.2 shows the three aspect organizations. The *B-organization* concerns the essence of the enterprise. It consists of actors who directly contribute to the enterprise's goals and functions by performing ontological production acts. These actors are known as *B-actors* and are able to perform B-transactions, the ontological transactions we defined in the previous paragraph. B-actors are, for instance, consultants or sales persons. The *I-organization* embraces the content aspects of information and knowledge in the enterprise (Dietz, 2006b). Actors in the I-organization, who are called *I-actors*, bring changes to information and knowledge by performing infological production acts. In other words, I-actors perform I-transactions. Business controllers are typical actors in the I-organization producing infological things. The *D-organization* deals with the documentation of information in the enterprise and only takes into account the form of information. To achieve this, actors in the D-organization perform datalogical production acts and thus D-transactions. These actors are known as *D-actors*, who are for instance archivists.

4.3 Defining Service Based on the Ψ-theory

According to the Ψ-theory, the previous paragraphs have shown, the operation of organizations is all about communication between and production by social actors. Is not the main concern of service-orientation to support the operation of an organization and therefore also to support the communication between and production by social actors? Because the Ψ-theory describes the interaction between the requesting party and the offering party in a very formal way, it provides a basis for formalizing the notion of service. Based on this theory we have elaborated on the notion of service (Albani et al., 2009) and we will summarize in the next paragraphs the main results which are of relevance for the specification of services.

The definition of service is based on the complete transaction pattern. Though a service has many similarities with a transaction in the Ψ-theory, they are not equal. While the transaction includes all acts of the initiator and the executor, the service concept only regards the executor side. We therefore define a service as a part of a transaction rather than a whole transaction.

Definition 3. *A service is a pattern of coordination and production acts, performed by the executor of a transaction for the benefit of its initiator, in the order as stated in the complete, universal pattern of a transaction. When implemented it has the ability*

- *to get to know the coordination facts produced by the initiator and*
- *to make available to the initiator the coordination facts produced by itself.*

By universal we mean that this pattern applies to interaction between actors in all types of organizations (non-profit and commercial) and in all countries and cultures of the world.

When looking at the complete transaction pattern, everything except the coordination acts of the initiator (request, quit, reject and accept) are part of the service. But in order to communicate with the executor of the service, the initiator needs to be aware of the complete transaction pattern.

Additionally, we call a service a *composite service*, if the execution of the service requires the executor to initiate certain other services. This happens exactly then, when a transaction is enclosed in some other transaction.

This definition of a service just given is a very generic one, since it holds for two kinds of actors, *human actors* and *IT systems* and three kinds of production acts, namely *datalogical*, *infological* and *ontological*.

Services executed by human actors or IT systems only differ in the way they are implemented; human services are implemented by human beings, whereas IT services are implemented by IT systems. IT systems assist human actors in their activities. For both human actors and IT systems we can distinguish between communication acts and production acts on the datalogical, infological, and ontological level as described in the organization theorem 4.2.4 (though at ontological level machines can only mimic the behavior of the responsible human actors, because machines can never reach true social understanding and cannot create really new, original things). Examples of datalogical production acts are storing, copying, transmitting of documents or data. Acts such as reasoning, computing, deriving or reproducing knowledge are examples of infological production acts and the acts concerning the creation of original new things, such as creating material products or making judgments are examples of ontological production acts.

The basic concept of dealing with coordination and production aspects between an initiator and an executor party as defined in Ψ-theory and in the generic service definition given above allow us to distinguish between six different types of services:

- ontological human service
- infological human service
- datalogical human service
- ontological IT service
- infological IT service
- datalogical IT service

All service types conform to the definition of service given above, following the same service pattern and the described abilities. They only differ in the way they are implemented, either by human actors or by IT systems and in the different kinds of coordination and production acts as described above. In chapter 7 we will see that we can use the same specification aspects for specifying all six types of services, though the way of specification is different for human and IT services.

4.4 Conclusions

In this chapter we provided a definition of the notion of service and introduced six different types of services, based on the Ψ-theory: ontological human services, infological human services, datalogical human services, ontological IT services, infological IT services, and datalogical IT services. The first distinction is between human services, i.e. services executed by human beings, and IT services, i.e. services executed by IT systems. The second distinction corresponds to the three aspect organizations, as proposed by the organization theorem: the B-organization, the I-organization and the D-organization.

Chapter 5

The Development Process for Service-Oriented Systems

Abstract Definitions of both the notion of Service-Oriented Architecture (SOA) and Service-Oriented Design (SoD) are often not clear or seem to be ambiguous, which makes it hard to compare the various SOA and SoD methodologies. To get a better understanding of SOA and SoD, we apply the Generic System Development Process, a conceptual framework for developing systems of any kind, and specialize it for service-orientation. As a result, we define SOA as a coherent set of design principles that need to be taken into account in the development process of service-oriented systems. SoD deals with the design of service-oriented systems, which are enterprises consisting of humans and often (though not necessarily) also of IT systems. We distinguish between two activities in SoD: function design and construction design of the service-oriented system. Function design deals with designing the black-box model of the service-oriented system, i.e. specifying its externally visible behavior. Construction design deals with the design of the highest-level white-box model and with the decomposition of the highest-level white-box model to the lowest-level white-box model. An important constructional principle in service-orientation is modularization. We show that a service-oriented system consists of modules, which are also service-oriented systems themselves, i.e. they are subsystems. These modules interact by using each other's services. We distinguish between three types of modules based on the types of services they contain; they can contain either ontological services, infological services, or datalogical services.

5.1 Introduction

In chapter 4 we defined the notion of 'service'. We made a distinction between ontological, infological, and datalogical services based on the type of production fact they bring about. Services can either be implemented by human beings (human services) or by IT systems (IT services). Now we know what a service is, we can look deeper into the concepts of Service-Oriented Architecture (SOA) and Service-Oriented Design (SoD). Comparing different methodologies for service-orientation is quite a challenge because they lack a common view of SOA, SoD and the steps involved in design. The contribution of this chapter is to present a clear terminology for SOA and SoD based on the Generic System Development Process (GSDP) (Dietz, 2008; Hoogervorst and Dietz, 2008). The GSDP has been devised for the sake of understanding the mental activity of designing systems of any kind more profoundly than it is commonly the case. One could consider the GSDP as a holistic theorem about designing. Such a theorem was needed for the emerging discipline of *Enterprise Engineering* (Hoogervorst, 2009). Our goal is not to create a complete method for the development of service-oriented systems. It would be impossible to use the GSDP for this purpose as the detailed steps involved in designing a system depend on the type of system under consideration. Rather we want to use the GSDP to determine the scope of different methodologies for service-orientation. The results of the analysis and positioning of various state-of-the art methodologies for service-orientation is presented in chapter 6.

The remainder of this chapter is structured as follows. We explain the GSDP in section 5.2. Section 5.3 presents the specialization of the GSDP for service-orientation. In section 5.4 we show how Business Component Identification 3D (BCI-3D) can be used to delimit coarse-grained modules of service-oriented systems. Finally, we draw our conclusions in section 5.5.

5.2 The GSDP

Many definitions of SOA (OASIS, 2006b; OMG, 2006; The Open Group, 2006; W3C, 2006a) mention the notion of architecture, architectural style or paradigm, but they lack a clear definition of these notions. Next, when a definition of architecture is provided, it is usually the *descriptive* definition, meaning that architecture is conceived as high-level models, referred to by names like ‘high-level components’ or ‘blue prints’. Among others, The Zachman Institute for Framework Advancement (2007) uses this notion: “Architecture is that set of design artifacts, or descriptive representations, that are relevant for describing an object, such that it can be produced to requirements as well as maintained over the period of its useful life”. Another definition that is often, explicitly or implicitly, adopted in the context of SOA is the definition from the IEEE 1471 standard (Maier et al., 2001): “the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”. This makes the exact relationship between architecture and design unclear.

Figure 5.1 exhibits the GSDP. This process depicts the most basic steps that need to be taken in designing a system of any kind. The GSDP is based on the *prescriptive* notion of architecture, i.e. it defines architecture conceptually as the normative restriction of design freedom. Architecture comes in addition to requirements. Such a restriction is necessary and useful because the design freedom of designers, particularly in the field of software engineering, is undesirable large. Practically, architecture is seen as a consistent and coherent set of design principles that embody general requirements” (Dietz, 2008).

Whether the system is a software system, a car, or an enterprise, according to the GSDP two different perspectives can and must always be distinguished: the function perspective and the construction perspective (Dietz, 2006b, 2008). Taking the function perspective, one ‘sees’ the function and the (external) behavior of a system; the corresponding type of model is the *black-box model.* Taking the construction perspective, one ‘sees’ the construction and the operation of a system; the corresponding type of model is the *white-box model.* In developing a system both the function perspective and the construction perspective are relevant. The GSDP defines the most basic steps in a development process. The starting point is the need by some system, called the *using system* (US), of a supporting system, called the *object system* (OS). A clear distinction between the US and the OS is often neglected, leading to blurred discussions about the functionality of the OS. In the GSDP, the US is the stable starting point for the development process. One must have an appropriate understanding of the US in order to successfully design an OS. By nature, this understanding must be constructional understanding, since it is the construction of the US that is going to be supported by the function of the OS. So one starts with conceiving a white-box model of the US.

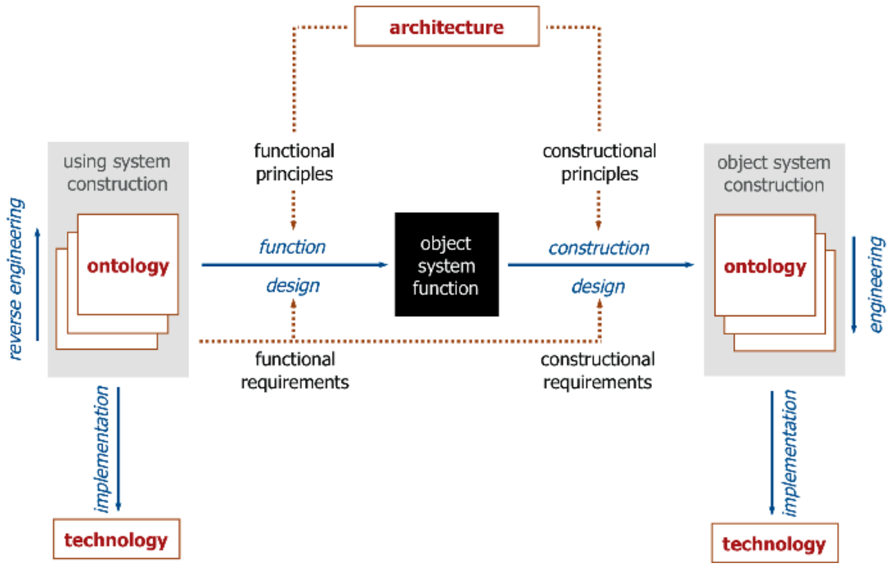


Figure 5.1: The generic system development process (Dietz, 2008).

Preferably it is an ontological model (Dietz, 2006b), since otherwise one can easily become confused by irrelevant implementation issues of the US. From the white-box model of the US one determines the functional requirements for the OS (function design). These requirements are by nature formulated in terms of the construction and operation of the US. Consequently, they need to be fully independent of the construction of the OS. The next basic design step is to devise specifications for the construction and operation of the OS, in terms of a white-box model of the OS (construction design). For this design phase, the US may provide constructional requirements. A thorough analysis of the white-box model of the OS must guarantee that building the OS is feasible, given the available technology. The GSDP also includes the important experience from practice that designing is an iterative process: the final result of every design process is (or should be) a balanced compromise between reasonable functional requirements and feasible constructional specifications (Dietz and Albani, 2005). For the sake of simplicity, however, we did not include it in the figure. In addition to the functional and constructional requirements, there may be functional and constructional principles respectively. These design principles are the operational shape of the notion of architecture. They generally hold for a class of systems. An example of a functional principle is that man-machine dialogs must comply with some standard. An example of a constructional principle is that the applications must be component-based. Ideally the construction design phase results first into an ontological model of the OS, i.e. a white-box model that is completely independent of its implementation. Gradually this ontological model is transformed into more detailed (and more implementation dependent) white-box models, the last one being the implementation model. This process is called engineering. If the OS is a software application, then the implementation model would be the source code in some programming language. The act of implementing consists of assigning appropriate technological means to the implementation model, e.g. running the source code on an appropriate platform (also often called deployment).

5.3 Specializing the GSDP

In the previous section we explained the GSDP. In this section we will show how the GSDP can be used for clarifying the notions of service-orientation by specializing the GSDP for service-oriented systems.

5.3.1 Design of Service-Oriented Systems

As we have seen in chapter 4, services can be implemented by human beings or IT systems. A service-oriented system is an enterprise consisting of humans and often (though not necessarily) also of IT systems. Both the black-box model and the white-box model of an enterprise are relevant to service-orientation: the black-box model for specifying and using services offered by a service-oriented system and the white-box model for building or changing services offered by a service-oriented system. We can decompose a service-oriented system into coarse-grained modules. These modules are again service-oriented systems that offer each other services. So they can also be regarded from a black-box and white-box perspective.

As explained in the previous chapter the organization of an enterprise is the layered integration of three aspect organizations: the B-organization, the I-organization, and the D-organization. These organizations offer ontological services, infological services, and datalogical services, respectively. We can apply the GSDP for designing all three service-oriented organizations. When we apply the GSDP to the B-organization, the using system is the market to which the enterprise offers its services (Dietz, 2008) and the B-organization is the object system. We see two different activities in SoD: function design of the service-oriented system and construction design of the service-oriented system. Function design deals with identifying and specifying the services offered by the service-oriented system. *Service identification* is in our view the first step in function design. It deals with determining what services are offered by the service-oriented system. In this case we are interested in what services are offered by the B-organization to the market. *Service specification* (including Quality-of-Service requirements) is part of function design as well, since it specifies the external behavior of the offered services without caring about it internals. The market requires specifications of these services to be able to interact with the B-organization (see Figure 5.2). Once the specifications of the services that the B-organization offers to the market are available, one can design its construction. The first step in the construction design of the service-oriented B-organization is to create its ontological model. This is the highest-level white-box model of the B-organization. The ontological model of the B-organization provides an overview of all ontological services of the enterprise (those offered to the market as well as those offered only internally). The next step is to create lower level constructional models. This is called the engineering of the service-oriented system. The main constructional principle in service-orientation is modularity. That is why the following step in construction design is to delimit coarse-grained modules of services. Delimiting these modules is not an easy task. In this dissertation we will show how this can be achieved by clustering. The identified modules can interact by calling each other's ontological services (see Figure 5.3). As we explained earlier these modules are subsystems of the service-oriented system. After the delimitation of these modules, the internals of the modules need to be designed. The lower-level construction models are highly-dependent on the type of implementation technology used, i.e. humans or IT systems. The lowest-level construction model is also called the implementation model. In practice, most IT services are not constructed completely in a top-down way because not only new systems, but also existing systems, are used as building blocks for IT services.

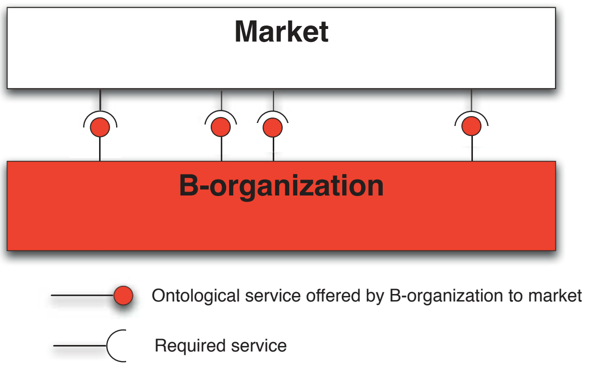


Figure 5.2: Ontological services offered by B-organization to the market

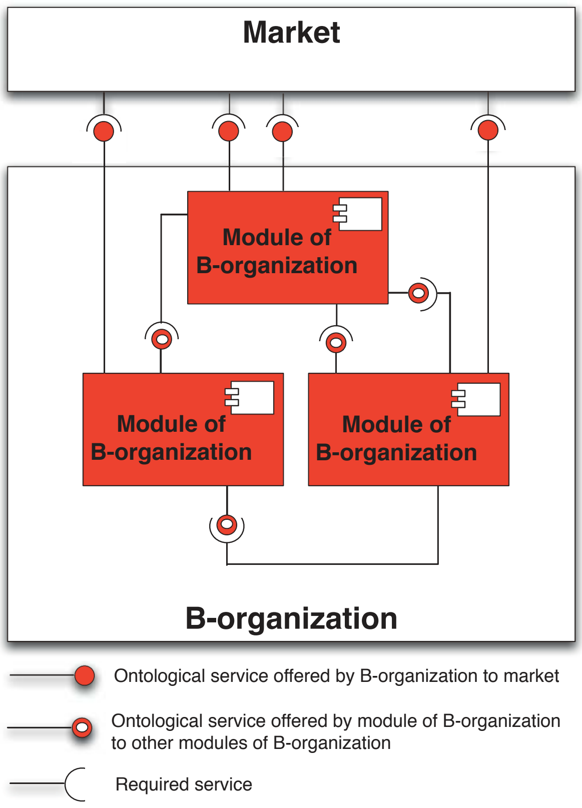

Market
Module of B-organization
Module of B-organization
Module of B-organization
B-organization
Ontological service offered by B-organization to market
Ontological service offered by module of B-organization to other modules of B-organization
Required service

Figure 5.3: Decomposition of B-organization into coarse-grained modules

As depicted in Figure 5.4 the I-organization offers infological services to the B-organization and the D-organization offers datalogical services to the I-organization. When the construction of the B-organization is designed, the I-organization can be designed. Again, the GSDP can be applied. Now the using system is the B-organization and the object system is the I-organization. When one starts with designing the function of the I-organization one needs to determine what infological services are required by the B-organization. To create the highest-level construction model of the I-organization all infological services need to be defined. Based on this model, coarse-grained modules of infological service can be defined (see Figure 5.5). The GSDP is applied for the third time to determine the services that the D-organization needs to offer to the I-organization (i.e. what is the function of the D-organization) and to determine how the D-organization is constructed. So in applying the GSDP now the using system is the I-organization and the object system the D-organization.

5.3.2 Implementation of Service-Oriented Systems

The implementation of the object system is the assignment of technology to the lowest-level white-box model (the implementation model). Hoogervorst and Dietz (2008) use technology in the broadest possible meaning, e.g. human beings, software systems, electronic machines. In the context of service-orientation this means that services are either allocated to human beings or to IT systems. When speaking of IT services, this is usually called deployment. Most enterprises have at least two test environments to deploy IT services after they have been developed: a test environment for verification (checking whether the service matches the specifications) and one for validation (checking whether the service is useful to potential service consumers). Both tests can lead to repeated execution of previous steps in the design process and redeployment to the test environment. Finally, the services are deployed to the production environment. For human services implementation means that the humans who need to execute the services are informed about what they need to do and educated if needed. Also, they need to receive the required procedures that they need to follow.

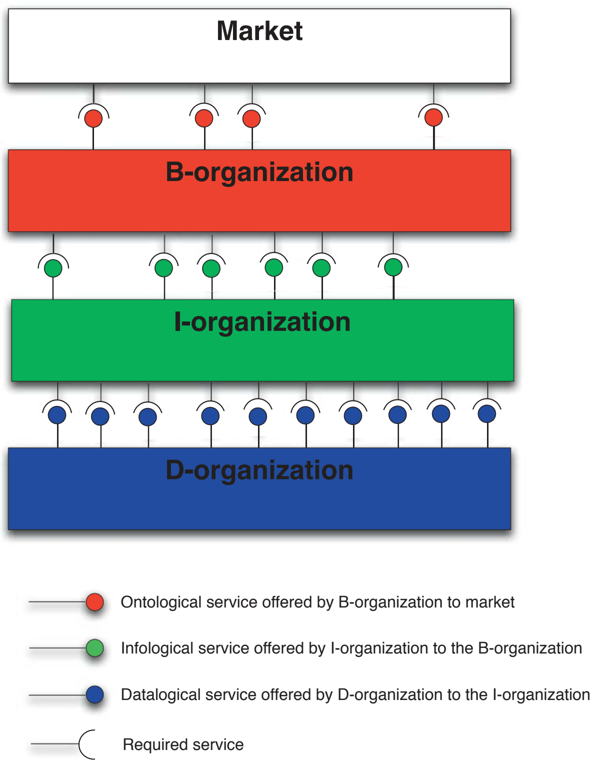

Market
B-organization
I-organization
D-organization
Ontological service offered by B-organization to market
Infological service offered by I-organization to the B-organization
Datalogical service offered by D-organization to the I-organization
Required service

Figure 5.4: Services offered by B-, I-, and D-organization

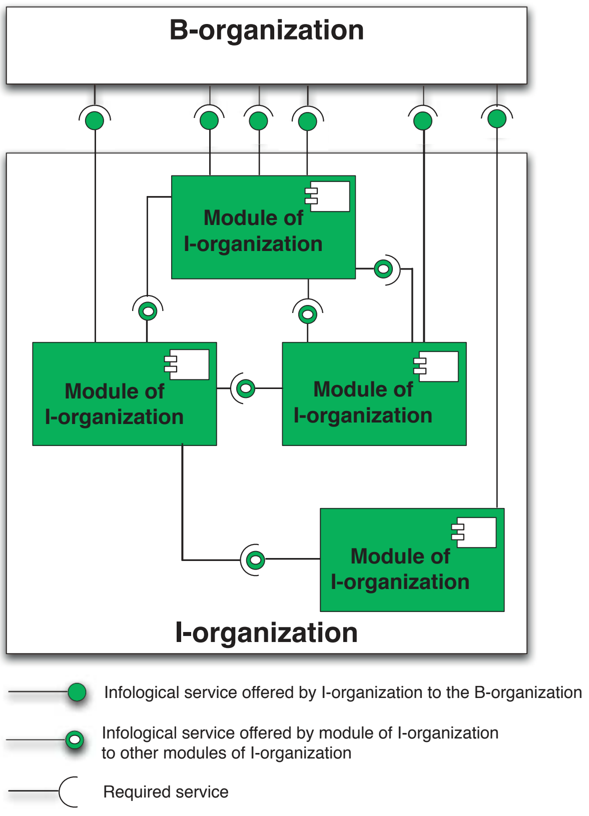

B-organization
Module of I-organization
Module of I-organization
Module of I-organization
Module of I-organization
I-organization
Infological service offered by I-organization to the B-organization
Infological service offered by module of I-organization to other modules of I-organization
Required service

Figure 5.5: Decomposition of I-organization into coarse-grained modules

5.3.3 Service-Oriented Architecture

The architecture of a system consists of the (functional and constructional) design principles that have been or are going to be applied in designing the system. We define SOA as a consistent and coherent set of design principles that need to be taken into account in the development process of service-oriented systems. The functional principles deal with the external function and behavior of a service-oriented system, e.g. "A service must be compliant with the national law". The constructional principles deal with the internal construction and operation of the service-oriented system, e.g. "Modules must be constructed with commercial-of-the-shelf components".

Marks and Bell (2006) provide nine criteria that services must meet: coarse-grained, well-defined service contracts, discoverable, business aligned, re-usable, durable, loosely coupled, composable, and interoperable. Though these criteria as well as the criteria mentioned by other SOA authors provide some guidelines, they are often not precisely enough to make it possible to test whether or not services conform to them. This is still a huge problem in practice. In this research we aim at getting a better insight in different criteria for delimiting coarse-grained modules of service-oriented systems and identifying the services they offer by interviewing practitioners in several case studies.

Table 5.1 shows how we can apply the terminology of the GSDP to service-oriented systems. We can apply this terminology to a service-oriented system as a whole as well as to its coarse-grained modules (which are also service-oriented systems).

5.4 Using BCI-3D for Construction

As we have seen in chapter 3, we need to look at different types of modularity from a business point of view to achieve organizational flexibility (product modularity, process modularity, and organizational modularity). Alignment of these different types of modularity is an important issue as we see in many articles (e.g. Brusoni and Prencipe (2001) and Todorova and Durisin (2008)). Unfortunately, a modular product does not automatically result in a modular organization structure (Hoetker, 2002) and a wrong organization structure can even lead to poor choices in product design as product designs tend to 'follow' the organization design (Conway, 1968).

Term	**Meaning**	**End result**
SOA	a coherent set of design principles that need to be taken into account in the development process of service-oriented systems	not applicable
Function design of the service-oriented system	the design of the external behavior of the service-oriented system	the specifications of the services that the service-oriented system offers
Construction design of the service-oriented system (including engineering)	the design of the highest-level white-box model and the decomposition of the highest-level white-box model to the lowest-level white-box model of the service-oriented system	the design of the internals of the service-oriented system
Implementation of service-oriented systems	the mapping of the lowest-level white-box model of the service-oriented system to technology, i.e. the deployment of services	the deployed services (to human beings and/or IT systems) of the service-oriented system

Table 5.1: Terminology derived from applying the GSDP to service-orientation

Due to its transactional nature, business processes in the ontological model of the enterprise are highly aligned with the (decomposable) product structure. Business Component Identification 3D (BCI-3D) is a method for identifying coarse-grained modules that uses a business model as input. These coarse-grained modules can contain humans as well as IT systems. Several papers are published (Albani et al., 2005; Albani and Dietz, 2006; Dietz and Albani, 2009) on how to use the ontological model of the enterprise as input for the method.

To further explain how the notion of Enterprise Ontology and BCI-3D can help in delimiting modules we introduce the Design Structure Matrix (DSM) (Browning, 2002), a general (so not IT-specific) means to deal with modularity. DSMs are used by BCI-3D for analyzing the dependencies among transactions, among information objects, and between transactions and information objects. Browning (2002) explains how DSMs provide help in dealing with product, process, and organization modularity. A DSM is a square matrix with identical row and column labels. When reading down a column one sees the input sources and reading across a row one sees the output sinks. Figure 5.6 depicts an example in which module A provides input to module C, module B to module A, and module C to module D. Browning (2002) distinguishes between several different kinds of DSMs.

	A	B	C	D
A	A		●	
B	●	B		
C			C	●
D				D

Figure 5.6: Example of a DSM

Figure 5.7 shows his DSM taxonomy (adapted from Browning (1999)). Static DSMs represent elements that exist simultaneously and time-based DSMs represent elements that 'flow in time'. The two types of static DSMs are called component-based DSM and people-based DSM. The first is used for modeling components and their relationships, i.e. it deals with product modularity. The second is used for modeling organization structures based on people and/or groups and their interactions, i.e. it deals with organization modularity. In time-based DSMs we find two subtypes: activity-based DSMs and parameter-based DSMs. The first is used for modeling processes and activity networks based on activities and their information flow and other dependencies. The second is used for modeling low-level relationships between design decisions and parameters, systems of equations, subroutine parameter exchanges, etc. Browning states that gaining a better understanding between product structures and organization structures is a promising area for further research. As said before, the notion of Enterprise Ontology can contribute to this to a large degree.

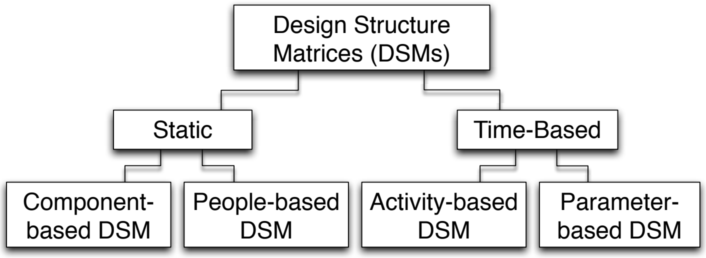
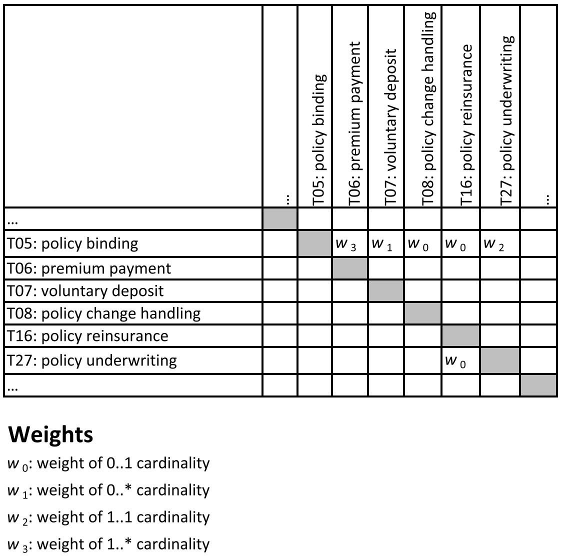


Figure 5.7: DSM taxonomy of Browning (2002), adapted from Browning (1999)

Let us show how BCI-3D uses the notion of DSMs by creating three matrices as an example. The first matrix is an activity-based matrix based on the ontological process model (see Figure 5.8). The second one is a component-based matrix based on the ontological state model (see Figure 5.9). As depicted in the figures, BCI-3D adds the concept of weights to the matrix. In the DSMs presented by Browning only dots are used to indicate a relationship. BCI-3D distinguishes between different weights for the relationships based on the Enterprise Architecture (using the normative definition, i.e. a set a design principles). So we can see the matrices of BCI-3D as *weighted DSMs*, as also proposed by Hoffman and Eugster (2008).

Also, we need a third matrix. This matrix defines the weights of the relationships between the transactions from the ontological process model and the information objects derived from the ontological state model (this matrix is not depicted). We can get an overview of all relationships by using a directed graph as a means for representation (see Figure 5.10). So three different DSMs are combined (transactions/transactions, transactions/information objects, and information objects/information objects). The transactions are depicted as blue spheres and the information objects as red spheres (this graph only represents a part of the complete graph of the enterprise). The arrows depict the direction of the relationships and the label shows the weight of the relationship. Taking the weights of the relationships as input, BCI-3D can calculate modules that have maximum cohesion and minimal coupling.

As processes in the ontological model are automatically aligned to the product structure, alignment between product modularity and process modularity is not an issue. Also, the ontology model can be used for checking



	…	T05: policy binding	T06: premium payment	T07: voluntary deposit	T08: policy change handling	T16: policy reinsurance	T27: policy underwriting	…
…	▨							
T05: policy binding		▨	w_3	w_1	w_0	w_0	w_2	
T06: premium payment			▨					
T07: voluntary deposit				▨				
T08: policy change handling					▨			
T16: policy reinsurance						▨		
T27: policy underwriting						w_0	▨	
…								▨

Weights

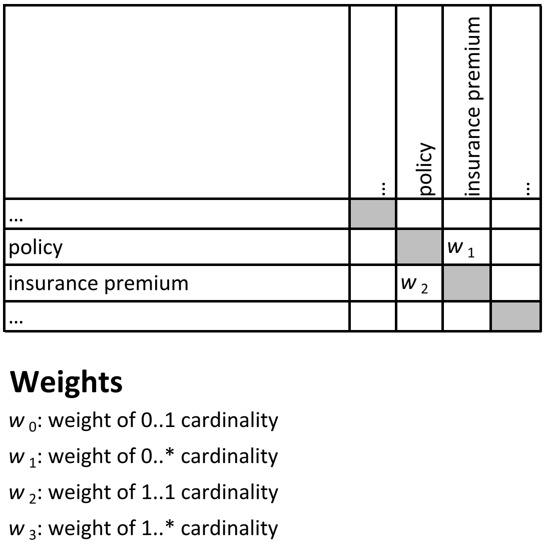
w_0: weight of 0..1 cardinality

w_1: weight of 0..* cardinality

w_2: weight of 1..1 cardinality

w_3: weight of 1..* cardinality

Figure 5.8: DSM transactions



	…	policy	insurance premium	…
…	▨			
policy		▨	w_1	
insurance premium		w_2	▨	
…				▨

Weights

w_0: weight of 0..1 cardinality

w_1: weight of 0..* cardinality

w_2: weight of 1..1 cardinality

w_3: weight of 1..* cardinality

Figure 5.9: DSM information objects

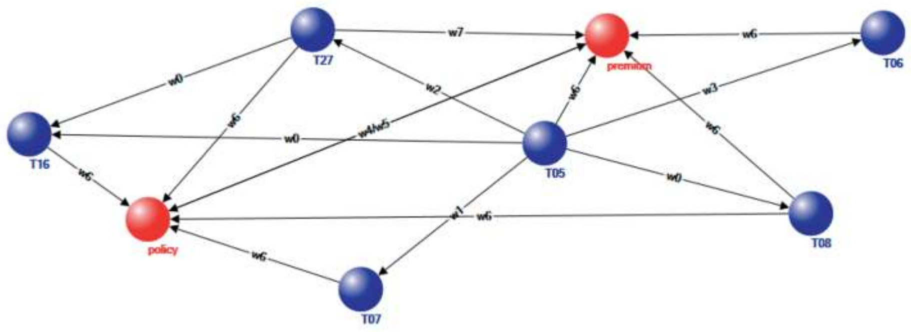


Figure 5.10: Graph representation of three DSMs

the alignment between product/process modularity and organizational modularity. Take, for instance, the *result structure chart* (Dietz, 2006b) depicted in Figure 5.11. Figure 5.12 depicts one of the coarse-grained modules that is the result of applying BCI-3D with high values for weights of relationships among transactions. This module offers the service that is part of T02 to its environment. This service needs to be called by the module that contains the service that is part of T01.

Figure 5.13(a) shows how actors are clustered (an actor is located in the module containing the transaction it executes). The example module presented in 5.12 corresponds with the grouping of actors in Figure 5.13(a) module C. However, Op 't Land (2008) states that communication lines as found in the ontological model of the enterprise are not the only criteria for making decisions about the grouping of actors. Additional criteria include, for example, 'keep actors together, when they use the same language/culture' and 'keep actors together, when they operate under the same regulatory, legal and tax-regime'. So we see that criteria exist for structuring an organization that are not directly related to the product or process structure of the enterprise. When we also take these criteria (by assigning different weights to relations of transactions performed by certain actors) into account we can get a different grouping. A possible example grouping is depicted in Figure 5.13(b). The changes with the original grouping of actors are depicted in red. Fortunately, as Op 't Land also defines the transactional structure as one of the most important criteria, there will be a lot of overlap between both ways of actor grouping. Now let Figure 5.13(c) be the actual organization structure of an organization. When we compare Figures 5.13(b) and 5.13(c), we can see where 'things go wrong'. The differences between the two figures are depicted

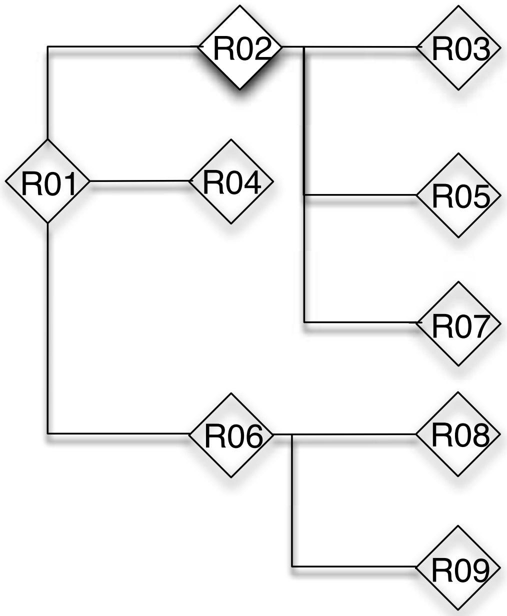


Figure 5.11: Product structure

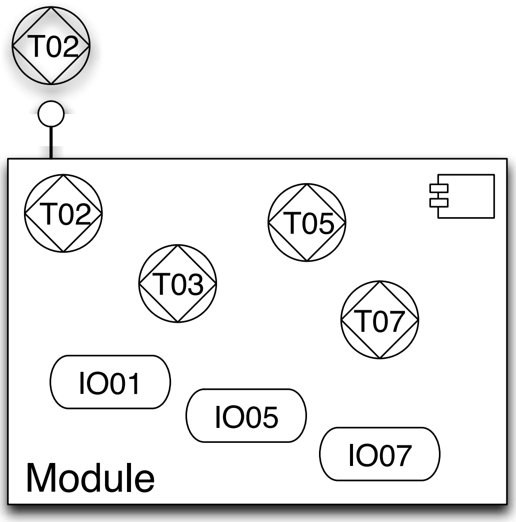
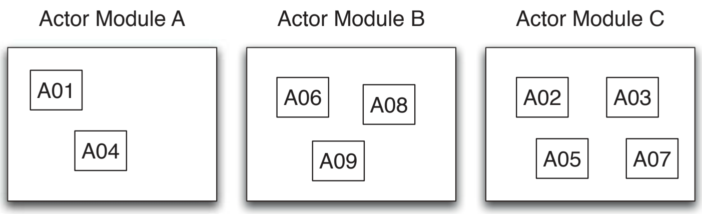
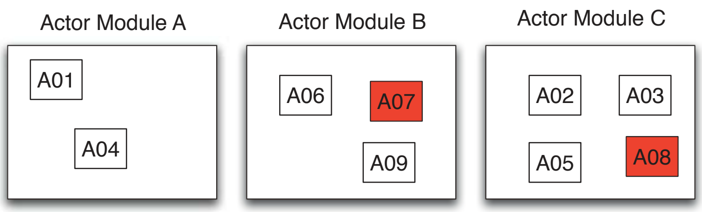


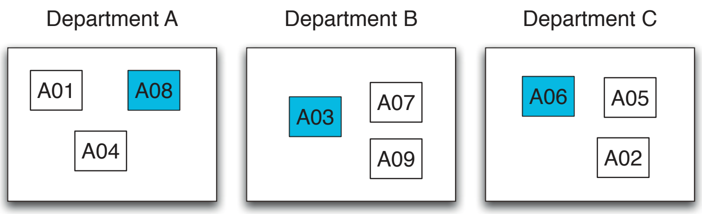
Figure 5.12: Example module of service-oriented system



(a) Actor clusters based on product structure



(b) Actor clusters based on product structure taking into account principles Op 't Land

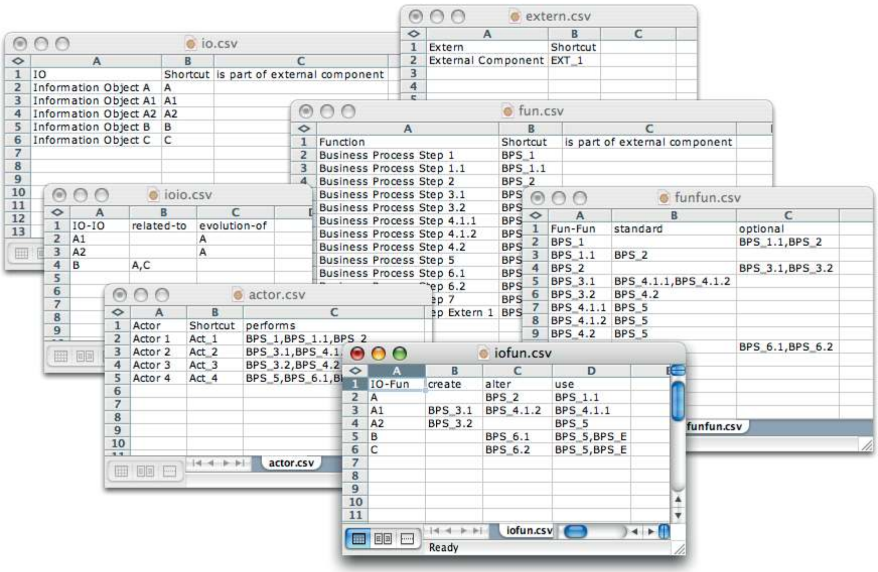


(c) Actual organization structure

Figure 5.13: Clusters of actors

in blue.

Figure 5.14 shows screenshots on how a model is entered as input for BCI-3D. We see that the relationships among transactions are captured in the table funfun.csv. The relationships among information objects are specified in the table ioio.csv. Relations between transactions and information objects are captured in the table iofun.csv. Also, we can specify which actor executes which transaction (table actor.csv) and, if relevant, the relationships with external modules (extern.csv). Figure 5.15 shows example parameter settings of BCI-3D in which the different weights are defined. These weights are used to define the strength of the relation. The result of the application of BCI-3D is an overview of identified modules. Figure 5.16 shows a graphical representation of this result. The transactions and information objects that are grouped together in the three-dimensional graph belong to the same module. In this example we see three modules: (i) the module consisting of BPS 1, BPS 1.1, BPS 2, BPS 3.2, BPS 4.2, IO A, IO A2, and IO B, (ii) the module consisting of BPS 3.1, BPS 4.1.1, BPS 4.1.2, BPS 5, BPS 6.1, BPS 6.2, BPS 7, IO A1, and IO C, and (ii) the external module BPS E. BPS is an abbreviation for business process step (as BCI-3D can also be used with other business modeling approaches). In our case the business process step is a transaction. IO is short for an information object.


io.csv

	A	B	C
1	IO	Shortcut	is part of external component
2	Information Object A	A	
3	Information Object A1	A1	
4	Information Object A2	A2	
5	Information Object B	B	
6	Information Object C	C	

extern.csv

	A	B	C
1	Extern	Shortcut	
2	External Component	EXT_1	

fun.csv

	A	B	C
1	Function	Shortcut	is part of external component
2	Business Process Step 1	BPS_1	
3	Business Process Step 1.1	BPS_1.1	
4	Business Process Step 2	BPS_2	
5	Business Process Step 3.1	BPS	
6	Business Process Step 3.2	BPS	
7	Business Process Step 4.1.1	BPS	
8	Business Process Step 4.1.2	BPS	
9	Business Process Step 4.2	BPS	
10	Business Process Step 5	BPS	
11	Business Process Step 6.1	BPS	
12	tep 6.2	BPS	
13	ep 7	BPS	
14	ep Extern 1	BPS	

ioio.csv

	A	B	C
1	IO-IO	related-to	evolution-of
2	A1		A
3	A2		A
4	B	A,C	

funfun.csv

	A	B	C
1	Fun-Fun	standard	optional
2	BPS_1		BPS_1.1,BPS_2
3	BPS_1.1	BPS_2	
4	BPS_2		BPS_3.1,BPS_3.2
5	BPS_3.1	BPS_4.1.1,BPS_4.1.2	
6	BPS_3.2	BPS_4.2	
7	BPS_4.1.1	BPS_5	
8	BPS_4.1.2	BPS_5	
9	BPS_4.2	BPS_5	
			BPS_6.1,BPS_6.2

actor.csv

	A	B	C
1	Actor	Shortcut	performs
2	Actor 1	Act_1	BPS_1,BPS_1.1,BPS_2
3	Actor 2	Act_2	BPS_3.1,BPS_4.1.
4	Actor 3	Act_3	BPS_3.2,BPS_4.2
5	Actor 4	Act_4	BPS_5,BPS_6.1,B

iofun.csv

	A	B	C	D
1	IO-Fun	create	alter	use
2	A		BPS_2	BPS_1.1
3	A1	BPS_3.1	BPS_4.1.2	BPS_4.1.1
4	A2	BPS_3.2		BPS_5
5	B		BPS_6.1	BPS_5,BPS_E
6	C		BPS_6.2	BPS_5,BPS_E

Ready

Figure 5.14: Tables of relationships used by BCI-3D (Birkmeier, 2008)

IO-IO
- related-to: 5 - part-of: 8 - evolution-of: 6 - state-of: 7
BPS-BPS
- standard (1..1): 20 - optional (0..1, 0..n): 15 - frequent (1..n): 25 - notify: 15
BPS-IO
- create: 10 - use: 5
ACT
- Actor: 20

Figure 5.15: Example parameter settings (Birkmeier, 2008)

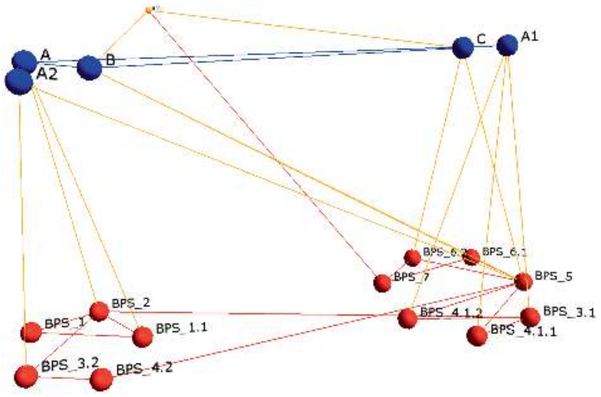


Figure 5.16: Example graph produced by BCI-3D (Birkmeier, 2008)

5.5 Conclusions

Currently, most methodologies for service-orientation fail to make a clear distinction between the notions of SOA (Service-Oriented Architecture) and SoD (Service-oriented Design). The main contribution of this chapter is an elucidation of these notions based on the Generic System Development Process (GSDP). SOA has been defined as a consistent and coherent set of design principles that need to be taken into account in the development of service-oriented systems. We have defined SoD as the design part of a development process, according to the GSDP. It consists of producing successive conceptual models of the object system under consideration.

We introduced the following phases in the development process for service-oriented systems: function design of service-oriented systems, construction design of service-oriented systems (including engineering), and implementation of service-oriented systems. Also, we made a distinction between the B-organization, the I-organization, and the D-organization. The B-organization offers services to the market, the I-organization offers services to the B-organization, and the D-organization offers services to the I-organization.

Function design refers to design of the external behavior of a service-oriented system. This means identifying the services that the system offers to its environment and making specifications of these services. Construction design refers to designing the internal structure of the service-oriented system. These service-oriented systems can be constructed of modules that offer services to each other. As these modules are also service-oriented systems, they also require function design (design of external behavior) and construction design (design of internal structure). One way to delimit these coarse-grained modules is by applying BCI-3D. This method uses the notion of a Design Structure Matrix (DSM) to find modules with maximum cohesion and minimal coupling. BCI-3D provides an overview of these modules and the required interaction. It only provides a high-level white-box model of the internal structure of the modules. Implementation of a service-oriented system refers to assigning human beings or IT systems to its lowest level white-box model (implementation model). In this step we deploy the service to an IT system or a human being.

As said, BCI-3D is a method to delimit coarse-grained modules of service-oriented systems. This method, however, is not the only method available for the development process of service-oriented systems and it does not cover the complete scope of the development process. Using the high-level distinction of phases presented in this chapter we are able determine the scopes of methodologies for service-orientation (which we will do in the next chapter).

Chapter 6

Positioning Methodologies for Service-Orientation

Abstract We use the Development Process for Service-Oriented Systems to analyze and position six state-of-the art methodologies for service-orientation. Two criteria are applied for evaluating the methodologies. One is the coverage of the system development process. The other criterion is the depth in which each of the development phases are dealt with. Two of the methods cover the whole development process, namely the methodology of Papazoglou en van den Heuvel and SOMA. However, they do not elaborate all phases very thoroughly. Some of the specialized methodologies provide an in-depth contribution to specific steps of the development process, like SMART for the construction of the internal structure of the modules offering services and the Goal Driven Approach for delimiting modules and allocating services to them. None of methodologies combines full coverage with full depth.

6.1 Introduction

In the previous chapter we specialized the Generic System Development Process (GSDP) for service-orientation. We also showed how Business Component Identification 3D (BCI-3D) can be used for delimiting coarse-grained modules of service-oriented systems. BCI-3D, however, does not cover the complete development process. In this chapter a positioning of six other state-of-the-art methodologies for service-orientation is presented and discussed. For this positioning we apply the coverage of the development process as a criterion, i.e. we look to what extent the different methodologies cover all activities within our GSDP-based process, as well as the depth with which this is done.

This chapter is structured as follows. In section 6.2 we discuss work related to the topic of this chapter. After that, we discuss several methodologies in section 6.3. Section 6.4 presents their overall positioning in the GSDP. We give examples on how to use these different methodologies using an insurance case example in section 6.5. Section 6.6 concludes this chapter.

6.2 Related Work

Though much is written about methodologies for service-orientation, only little literature is available on the comparison of these methodologies. The most closely related research to our research is the work by Ramollari et al. (Ramollari et al., 2007). They look into ten methodologies, including Service-Oriented Modeling and Architecture (SOMA) (Arsanjani and Allam, 2006), Service-Oriented Architecture Framework (SOAF) (Erradi et al., 2006b), and the methodology of Papazoglou and van den Heuvel (Papazoglou and van den Heuvel, 2006). The authors analyze the following aspects of the methodologies: (i) delivery strategy, (ii) lifecycle coverage, (iii) degree of prescription, (iv) availability, (v) process agility, (vi) adoption of existing processes, techniques, and notations, (vii) industrial application, and (viii) supported role(s). Regarding the coverage of the lifecycle, the methods are classified in the following categories: 'complete', 'analysis and design', 'analysis and design and planning next phases', 'analysis and design and implementation', and 'initial planning'. It remains unclear why the authors have chosen these specific categories. Also, no explanation is provided on the reason why a methodology is classified in a certain category.

Other related work focuses on comparing approaches for executing very specific steps in the service-oriented development process, e.g. service composition (Mantovaneli Pessoa et al., 2008) or the usage of semantic web service approaches (Wahl and Sindre, 2009). Also, several comparison frameworks for component-based development are proposed, e.g. the ones of Boertien et al. (2005), Bunse et al. (2004), and Jisa (2004). Unfortunately, all of them lack sufficient clarity and precision in the definition of the core concepts to be useful for our research.

6.3 The Methodologies

This section presents state-of-the-art methodologies for service-orientation. These methodologies appear to be either *generic*, i.e. having a broad scope and focusing mainly on which steps to take and not how to execute the steps, or *specialized*, i.e. focusing on a specific part and covering it in-depth. Examples of generic methodologies are the methodology of Papazoglou and van den Heuvel (2006), the IBM methodology SOMA (Arsanjani and Allam, 2006), and SOAF (Erradi et al., 2006b). Examples of specialized methodologies in the area of module and service identification are the Goal Driven Approach (Levi and Arsanjani, 2002) and Business Elements Analysis (McGovern et al., 2006). In this section we apply the terminology as used by the authors of the papers describing the methodologies. In section 6.4 we will map the design steps of these methodologies to the steps of the GSDP. Hereafter the generic methodologies are presented first, after which the specialized methodologies follow.

6.3.1 SOAF

The consulting company Infosys Technologies developed Service-Oriented Architecture Framework (SOAF). Our analysis of SOAF is based on the article by Erradi, Anand and Kulkarni (Erradi et al., 2006b). The goal of SOAF is to devise a systematic and repeatable process for implementing Service-Oriented Architecture (SOA). SOAF combines top-down modeling of business processes with the bottom-up analysis of the existing applications. The authors do not mention the origins of SOAF explicitly, except for past experiences with SOA in practice.

SOAF describes a number of activities that need to be performed for migrating to a service-oriented environment. Also, it describes the inputs required by a certain activity and the deliverables created by it. The activities are grouped into five phases. From the total of 19 deliverables, 6 are marked being most important. Not all of the activities are discussed in the article.

1. **Information Elicitation**
   The goal of the information elicitation phase seems to be, though it is not explicitly mentioned, to retrieve and analyze all information required for the service identification phase. Its activities are among others business interviews, business process modeling, and (tool assisted) application mining.

2. **Services Identification and Matching**
   Service identification is the iterative process for arriving at an optimal set of services. The authors advocate a hybrid approach combining top-down domain decomposition along with bottom-up application portfolio analysis. Matching is the process of determining how the top-down identified services should be mapped to the services that existing applications can offer. Some services may need to be developed from scratch and some applications may offer functionality that is not required in the future business process model.

3. **Services Realization**
   The service realization phase deals with making the identified services operational. The authors provide a taxonomy of different service realization approaches containing, for instance, screen scraping, rehosting, and adapter-based wrapping.

4. **Roadmap and Planning**
   The goal of this phase is to reduce disruptions to the business continuity during the SOA roll-out. This phase involves the detailed planning of projects to implement the transformation initiatives. Also, it deals with identifying business and technical risks and defining migration strategies.

6.3.2 P&H

Papazoglou and Van den Heuvel (University of Tilburg) propose the SoD and Development methodology (Papazoglou and van den Heuvel, 2006). The authors not only provide an overview of the activities required to migrate to a service-oriented environment, but they also explain some of the architectural principles that apply to the functional and constructional design of services, e.g. functional service cohesion, communicational service cohesion, identity coupling, and communication protocol coupling. This methodology is partly based on other related development models, such as Rational Unified Process (RUP) (Rational Software Corporation, 2001; Kruchten, 2003), Component Based Development (Herzum and Sims, 2000) and Business Process Modeling (Harmon, 2003). The authors' starting point is the idea that a good methodology for service-orientation is based on an iterative and incremental process and that it concentrates on business processes. Although the method originates from a university, the authors do not explicitly mention their theoretical points of departure.

The phases of SoD and Development methodology (P&H) are:

1. **Planning**
According to the authors the planning phase of a methodology for service-orientation is very similar to that of software development methodologies, including the RUP. This phase determines what the scope of the service-oriented environment is and how feasible the wishes of the organization are.

2. **Analysis**
The analysis phase deals with the requirements of the service-oriented environment. Business process (re)design is the starting point of this phase, which has as goal to reuse business process functionality, i.e. application logic directly supporting the business process, in new composite applications. The basis for the new business processes consist of an as-is process model and a to-be process model. The organization creates this to-be process model by designing, simulating, and analyzing potential changes to the current application portfolio for potential Return-On-Investment (ROI) before committing to any of the changes.

3. **Service Design**
The service design phase needs the results from the analysis phase as an input, which are conceptual processes and services, and transforms them into a set of related, platform-agnostic interfaces. This interface is needed to construct the services by producing them from possibly pre-existing components or from new components or by assembling services out of other services. An important aspect of this phase is to think about how to combine higher-level services from existing services. Two main activities of this phase are (i) service specification and (ii) business process specification.

4. **Service Construction**
In this phase the services are constructed either from scratch or by modifying existing services, or by constructing wrappers on top of existing legacy applications.

5. **Service Test**
According to the authors, the most interesting type of testing for service

implementations is dynamic testing. Dynamic testing means running the implementation and comparing its actual to its expected behavior before it is deployed.

6. **Service Provisioning**
   The authors see service provisioning as a central issue in making revenue generating web services. The following aspects need to be taken into consideration for service provisioning: (i) service governance, which includes choosing between a central and distributed governance model, (ii) service certification, which is concerned with properties of a services that can predict end-to-end behavior of composite services, (iii) service metering and rating including choosing a system for measuring the use of services by service consumers, and (iv) service billing strategies which includes deciding which services are offered for free and which services have to be paid for.

7. **Service Deployment**
   The deployment phase deals with rolling out the new processes to all the participants in the service-oriented environment. Some of the activities in this phase are publication of the service interface and deployment of the services at the provider side.

8. **Service Execution**
   The service execution phase means that all services are fully deployed and operational, so the new processes can be and are actually carried out by all participants in the service-oriented environment.

9. **Service Management and Monitoring**
   The service management and monitoring phase is among others concerned with service version control and the monitoring of Service Level Agreements (SLAs).

6.3.3 SOMA

IBM proposes Service-Oriented Modeling and Architecture (SOMA) as a methodology. The first version was presented in 2004. SOMA is an extension to existing IBM analysis and design methodologies, including the Global Services Method, a methodology used by IBM Global Services personnel for client engagements, and the Rational Unified Process, a method for software development widely adopted in industry. SOMA started out as an IBM Global Services Method. Meanwhile IBM was working on SOA specific extensions to the Rational Unified Process. In 2006 the people working on both initiatives joined forces. In 2006 an overview of the method (Arsanjani and Allam, 2006), consisting of three phases, was published. More recently (2008), a more detailed article (Arsanjani et al., 2008) on SOMA has been published. Two additional phases were added: (i) implementation and (ii) deployment, monitoring, and management. As far as we know, SOMA has no strong theoretical basis. Instead it is based on a large number of project experiences over 2002-2004 (Arsanjani, 2006).

SOMA distinguishes between the following five major phases:

1. **Identification**
   This phase deals with determining which services are required. It starts with identifying the business domain that is the focus for transformation to SOA. The next step is to identify the key business processes that support this domain. Candidate services are identified from three inputs: the key business processes, the existing software assets, and business goals.

2. **Specification**
   In the service specification phase the details of the services are designed. The priority of which candidate services to specify first is based on a set of criteria.

3. **Realization**
   The realization phase consists of different types of activities. These activities include the exploration of technical feasibility, the building of prototypes, and the selection of patterns for building the services.

4. **Implementation**
   The implementation phase focuses on building the services, either by building them from scratch, wrapping existing systems, or assembling them from other services.

5. **Deployment, Monitoring, and Management**
   This phase deals with packaging, provisioning, executing user acceptance testing, and deployment of services to the production environment.

6.3.4 Business Element Approach

McGovern et al. (2006) describe the Business Element Approach in their book "Enterprise Service-Oriented Architectures". Its main goal is to make it possible to trace changes to business processes or rules quickly and unambiguously to one or more specific components. The Business Element Approach has its roots in Component Based Development methodologies.

The following phases belong to the Business Element Approach:

1. **Requirements Definition**
   The objective of this activity is to define precisely what services the planned system should provide, how they are carried out, and what business features it should exhibit.

2. **Business Element Analysis**
   The Business Element Approach distinguishes between three kinds of business elements that can be derived from resource and process models, viz. Resource Business Elements (RBE's), Service Business Elements (SBE's), and Delivery Business Elements (DBE's). An RBE, which groups the (information) resources, comprises a focus resource, being independent (e.g. Customer) and its auxiliary resources (e.g. Address), always belonging to a certain focus resource. An SBE consist of the highest-level immediate steps. An intermediate process is a process of which steps are required to complete as soon as possible, and whose intermediate states are of no concern to the business in that they are not required to be remembered after the process is completed. A DBE is a grouping of Service and Resource Business Elements that together deliver a business solution to a business problem, and which provides services to requesters.

3. **Mapping to Components**
   In this phase the business elements that capture important business concepts are mapped to different types of components that are of importance for structuring concepts in IT systems. The SBE's are mapped to Process Business Components, the RBE's to Entity Components, and the DBE's to Application Components.

6.3.5 Goal Driven Approach

The Goal Driven Approach (Levi and Arsanjani, 2002) derives services from business goals. The services are depicted in a Goal Service Graph and they are allocated to enterprise components. This Goal Service Graph provides traceability of IT services to business goals. These enterprise components are identified by clustering highly interdependent (coupled) use cases. One of the authors of the article on the Goal Driven Approach also co-authored the article on SOMA we used in our paper. The Goal Driven Approach finds its basis in the world of Component Based Development and Object-Oriented analysis and design methods. Also, it builds upon formal grammar specification to define domain-specific languages. SOMA (Arsanjani et al., 2008) mentions this method as one of the possible methods for service identification.

The Goal Driven Approach consists of:

1. **Domain Decomposition**
   This phase is meant for identifying business processes and dividing the domain into functional areas based on department boundaries, business process boundaries, and value chains.

2. **Subsystem Analysis**
   After the functional areas of the major business process areas are identified, they are all broken down into their constituent business processes or functional areas as defined by the business analysts and modelers.

3. **Creation Goal Model**
   This phase comprises the identification of high-level business goals and their refinement into sub-goals with explicit dependencies. The common way to retrieve organizational goals is through interviewing people from the business side of a project. Also, a verification of these goals with executives is required since they may be different from what business modelers think they are.

4. **Service Allocation**
   After the structure of the components is identified in the domain decomposition and subsystem analysis phases, it is time to match them to their functions or services that are identified through the Goal Service Graphs.

5. **Specification of Enterprise Components**
   The specification phase focuses on three aspects: services (interfaces), contracts, and manners.

6. **Structuring Enterprise Components**
   Finally, the internals of the enterprise components are designed. The authors explicitly mention their preference for reconfigurable components that are composed of subcomponents in a soft-wired way.

6.3.6 SMART

Service-Oriented Migration and Reuse Technique (SMART) (Lewis et al., 2006) is a methodology that describes in detail how to construct identified services from legacy systems. SMART takes into account that in practice it is often not easy to construct services from legacy systems. Since almost no organization has the luxury to build up its entire IT-environment from scratch, it is important to recognize the risk involved in migrating to a service-oriented environment. According to the SMART methodology an organization needs to thoroughly assess the capabilities of its legacy systems and carefully analyze the risk of migrating. SMART uses code analysis with the ARMIN tool, a process which the authors call architecture reconstruction, to determine the structure of the existing systems. SMART is developed by the Software Engineering Institute (SEI). The US Department of Defense (DoD) sponsored the work. SMART was derived from the Options Analysis for Reengineering (OAR) method developed at the SEI. OAR is a systematic, architecture-centric means for mining existing components for a product line or new software architecture. The method incorporates a set of scalable techniques and exercises to collaboratively analyze existing components, determine viable mining options, and evaluate the most promising options (Smith et al., 2002).

The phases of SMART are (Lewis et al., 2005):

1. **Establish Stakeholder Context**
   The first step in the SMART methodology is to get information about the stakeholders concerned with the migration to service-orientation. These include the owners and end users of the legacy systems, the potential end users of the services, the funders of the project, groups defining the target services, and verification and validation groups that will

certify the properties of the new services. The main goal of this activity is to gain information about who has knowledge about the legacy systems and what the demand for future services is.

2. **Describe Existing Capabilities**
   The goal of the second activity of SMART is to obtain descriptive data about the components of the legacy system, including name, function, size, language, operating platform, and age of the legacy components, design paradigms, code complexity, level of documentation, module coupling, interfaces for systems and users, and dependencies on other components and commercial products. This phase is intended to gather evidence about potential services that can be created from the legacy components and to gather sufficient detail about the target SOA to support decisions about what services may be appropriate and how they will interact with each other and the SOA.

3. **Analyze the Gap between Service-Based State and Existing Capabilities**
   The goal of the fourth activity is to identify the gap between the existing state and the future state, and to determine the level of effort and cost needed to convert the legacy components into services.

4. **Develop Strategy for Service Migration**
   The final activity of SMART involves recommending one or more of the options for mapping services to components, selecting a strategy to achieve the goal, and presenting the SMART team findings.

6.4 Positioning the Methodologies

In the last section we presented six state-of-the-art methodologies for service-orientation having either a generic or a specialized nature. In this section we investigate to what extent the different methodologies cover the GSDP-based development process for service-oriented systems. Table 6.1 exhibits the mapping of the six methodologies to the service-oriented development process derived from the GSDP. The vertical axis consists of the phases of the methodologies as described in section 6.3. The horizontal axis comprises the phases of the specialized version of the GSDP.

Methodology	**Phase**	**Function design**	**Construction design**	**Implementation**
Business Element Approach	Requirements Definition	O	-	-
	Business Element Analysis	O	O	-
	Mapping to Components	-	O	-
Goal Driven Approach	Domain Decomposition	-	O	-
	Subsystem Analysis	-	O	-
	Creation Goal Model	O	-	-
	Service Allocation	O	O	-
	Specification of Enterprise Components	O	-	-
	Structuring Enterprise Components		O	-
SOAF	Information Elicitation	O	-	-
	Services Identification and Matching	O	O	-
	Services Realization	-	O	
	Roadmap and Planning	-	-	-
P&H	Planning	-	-	-
	Analysis	O	-	-
	Service Design	O	-	-
	Service Construction	-	O	-
	Service Test	O	O	-
	Service Provisioning	O	-	-
	Service Deployment	-	-	O
	Service Execution	-	-	-
	Service Management and Monitoring	-	-	-
SOMA	Identification	O	-	-
	Specification	O	-	-
	Realization	-	O	-
	Implementation	-	O	-
	Deployment, Monitoring, and Management	-	-	O
SMART	Establish stakeholder context	-	-	-
	Describe existing capabilities	-	O	-
	Describe the future service-based state	-	O	-
	Analyze the gap	-	O	-
	Develop strategy for service migration	-	O	-

Table 6.1: Classification of methodologies for service-orientation

The Business Element Approach and the Goal Driven Approach deal with function design as well as construction design. The Business Element Approach focuses on function design in its requirements definition phase and in its business element analysis phase. The requirements definition phase defines what services are needed in the service-oriented system. In the business element analysis phase modules are defined. In this phase it is also determined which service is delivered by which module. Additionally, a high-level construction model is provided. Thus the business element analysis phase deals with construction design to some extent. In the mapping to components phase the construction is further defined by making a mapping of the high-level construction models of modules to IT components. In the Goal Driven Approach function design is done in the creation goal model phase. Modules are defined in the domain decomposition and subsystem analysis phases. In the service allocation phase services are allocated to modules, so this is part of function design as well as construction design. The Business Element Approach as well as the Goal Driven Approach deal with construction of the service-oriented system and they identify services for the interaction with the modules. Though they do provide assistance in constructing the service-oriented system (enterprise) as a whole by delimiting modules, they do not provide assistance in designing the internal structure of the modules. The only specialized methodology that has a different focus is SMART. This methodology focuses on constructing the internals of modules, since it looks at how legacy systems can implement the identified candidate services. It assumes other methodologies are applied for acquiring these candidate services. The SMART phase of establishing stakeholder context is not really a design activity; it takes place before the actual development process starts.

Looking at the generic methodologies, we see that P&H and SOMA have the broadest scopes since SOAF does not cover the implementation phase. Let us have a closer look at these three methodologies.

SOAF's information elicitation and its service identification and matching phase both deal with service function design. The latter, however, also deals (though minimally) with service construction design as the ease of composition is also taken into account. The SOAF realization phase places more emphasis on construction design and the decomposition of the highest-level white-box model into lower-level white-box model, viz. service engineering. The planning of the deployment of the service takes place in the SOAF Roadmap and Planning phase. The phase does not deal with the deployment itself.

In the planning phase of P&H some decisions about the function and construction of the services are taken like the business processes they need to support and the existing systems that can be used for implementation. The thorough function design of the services takes place in the analysis and service design phases, after which the construction takes place in the service construction phase. Testing belongs both to the function design and to the construction phase as it is concerned with testing the function as well as the construction (links between the components) of a service. The methodology explicitly mentions the service deployment step (the service implementation phase).

In the paper from 2006 (Arsanjani and Allam, 2006) SOMA consists of three phases: identification, specification, and realization. In later work (Arsanjani et al., 2008) two phases are added: implementation, and deployment, monitoring, and management. However, these two phases are not discussed in detail in the new paper (they are said to be out of scope). SOMA's identification phase as well as its specification phase both map to the function design phase of the GSDP. The SOMA realization phase as well as its implementation phase deal with construction. Finally, the deployment, monitoring, and management phase deals with the implementation phase of the GSDP.

In both SOMA and P&H the authors speak of activities like 'monitoring' and 'management'. However, we do not consider these activities to be part of the development process for services as defined in the GSDP. These activities deal with the operation of the system instead of with its development. The amount of detail in which the phases of the generic methodologies are described is the largest in P&H. Also, this methodology does not only deal with the steps that need to be performed in the service design process, but also emphasizes the principles applied during the design process. These six design principles focus on minimizing coupling between services.

6.5 Examples for Applying the Methodologies to an Insurance Company

In this section we show some examples of applying the different methodologies to the insurance company Protector. The business models from these case are derived from a real-world insurance company. For the sake of simplicity, we only show part of the models.

6.5.1 Introduction

Protector, an insurance company, sells three types of *life insurance* products. The first type of product, the *term life insurance*, protects the beneficiaries of a policy from the financial damage they suffer in case the insured dies during the policy term. The second type, *pension insurance*, can be seen as an insurance that protects the insured from a large income loss if he reaches his pension age or that protects his life partner and young children from large income loss after the insured (would have) reached his pension age in case of the insured's death. The third type, the *capital sum insurance*, is an insurance for building up capital. When the end date of the policy is reached, then the benefit will be paid in a single payment. This product type is, for instance, suitable for saving money to pay off a mortgage. Protector offers multiple products of each type.

Protector sells these products either to a company, i.e. *collectively*, or to an individual person, i.e. *individually.* Some products may be sold both collectively and individually, some only collectively or individually. An example of a collective insurance is a pension insurance provided by a company to its employees. An individual insurance could be a term life insurance related to a mortgage. We use the word *policy* for the individual policy as well as for a participation in a collective contract. An insurance policy has an insurant, one or more insured, and one or more beneficiaries. The insurant is an organization or person that is responsible for the payment of the premium of a policy. The insurant is the client of Protector. The insured is a person who is the 'insured object'. The beneficiary is a person who receives a payment if the insurant has a right to a benefit according to the product rules of a policy. Figure 6.1 depicts the relationship between these and other information objects in UML notation. When a certain insured person would cause a high risk for Protector, because for example the insured amount is high, then the policies are *reinsured* by a *reinsurer.* This means that a part of the insured amount is insured by the reinsurer in order to spread the risk. A reinsurer can be a 'regular' insurance company or an insurance company that is specialized in insuring insurance companies. Sometimes reinsurance is legally obligatory, sometimes it is a choice made by Protector itself.

Protector has two main business goals for the next 5 years: (i) standardizing their product portfolio and (ii) increasing customer satisfaction. Currently, the company tends to create new products for almost every corporate client, which results in huge policy management problems. The company

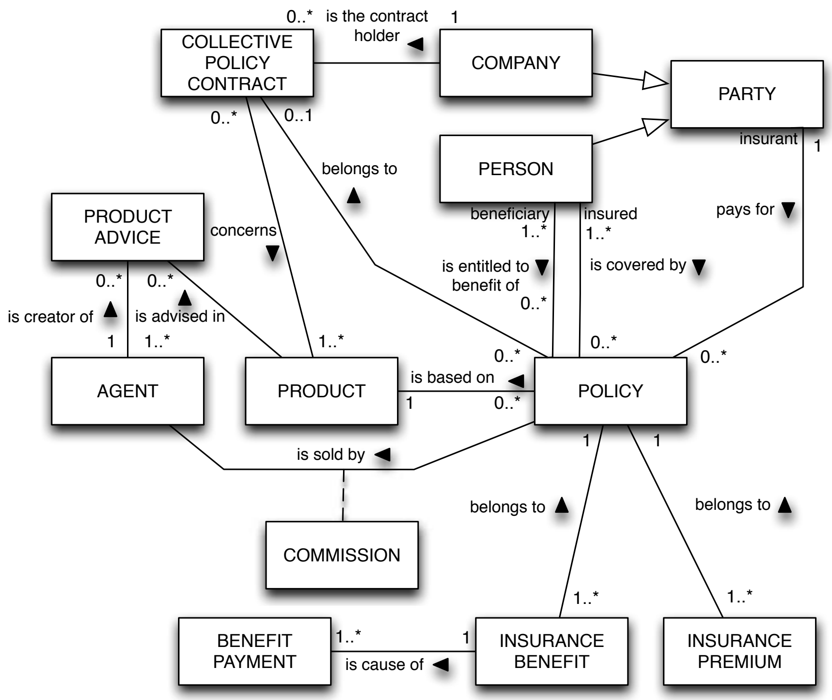


Figure 6.1: Information object diagram for Protector

wants to improve its product management process and restrict the freedom of sales employees in defining their own new products. Customer satisfaction is low because it takes a long time to process changes to a policy. Also, clients are complaining about the lack of support of self service through the Internet. The client has to contact the company by phone or mail. Table 6.2 exhibits the six main software systems of Protector. Protector is considering replacement of the legacy systems.

System	**Purpose**
Siebel	A CRM for managing data related to reinsurers, insurants, beneficiaries, and insured persons
InterAgent	A custom-made system for managing data related agents, the advices they provide, and the commission they receive
OmniPayment	A custom-made system for paying insurance benefits to beneficiaries, for paying reinsurance premium to the reinsurer, and for pay commission to agents
DirectPolicy	A newly acquired commercial system suitable for pension and capital sum policy administration. This system can be used for self-service over the internet.
DinoPolicy	A legacy system for pension policy administration that can only be used by employees of Protector
TermPolicy	A legacy system for term life and capital sum policies that can only be used by employees of Protector

Table 6.2: Software Systems of Protector

6.5.2 Function Design in the Business Element Approach

Services are identified by process decomposition. We first need to look at the top-level immediate steps defined in the process model (i.e. a step "that is required to complete as soon as possible, and whose intermediate states are of no concern to the business in that they are not required to be remembered after the process has completed"). After that, the subsidiary immediate steps are defined. Services identified in this case are among others 'Request

Quotation', 'Change Policy', and 'Add Product to Portfolio'. Let us have a look at the 'Request Quotation' service. For this service we could define the following Subsidiary immediate steps: 'Validate Client Data', 'Record Client Data', 'Validate Policy Related Data', 'Record Policy Related Data', and 'Send Confirmation Letter'.

The method does not deal with service specification.

6.5.3 Function Design in the Goal Driven Approach

We can identify services using Goal Service Graphs. Working from the two business goals mentioned in the case description, we construct the Goal Service Graph as depicted in Figure 6.2. We decompose these business goals until we get a set of services that will achieve a certain goal.

1. Standardize product portfolio
   (a) Migrate expired policies to new standardized products based policies
      - *ConvertPolicy*
   (b) Apply strong product management
      - *Check for duplicate products*
      - *Calculate risk for product*
2. Increase customer satisfaction
   (a) Decrease change processing time
      - *Search for policy*
      - *Calculate effects of change*
   (b) Enable self service through the Internet
      - *Request quotation*
      - *Request policy*
      - *Change policy*

Figure 6.2: Goal Service Graph

When looking at the second step of function design (service specification), the authors speak about Enterprise Component Specification. This is essentially the same as the specification of all services of a module. They address the following three key ingredients of such specification: services (interfaces), contracts, and manners. Services (interfaces) specify "what capabilities the component provides to support the business goals and processes, what it requires to do so, and an abstract specification of the design of how the services realize goals". Contracts refer to the specification of pre- and post-conditions of each service and the sum total of which services are provided and required by the component. Manners specify "how the component in a given state should behave within a given context; which subset of rules to check once an event has been triggered".

6.5.4 Function Design in SOAF

SOAF positions service identification as an iterative process for arriving at an optimal set of services. The authors propose a hybrid approach combining top-down domain decomposition along with bottom-up application portfolio analysis. The activities result in a list of candidate services that further needs to be rationalized and consolidated. The activities are not discussed in sufficient detail for enabling us to apply the ideas to our case.

Service specification is called service description in SOAF. The following aspects are considered to be part of the service specification: the service interfaces and data types of exchanges messages, the behavioral model of the service including the supported message exchange patterns (e.g. one-way/notification or request-response), the supported conversation and temporal aspects of interacting with the service, and the service policy. The service 'BindPolicy' might comprise the specification elements exhibited in Table 6.3.

6.5.5 Function Design in P&H

P&H deals with service identification, though not in much detail. We were not able to identify services from our case using this methodology. The authors, quoting Johnston (2005), mention the following service specification elements: structural specification, behavioral specification, and policy specification. Structural specification refers to how the interface, specified in the Web Service Definition Language (WSDL), is structured, e.g. messages, port types, and operations. The paper does not discuss how this relates to service-oriented environment that do not use Web services. The behavioral specification deals with understanding the effects and side effects of the services

Bind Policy	
Exchange pattern	Request-response
Input	Id of policy data object
Output	DateTime of the moment of binding
Preconditions	Policy *pol* has been created Policy *pol* has been underwritten
Effect	Policy *pol* is bound
Availability	99,5%
Required protocols	XML Encryption XML Signature

Table 6.3: Example of part of a service specification in SOAF

(called 'operations' by the authors) and the semantics of input and output messages. In our case the behavioral specification of the service 'Bind Policy' could state how the consumer cannot be used if the policy is not created and underwritten first. The policy specification denotes policy assertions and constraints on the service. These assertions may cover security, manageability, etc. In our case policy specifications might be that the Bind Policy services (i) uses XML Encryption and (ii) is available 99,5% of the time.

6.5.6 Function Design in SOMA

For the process of service identification SOMA mentions three main complementary techniques: goal service modeling, domain decomposition, and existing asset analysis. For the domain decomposition the authors refer to a technique called variation-oriented analysis and design (VOAD). Existing asset analysis takes a bottom-up approach; it looks at the existing application portfolio and other assets and standards that may be used in identifying good candidate services. We already explained the Goal Driven Approach in subsection 6.5.3. In the paper we used as a source for writing this subsection, the mentioned goal service modeling and domain decomposition techniques were combined. When applying the existing asset analysis approach, we could suggest candidate services like 'Pay Reinsurance Premium' offered by the OmniPayment application and 'Create Term Life Policy' offered by the TermPolicy application.

SOMA addresses the importance of service specification. The specification describes the following aspects of a service (called ‘operation’ in the paper): non-functional requirements, input, output, and error messages. Also, the authors mention a context diagram, i.e. “the service ecosystem for a group of related services illustrating service consumers and service providers and indicating the flow of messages between services and the various underlying systems that implement them”. In our opinion the underlying systems that implement the services would not belong to the service specification.

6.5.7 Construction Design in the Business Element Approach

The Business Element Approach uses the information objects model and the process model to define ‘elements’, i.e. modules. The criteria for identifying Resource Business Elements (RBE’s) are whether they are “real” and “independent”. “Real” means that a Subject Matter Expert (SME) both uses and understands it. An “independent” resource is one that somebody can talk about without first saying to what it belongs. As our information object model is constructed together with SME’s all objects in it are “real”. The policy object is an “independent” resource, the benefit payment, insurance benefit, and insurance premium objects are not. In the example given by the authors, roles are also explicitly modeled as objects, which is not the case in our model. We can, therefore, also consider the following objects as auxiliary resource elements: insurant, insured, and beneficiary. This leads to the RBE depicted in Figure 6.3.

The services are grouped in SBE’s by grouping the steps by RBE. In our case this lead to the results exhibited in Table 6.4.

A DBE is “a grouping of Service and Resource Business Elements that together deliver a business solution to a business problem, and which provides services to requesters”. In our case example DBE’s are Product Management, which contains the Product Service SBE and the Product RBE, and Policy Management, which contains the Policy Service SBE and the Policy RBE, the Person RBE, and the Company RBE.

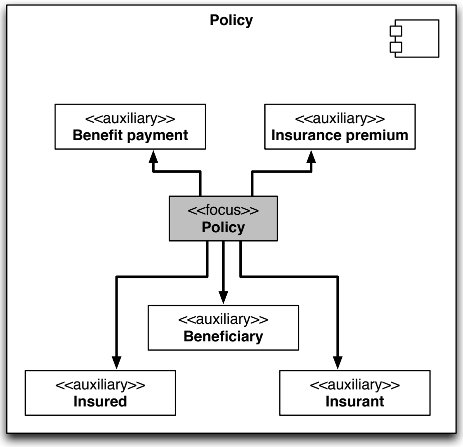

Policy
<<auxiliary>> Benefit payment
<<auxiliary>> Insurance premium
<<focus>> Policy
<<auxiliary>> Beneficiary
<<auxiliary>> Insured
<<auxiliary>> Insurant

Figure 6.3: One of the RBE's of Protector

6.5.8 Construction Design in the Goal Driven Approach

When applying the Goal Driven approach, we define the module boundaries in terms of business processes. Exact criteria for making this division in modules (called components in the article) are not given. A possible business process division is:

- Product Portfolio Management
- Customer Relationship Management
- Quotation
- Policy Management
- Reinsurance

In the E-bazaar example (Levi and Arsanjani, 2002) given by the authors, these major business process areas are broken down into lower level business processes. E.g., product portfolio management can be decomposed as follows: Product Portfolio Management = {Product Development, Product Addition, Product Removal}.

In the service allocation step we assign services to the modules. We can map the 'Calculate risk for product' and the 'Check for duplicate products' services to the 'Product Portfolio Management' module. In the process of

SBE	**Service**	**Subsidiary Immediate Steps**
Policy Service	Request Quotation	Validate Client Data Record Client Data Validate Policy Related Data Record Policy Related Data Send Receival Confirmation Letter
	Change Policy	Validate Requested Changes Calculate Risk Involved With Changes .....
	.....	.....
Product Service	Add Product to Portfolio	Check for Duplicate Products Insert Product
	Remove Product from Portfolio	
	.....	.....
	Develop Product	Make Initial Product Design Calculate risk for product Make Product Final .....

Table 6.4: Two of the SBE's of Protector

developing a new product, we need to calculate the risk involved in a certain product (mostly by statistical analysis). The second service is used in the process of adding new products to the portfolio to decide whether it is sensible to start working on a new product.

6.5.9 Construction Design in SOAF

SOAF does not describe in detail what steps to take for construction design, but it does present a taxonomy of service realization approaches. The precise steps depend on the approach taken. The taxonomy distinguishes between non-invasive and invasive approaches. Non-invasive service enablement is defined as "a tactical approach to align existing systems to business needs through wrapping by using new layers of flexible technologies such as Enterprise Application Integration (EAI) solutions, messaging tools, and recently with standardized interfaces using web services". Invasive transformation is defined as "a strategic approach that aims to revitalize and streamline the application portfolio to ease maintenance, extension, and interoperability". In our Protector case this means the following. Let us say we need a service for 'Policy Quotation'. A non-invasive approach would be constructing a service by screen scraping a legacy application like DinoPolicy or by creating a Java DataBase Connectivity (JDBC) adapter for an existing database to get some additional required data. In an invasive approach, one would reconsider the current application landscape. Examples are to reengineer Dinopolicy to include the additional data or to buy a COTS system that comprises the complete functionality.

6.5.10 Construction Design in P&H

The authors discuss the following possibilities for creating a service: (i) starting from scratch, (ii) use an existing application to construct the service, and (iii) compose a new service from other services. They discuss green-field development, top-down development, bottom-up development, and meet-in-the-middle development. Green-field development assumes that first the service is constructed and then the service interface is generated. Top-down development starts with a service interface after which the service is constructed. Bottom-up development assumes the service interface is developed from an existing application. Meet-in-the-middle refers to the mapping of an already existing service interface (for which the service is already constructed) to a new service definition.

6.5.11 Construction Design in SOMA

In the paper from 2006 (Arsanjani and Allam, 2006) SOMA consists of three phases: identification, specification, and realization. In later work (Arsanjani et al., 2008) two phases are added: implementation and deployment, monitoring, and management. However, these two phases are not discussed in detail in the new paper (they are said to be out of scope). We can conclude from the paper that the following activities in the Realization phase belong to construction design: Refine and detail components, Establish realization decisions, Perform technical feasibility exploration. In our eyes, the contents of the Detail SOA solution stack layer step are not very clearly described.

Also, the construct, generate and assemble services step of the Implementation phase belongs to construction design. During the technical feasibility step prototypes are built that exercise the realization decisions and that have the highest potential impact and risk to the non-functional requirements. In our case we could imagine that an important decision for a Policy Quotation Service is security, since the client has to deliver confidential data to Protector. The result of the Construct, generate and assemble service step is verified in the steps Execute unit test and Execute integration and System test (also belonging to the implementation phase).

6.5.12 Construction Design in SMART

SMART is a technique that "helps organizations analyze their legacy systems to determine whether their functionality, or subsets of it, can reasonably be exposed as services in an SOA". It describes in a large amount of detail which steps are required to analyze legacy systems. Some of the techniques used are code analysis and architecture reconstruction. Let us assume OmniPayment is an application written in an object oriented language. Example results from the legacy analysis are exhibited in Table 6.5.

OmniPayment	
# of services covered	6
Total # of lines of code	13,284
# of classes affected	22
# of lines of code affected	5,392
Level of difficulty	Medium
Level of risk	Low
Use of coding guidelines	Strictly followed
Use of design patterns	Many violations found

Table 6.5: Example results from the legacy analysis in SMART

6.5.13 Implementation in P&H

The authors make a clear distinction between development and deployment. They define deployment as "rolling out new processes to all the participants, including other enterprises, applications and other processes" (a process being a Business Process Execution Language (BPEL) process). The phase is not discussed in detail, so we cannot apply it to our case.

6.5.14 Implementation in SOMA

As we already mentioned in section 6.4 and subsection 6.5.11, the articles we used as sources do not describe the deployment, monitoring, and management phase. Though the concept of service deployment is mentioned, the author do not give any suggestions on how to deal with it. Therefore, we were not able to apply it to our case.

6.6 Conclusions

In this chapter we looked into six methodologies for service-orientation. Two of them cover the whole development process, namely P&H and SOMA. However, they do not describe all phases very thoroughly. Some of the specialized methodologies provide an in-depth contribution to specific steps of the development process, like SMART for the construction design phase. In the previous chapter we have seen that we can apply function design and construction design to the enterprise as a whole as well as to the coarse-grained modules of the enterprise (which are subsystems). The methods investigated in this chapter deal with IT systems only, so they do not take into account human services. The Business Element Approach and the Goal Driven Approach focus on delimiting modules and allocating services to modules. They are similar to the basic idea of BCI-3D: they deal with the construction of the service-oriented system (enterprise) as a whole and identify services for interaction, but they do not focus on constructing the internals of the modules. When applying SOAF, SOMA, as well as P&H to the case, we were left with some questions. All three methodologies prescribe the steps to be executed, but not clearly and completely. In contrast, SMART is very precise in how to perform the service construction phase. Because SMART is the only method we found that deals with the construction of modules in detail, we were unable to compare it thoroughly.

All in all, the GSDP has been very helpful in discussing the coverage of the investigated methodologies, and to elucidate and sharpen the core notions in service-orientation. Next, the application of the methodologies to the same case has shown the differences in the depth in which the activities are described. None of the methodologies combines full coverage with full depth.

Chapter 7

Deriving a Service Specification Framework from the Ψ-theory

Abstract In recent years, the WSDL and UDDI standards arose as ad-hoc standards for the definition of service interfaces and service registries. However, even together these standards do not provide enough basis for a service consumer to get a full understanding of the behavior of a service. In practice this often leads to a serious mismatch between the provider's intent and the consumer's expectations concerning the functionality of the corresponding service. Though additional standards have been proposed, a holistic view of what aspects of a service need to be specified is still lacking. This chapter proposes a service specification framework, which is based on a founded theory, the Ψ-theory. This framework can be applied both for specifying human services, i.e. services executed by human beings, and IT services, i.e. services executed by IT systems.

7.1 Introduction

In service-orientation service providers interact with service consumers by offering them services. In this interaction, both parties have certain expectations of each other's *responsibilities*. Serious problems can occur if these expectations are not made explicit in a service specification. Let us consider an example of a service called 'consult legal expert'. We distinguish between two different kinds of services, 'human services' and 'IT services'. Human services are tasks executed by human beings, whereas IT services are tasks executed by IT systems. The example service 'consult legal expert' is of the first type. The role of service consumer is fulfilled by an employee of a retail company, the role of provider by an legal expert working for a law firm. The employee of the retail company may expect the service to have the following behavior: the expert deals with all his problems for the planned duration of the visit, let's say 30 minutes. The legal expert, however, may regard the consultation as dealing with one specific problem with a maximum duration of 30 minutes. When the legal expert is requested to deal with two unrelated problems and the visit takes 29 minutes, the employee of the retail company may be surprised to receive an invoice for two consultations instead of one. Likewise, an ill-specified IT service can also lead to misunderstandings. Take for example an IT service for calculating a mortgage amount. If we only know the input and output of the service, then we have no idea how soon the results are returned (e.g., within 2 minutes or 3 days). Also, we do not know whether or not this calculation is legally binding when requesting a quotation for a mortgage based on this calculation.

Though the Web Service Definition Language (WSDL) (W3C, 2001) standard enables provider and consumer to have a common view on the interface of the service and the Universal Description Discovery Integration (UDDI) (Clement et al., 2004) can be used as a means for publishing some service information, they together do not provide enough basis to deal with questions regarding for instance the semantics of provided functionality, the semantics of the input and output parameters, the availability of the service, and the costs of calling the service. The original intent of the UDDI was to enable worldwide run-time service discovery, but it even falls short in enabling good design-time discovery.

The recognition of the importance of a comprehensive service specification becomes clear when looking at the efforts of numerous standardization organizations to develop service-related standards. Yet, the other side of the coin is that this development has led to a morbid growth of specification standards. In our eyes it is time to take a step back and focus on *what* one should specify instead on *how* it should be specified. We take a different approach and base our work on a solid theory: the Ψ-theory (Dietz, 2006b).

The main contribution of this chapter is therefore a generic service specification framework based on the service definition given in chapter 4. This definition is based on the Ψ-theory. It recognizes human services as well as IT services. Our generic framework therefore can be used by service providers for specifying both human services and IT services to enable service consumers (i) to find a certain service, (ii) to determine whether the provided functionality corresponds to their needs and (iii) to know how to use a certain service.

The structure of this chapter is as follows. We start by discussing related work in section 7.2. Section 7.3 presents our generic service specification framework and its derivation based on the Ψ-theory. An example case of an insurance company is used to show the application of the generic service specification framework in section 7.4. We conclude the chapter with a discussion of the results and recommendations for further research in section 7.5.

7.2 Related Work

As mentioned in the introduction, the UDDI standard (Clement et al., 2004) is currently most popular in practice as a standard for service registries. This XML-based standard states both what to specify (to some extent) and how to specify it. The UDDI only prescribes a very small set of information that has to be specified. It has possibilities for describing the service function in the T-Models, but these T-Models are unstructured. Therefore there is no consistency across specifications, which makes automated discovery and also manual discovery difficult. Also, in each individual case one again has to think about which aspects to describe in the T-models.

In the web service standards community researchers and practitioners state that the service contract consists of a interface definition (WSDL), a message structure definition (XML Schema), and, if required, a policy definition. These policies specify rules and constraints that must be met by the consumer before it can access the web service. Policies are used to specify aspects of a service that cannot be specified in WSDL or XML schema. These aspects include among others technical limitations, choice of security protocol, privacy constraints, and type of reliable messaging used. These policies do not prescribe what one should specify about a service, but they provide a generic structure for specifying several aspects. WS-Policy (W3C, 2007) is the proposed XML-based standard that allows providers to specify their policies and that allows consumers to specify their policy requirements.

Also, two standards for specifying the Service Level Agreement (SLA) are evolving; WSLA (Keller and Ludwig, 2003) proposed by IBM and WS-agreement (Open Grid Forum, 2007) proposed by the Open Grid Forum (OGF). These standards focus on specifying the agreements made by service consumers and providers and the way to evaluate and measure these agreements. In this sense they have a broader scope than only specifying the service itself. However, they focus mainly on quality aspects like, for instance, performance.

More comprehensive frameworks are the business component specification framework (Ackermann et al., 2002) and the faceted specification approach (Walkerdine et al., 2007). Though the aspects mentioned in these frameworks seem plausible, there is no clear rationale why the frameworks are constructed as they are. For instance, Ackermann et al. (Ackermann et al., 2002) propose seven different levels in their framework. To us it is unclear why we need these seven levels (why not three, five or eight?) and if there is a hierarchical relationship between these levels like the name 'level' seems to imply. Additionally, these standards are not based on a founded theory, therefore it is not clear why certain aspects need to be specified whereas others are not taken into consideration.

Researchers in the area of Artificial Intelligence proposed semantic web service standards like OWL-S (Colasuonno et al., 2006; Luo et al., 2006), WSMO (WSMO, 2007), and WSDL-S (Rajasekaran et al., 2004) for extending the UDDI. The goal of semantic web services is thoroughly specifying every aspect of a service in order to enable automatic matching of supply of and demand for services. It takes a lot of effort (if feasible at all) for a large enterprise to specify everything into so much detail that automatic matching on run-time becomes possible. At this moment none of the semantic web service approaches are popular in industry. Though industrial partners participate in research projects, we see little (if any) semantic service registries in real SOA environments. In practice the matcher of supply and demand is still a human being and not a machine.

All in all, the work from Ackermann et al. and Walkerdine et al. is most closely related to our work. We take a different approach by building our approach on a sound theory enabling us to explain why we need to specify certain aspects. Also, our work has similarities with the work from the semantic web services world. But we do not try to realize automated discovery on a world-wide scale. We aim at providing a framework for practitioners that they can use as a reference for knowing which aspects of the service they need to specify (with a clarification why they need them). They can use different standards to actually specify these aspects. Summarized, we focus on what aspects of a service need to be specified and not how they should be specified.

7.3 Generic Service Specification Framework

Figure 7.1 depicts our generic service specification framework. We base this framework on the generic service definition provided in chapter 4. Our rationale for applying this generic approach is that in our eyes the same aspects need to be specified for services executed by IT systems as for services executed by human beings. Though the aspects themselves are equal for both types of services, it may be the case that the form in which these aspects appear is quite different. We will see some examples in section 7.4.

Now, let us explain why the framework looks like it does. When we recall the service definition, we see that for calling a service basically three things need to be known to the service consumer, namely information on (i) who provides the service (the executor), (ii) which production fact is to be brought about by the executor, and (iii) how to interact with the service executor by executing and dealing with coordination acts. We translate these information needs into three main *aspect groups*, i.e.: *service executor*, *service production* and *service coordination*, respectively. As transactions can have a commercial as well as a non-profit character, we add *contract options* as an additional aspect group; the consumer needs to know what he has to pay for what service.

The *service executor* aspect group defines who is the provider of the service and contains two aspects, namely the *actor role* and *contact information*. The actor role aspect specifies the role of the actor that takes final responsibility for the service. In case of a human service this is the actor role of the human executing the production fact, while in case of an IT service this is the actor role of the human responsible for executing the production act, but who is supported by an IT system. This information can be gained from two types of diagrams provided by the ontological model of the enterprise, namely the Actor Transaction Diagram or the Process Model. It would go far beyond the scope of this chapter to introduce all the Enterprise Ontology models in detail. In the example case given later in this chapter we introduce only the most relevant ones. For further details we refer to the Enterprise Ontology book (see Dietz (2006b)). Since the initiator could feel an urge to contact the service executor, contact information of the executor needs to be provided in the specification framework. We could consider, for instance, situations in which a protocol error arises after calling an IT service and the fault condition denotes to contact the service executor. Also, the initiator may still have some questions about the service after reading its specification.

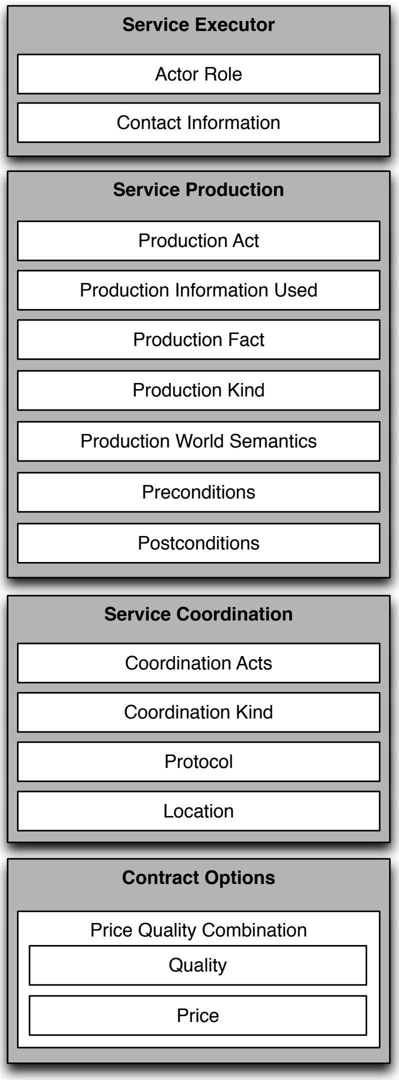


Figure 7.1: The generic service specification framework

The *service production* aspect group focuses on the production act to be performed by the executor. This is the actual value that the service executor offers to the service initiator. It should expose the service properties needed for choosing the right service by a potential service consumer. Unlike service coordination, service production does not concern the communication between the service initiator and the service executor. The aspects which need to be specified are *production act, production information used, production fact, pre- and postconditions, production kind, production world semantics.* The production act is gained from the Actor Transaction Diagram or the Process Model of the ontological model of the enterprise. The information which needs to be used in order to execute the actual production act is described in the Action Model of the ontological model. This model defines for every coordination act which information is required to deal with and therefore specifies the required input parameters. The execution of a production act results in a production fact. This is the actual value requested by the initiator. By having specified the production fact, which is gained from the Transaction Result Table of the ontological model, the result provided by the service has been defined. The Transaction Result Table defines the result type of each transaction and therefore defines the resulting production fact type for the service. Preconditions and postconditions state production facts that should always hold prior to, respectively after the execution of the service. These aspects are required to keep services autonomous. A provider should not have to prescribe to a consumer which other services to call before (or after) calling the provider's service. The provider only specifies the production facts that need to be true before (or after) calling his service. The consumer can choose which other services (if any) to call to meet preconditions. Information about the pre- and postconditions are gained from the Action Model of the ontological model. This model defines among others the operational business rules of an enterprise. Since we distinguish between different kinds of production acts – ontological, infological and datalogical – the production kind aspect defines the kind of service we are specifying. To have a common knowledge and understanding about the semantics of the service to be provided, the production world semantics need to be specified. This information is modeled in the State Model of the ontological model of the enterprise and can therefore be used for specifying the production world semantic aspect.

The *service coordination* aspect group has as goal to give the consumer all information required for realizing successful communication with the provider. As we have seen in section 4.1, the Ψ-theory states that communication between actors takes place by means of the coordination acts in a transaction. We therefore specify the required *coordination acts* for communicating with the executor. Next to this, for completely specifying the service coordination aspect group, we require three aspects that are implementation-dependent. Since the Enterprise Ontology models the essence of an enterprise in a completely implementation-independent way, we can not gain this information from the Enterprise Ontology models. Though these aspects are also not explicitly mentioned in the Ψ-theory, they logically follow when one thinks about how to access a service. First, a service consumer needs to know whether the service is an IT service or a human service, because IT systems and humans communicate in a different way. We call the related aspect *coordination kind*. Second, the consumer has to apply a certain protocol for successful communication. Knowing the location of a service in itself is pretty useless for a service consumer without knowing how he has to offer the service input to and receive the service output from the service provider. Successful communication between the consuming service component and the providing service component is enabled using *protocols*. Protocols define the rules governing the syntax, semantics and synchronization of communication. Typical examples of protocols for IT services include Internet Protocol (IP), Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP) and SOAP. Though often less explicitly defined, human services require protocols too. Take for example the service for ordering food in a five star restaurant versus in a fast-food restaurant. Not only the quality of the product (the food) and the quality of the service itself (the delivery time, the atmosphere etc) differ, but also the way in which the service consumer (the client) and provider (the waiter) interact. In the five star restaurant etiquette play a far more prominent role than in the fast food restaurant. Also, while in the five star restaurant it is protocol to sit down and wait for the waiter to come to you, you have to go to the counter to order in the fast-food restaurant. Finally, a service consumer needs to know the *location* of the service. This location can be either physical or logical. In IT services logical locations are preferred, e.g., a URL or TCP/IP hostname and port number. By using location addresses one can easily change the physical server at which the IT services are hosted. For human services physical locations are more usual. The location of a human service might be, for instance, the second floor room 2.110. Though less common, the location of a human service can be specified as a logical location, e.g., a phone number or an email address. Especially for IT service, the form in which the location is specified is highly dependent on the communication protocol used (Stal, 2006). Please note that these phone numbers and email addresses play a conceptually different role from those specified in the service provider contact information, though they can have the same value. The location specifies where the service consumer can *access* the service, while the service provider contact information specifies where the service consumer can *get information about* the service.

When entering into commitments, e.g. by providing a service to a service requester, the quality and the pricing of the provided service need to be discussed between the providing and the requesting party. The results of such negotiations, or predefined pricing and quality aspects, need to be specified in the specification framework as part of the whole contract. In the specification framework such *price quality combination* aspects are specified in the aspect group called *contract options*. These aspects can not be derived directly from the Ψ-theory, but the negotiation about such aspects can be modeled in separate Enterprise Ontology models. The service contract option aspect group specifies one or several contract options from which service consumers can choose. The contract option aspect consists of a particular *quality* level and the *price* for using the service with this particular quality level. The service executor might define different quality levels in order to anticipate on the various needs and financial positions of different consumers. Example pricing mechanisms are memberships or paying per call.

7.4 Insurance Case Example

In the next paragraphs, we use the life insurance case example introduced in chapter 6 in order to demonstrate how to apply the specification framework. Let us have a look again at the Actor Transaction Diagram (ATD). Figure 7.2 shows a part of the Actor Transaction Diagram. For explaining how we can apply the service specification framework, we have selected the subset of the transactions of the life insurer that is of relevance for handling new individual policies. We distinguish between the following four composite actor roles: potential individual policy holder, individual policy holder, insurer, and reinsurer. In our case, Protector fulfills the role of insurer. The reinsurer insures persons that would cause a high risk for the insurer, for example because the insured amount is high. This means that a part of the insured amount is insured by the reinsurer in order to spread the risk. A reinsurer can be another 'regular' insurance company or an insurance company that is specialized in insuring insurance companies. Sometimes reinsurance is legally obligatory, sometimes it is a choice made by the insurer itself.

The composite insurer actor role can be further decomposed into the following atomic actor roles: product advisor, policy quotator, policy binder, reinsurance premium payer, policy underwriter, and commission payer. This list is not complete as we only mention the roles that are related to handling new individual policies. The product advisor is responsible for providing a potential individual policy holder of advice of which products suit his needs. If the potential policy holder is interested in one or more products, he can request a quotation from the policy quotator. After that the potential policy holder can request policy binding, which makes the policy legally binding. The policy binder requests the policy underwriter to check whether or not the risk is acceptable and if an additional premium fee is required. The risk may be so large that the policy underwriter requests reinsurance. The individual policy holder is responsible for premium payment and for some types of products (e.g. pension insurance) he may make voluntary deposits. The reinsurance premium payer pays the reinsurance premium to the reinsurer.

In the remainder of this section, we apply the generic service specification framework to specify the service 'policy underwriting', which implements the T27 transaction as shown in Figure 7.2, as an example.

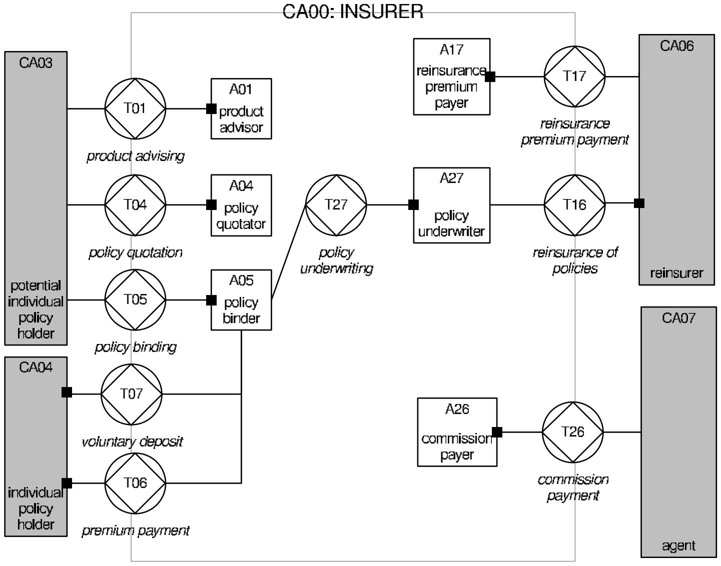


Figure 7.2: Actor Transaction Diagram of Protector

7.4.1 Service Executor

As mentioned in section 7.3, the service executor aspect group specifies the role of the actor that takes final responsibility for the service. So whether it concerns an IT service or a human service, the same actor role remains responsible. The only difference is that in case of an IT service the person fulfilling the actor role is supported by an IT system. As shown in Figure 7.2, the policy underwriter is the actor role executing transaction T27 and therefore responsible for the 'policy underwriting' service. The specification of the service executor aspect group for Protector looks as follows:

> **Actor Role**
> Policy Underwriter
>
> **Contact Information**
> University Street 1A
> 8291 BN Insurancetown
> 555-492022
> underwriting@protectorinsurances.com

7.4.2 Service Production

In the service production aspect group we specify the actual value that the policy underwriter actor role (in this case the policy underwriter) offers to the service initiator (in this case the policy binder actor role). The *production act* in our example concerns policy underwriting, which is part of transaction 'T27 Policy underwriting'. We define this act as follows:

> *Policy underwriting is the act of evaluating the risk and exposures of potential insurants. It involves making the decision whether or not the insurant can get coverage for the insured and what additional premium the insurant has to pay if the insured poses a larger than average risk.*

The *information used* aspect is derived from the Action Model. Table 7.3 shows which information is used in the different acts of 'T27 policy underwriting'. The notation of the process steps in the right column are as follow: Transaction/Process step. E.g., T27/ex denotes the execution of the production act in the transaction 'policy underwriting', and T27/ac denotes the accept coordination act of the transaction 'policy underwriting'.

Object class, fact type, or result type	**Process steps**
POLICY	T27/pm, T27/ex
INSURANT	T27/rq, T27/pm, T27/ex
party *par* is the insurant of policy *pol*	T27/pm, T27/ex
minimal_age	T27/rq
INSURED	T27/pm, T27/ex
person *per* is the insured of policy *pol*	T27/pm, T27/ex
INSURANCE PREMIUM	T27/ac
the payer of premium *pre* is the insurant of policy *pol*	T27/ac
INSURANCE BENEFIT	T27/ac
the beneficiary of insurance benefit *ben* is the beneficiary of policy *pol*	T27/ac

Figure 7.3: Information used in different acts of transaction T27

As shown we need information about the insurant in the request process step. Also, we need to know the minimal age for someone to act as an insurant. During the promise step as well as during the production step the executor needs to get information about the policy, the insured, and the insurant. These data are required to enable him to make the underwriting decision. The service initiator, namely the policy binder, requires information on the insurance premium and insurance benefit to allow him to decide whether or not to accept the production fact.

The *production fact* can be derived from the Transaction Result Table. As we can see in Figure 7.4 the result type of the transaction 'policy underwriting' is 'policy underwriting for policy *pol* has been done'. The *production* *kind* of this service is ‘ontological’, since it is part of the ontological model of Protector. Examples of an infological respectively datalogical service are ‘calculate premium’ and ‘store calculation results’.

Transaction	**Result type**
T01 Product advising	product advice *adv* is created
T04 Policy quotation	policy *pol* is quoted
T05 Policy binding	policy *pol* is bound
T06 Premium payment	the premium for policy *pol* for period *per* is paid
T07 Voluntary deposit	the voluntary deposit for policy *pol* is made
T17 Reinsurance of policies	the policy collection *pco* is reinsured for period *per*
T18 Reinsurance premium payment	the reinsurance premium for policy collection *pco* for period *per* is paid
T26 Commission payment	commission *com* is paid
T27 Policy underwriting	the underwriting for policy *pol* has been done

Figure 7.4: Transaction Result Table of Protector

We model the *production world semantics* in the State Model, which uses an Object Role Modeling (ORM) based notation technique called World Ontology Specification Language (WOSL) (Dietz, 2005). Figure 7.5 exhibits the complete State Model of Protector. Please note: in this model we provide more information than in the UML model depicted in chapter 6. In our service example we only use parts of this model, so we only explain the most important concepts. We use the term *policy* for the individual policy as well as for a participation in a collective contract. An insurance policy has an insurant, one or more insured, and one or more beneficiaries. The insurant is an organization or person that is responsible for the payment of the premium of a policy. The insurant is the client of Protector. The insured is a person who is the ‘insured object’. The beneficiary is a person who receives a payment if the insurant has a right to a benefit according to the product rules of a policy.

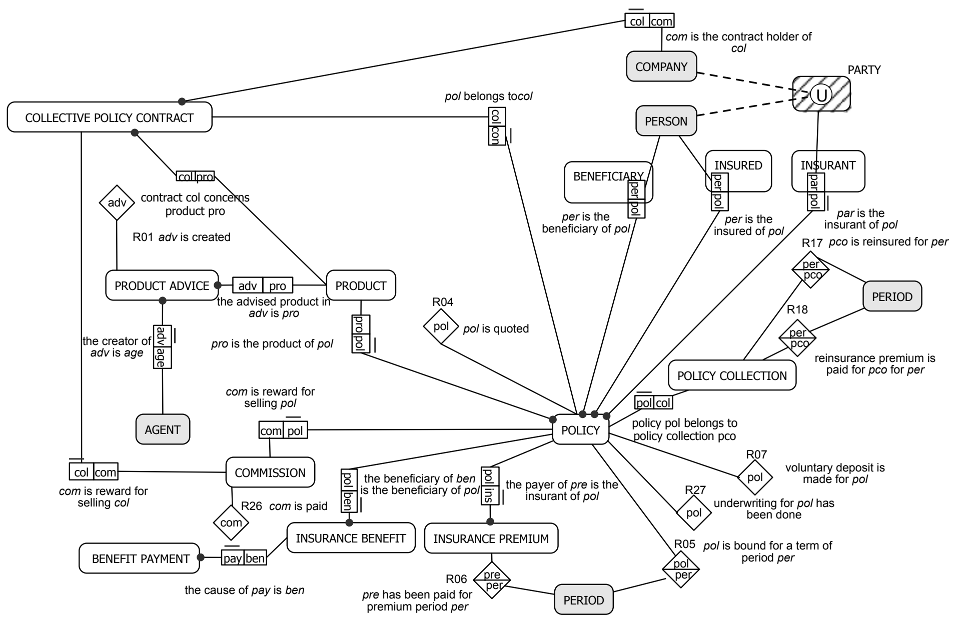

col com
com is the contract holder of col
COMPANY
PARTY
U
PERSON
COLLECTIVE POLICY CONTRACT
pol belongs to col
col pol
BENEFICIARY
per pol
per is the beneficiary of pol
INSURED
per pol
per is the insured of pol
INSURANT
par pol
par is the insurant of pol
col pro
adv
contract col concerns product pro
R01 adv is created
PRODUCT ADVICE
adv pro
the advised product in adv is pro
PRODUCT
R17 pco is reinsured for per
per pco
PERIOD
R18
per pco
reinsurance premium is paid for pco for per
R04
pol
pol is quoted
pro pol
pro is the product of pol
the creator of adv is age
adv age
AGENT
com is reward for selling pol
com pol
POLICY
POLICY COLLECTION
pol col
policy pol belongs to policy collection pco
R07
pol
voluntary deposit is made for pol
col com
com is reward for selling col
COMMISSION
R26 com is paid
com
pol ben
the beneficiary of ben is the beneficiary of pol
pol ins
the payer of pre is the insurant of pol
R27
pol
underwriting for pol has been done
BENEFIT PAYMENT
pay ben
the cause of pay is ben
INSURANCE BENEFIT
INSURANCE PREMIUM
R06
pre per
pre has been paid for premium period per
PERIOD
R05
pol per
pol is bound for a term of period per

Figure 7.5: State Model of Protector

In section 7.3 we stated that *pre-and postconditions* are gained from the Action Model. The action rules in this model are written down in a pseudo-algorithmic language. The following action rule, for example, specifies how the actor decides whether or not he promises to underwrite the policy.

```
on requested T27(insurant)

        if ( type(insurant)=person AND age(insurant)<minimal_age
        ) →
        decline T27(insurant)
        ◇ not ( type(insurant)=person AND age(insurant)<minimal_age
        ) →
        promise T27(insurant)

no
```

As becomes clear from this example the policy underwriter check whether the insurant is a natural person or a company. For companies he always promises to underwrite the policy, for persons only if the person is older than the minimal age (which in this case is 18). So one of the preconditions of the service states that if a person fulfills the role of insurant, he should have a minimum age of 18. Likewise, an example of a postcondition is that the insurance premium always is larger than 0.

7.4.3 Service Coordination

Within the *coordination acts* aspect group, the steps of a transaction which deal with communication between the initiator and the executor need to be defined. In our example, if the initiator calls the service ‘policy underwriting’ he wants to know whether the executor processes his request, or whether the executor may also decline such a request. If the initiator does not receive any notification, such as a promise or a decline, after having sent his request he would be unsure about his request being processed or not. For the specification of the coordination aspect, we therefore use a transaction pattern, which needs to be known and agreed upon by both parties. In section 4.2.2 three patterns have been introduced for describing the coordination between two parties, namely the basic transaction pattern, the standard transaction pattern and the complete transaction pattern. The most used pattern is the standard transaction pattern, which we introduced in Figure 4.1, and which we also used for the specification of the coordination acts in the Protector case.

As explained in the section 7.3, the *coordination kind* states whether we are specifying a human or an IT service. Because protocols and location are implementation-dependent aspects, it makes a large difference for their specification which type of service we are dealing with.

Let us assume ‘policy underwriting’ is a human service. The *protocol* should in that case prescribe how to interact with the human being fulfilling the role of service executor. We could imagine, for instance, the following procedure:

1. send quotation to the policy underwriter by mail
2. receive confirmation from policy underwriter by mail
3. discuss additional questions by telephone with policy underwriter

4. receive final underwriting decision by mail

Following from this protocol, we see that we need to specify two types of *locations* for accessing the service: the postal address and the phone number of the policy underwriter. Keep in mind that though these data could have an overlap with the service executor data, they serve a different purpose. The location aspects specifies data required for the operational use of the service, while the contact details in the service executor aspect group specifies data required for the service initiator to contact the service executor *about* the service. So these data can very well differ from each other.

Now let us think of 'policy underwriting' as an IT service. Example *protocols* for getting data across a network include HTTP or a queuing protocol if guaranteed delivery is required. We can apply the WSDL standard for structuring the messages exchanged. The *location* can be specified as a URL or as an IP address, e.g. 165.34.2.113.

7.4.4 Service Contract Options

In the service contract options aspect group, we specify which quality of service the service initiator gets for which price. Let us assume Protector uses an internal costing calculation system as many large organizations do. Protector distinguishes between two types of quality levels: 'regular' and 'urgent'. The regular service call has a price of EUR 8 and the urgent service call of EUR 12. For both IT services and human services we could specify the accompanying quality aspect as follows:

regular: the maximum response time in 90% of the calls is 5 hours
urgent: the maximum response time in 90% of the calls is 2 hours

Usually, IT services tend to have quicker response times than human service, though this does not have to be the case. Especially asynchronous services can take a long time to finish: multiple weeks, days, or even years.

Besides response time, we could also take into account other quality aspects like security aspects, accuracy of data, availability etc.

7.5 Conclusions

In current literature, a complete framework for specifying services is missing. As a consequence, insufficient information is provided to the service consumer about the behavior of the respective service. In this chapter we addressed the problem of service specification. Moreover, as we have based our research on the rigorous Ψ-theory, we have also improved our understanding of services by relating them to transactions. The function of the specification of a service is to give all stakeholders the information about the service they need, e.g. for service discovery, selection, and usage. Solely by specifying the input and output aspects of a service, as is the current practice, the service consumer does not get sufficient information to determine whether the service fits his needs. These specification aspects only reflect part of the total externally visible behavior of a service. Though many standards exist for specifying certain aspects of a service, a holistic approach is still missing. The main contribution of this chapter is the development of such a holistic framework, which we call the *generic service specification framework*. As an example for demonstrating the feasibility and the usefulness of the generic service specification framework, we analyzed an insurance case. For this case we provided two ontological partial models and discussed how these models can be used to specify all aspects of a service, according to the service specification framework.

Part III
Theory Meets Practice

Chapter 8

Case Study 1: The Port of Rotterdam

Abstract This chapter describes the results of a case study conducted at the Port of Rotterdam, a very large port in Europe. The goal of the case study is to evaluate our Enterprise Ontology-based service specification framework for its practical application. During a Rational Unified Process (RUP)-project at the Port of Rotterdam we specified the required services for the first iterations using this framework. The framework contributed to early error discovery and awareness of important, but often overlooked, service aspects. Overall the service specification framework fulfilled the needs of the project, whereas some findings led to improvements in our framework.

8.1 Introduction

In this chapter we describe our first case study conducted at the Port of Rotterdam in 2008. We sought to evaluate our service specification framework in a real-life situation. This case study contributed to our ultimate goal, i.e. creating a generic service specification framework that is both founded on a sound scientific theory and evaluated in several real-life projects. We position this case study as the evaluation step of the design science research approach (Hevner and March, 2003). In this case study we focused on the use of the service specification framework in a component-based software development project. We were not an observant outsider, but part of the project team working on the software development. Our role in this project was to specify the function of the services including the Web service files

(WSDL's and XSD's) using the service specification framework. Currently, there is no general consensus on *criteria for interpreting case study findings* in IS research. The criteria we apply for interpreting the results are as follows. Firstly, we consider data gathered by direct observation to be most reliable. After that follow the data collected by respectively interviews and participation-observation. Secondly, when a finding can only be supported by opinions of people, we require at least two people having the same opinion to support this finding. Finally, we take into account that the opinions of people may change during the course of time. Since the opinions of the people participating in the project may shift due to new insights during the course of action, we value opinions of people at a later time higher than the opinions of the same people at an earlier time.

The remainder of this chapter is structured as follows. In section 8.2 we describe the background of this case study. After that, we discuss its results in section 8.3. We show how these results lead to improvements in our service specification framework in section 8.4. Section 8.5 presents the conclusions of this chapter.

8.2 Case Study Background

The Port of Rotterdam Authority manages and develops the Rotterdam Port and Industrial Areas. It aims to: (i) promote effective, safe and efficient vessel traffic management, both within the port itself as well as in the coastal area and (ii) develop, construct, maintain and exploit port areas. Vessel traffic management is carried out by the *Harbor Master* Division, currently consisting of about 560 people, led by the Harbor Master. To perform its task, the Harbor Master Division has at its disposal: a central traffic management location, three (in near future two) traffic control locations and 10 patrol vessels. Furthermore, a shore based radar system supplies a complete view of all vessels present in the port area. An extensive portfolio of information systems support the Harbor Master Division in many of its tasks. Due to the advanced age of the current main 'backbone' information system of the Harbor Master Division resulting in costly maintenance and in long lead times for processing change requests, the decision has been taken to develop a new IT-system, named HaMIS (Harbor Master Information System).

Our evaluation took place in the *Rational Unified Process* (RUP) elaboration phase. The goal of the elaboration phase is to mitigate the main risks by testing the global design decisions, modeling the most important use cases, and building functionality of some of the use cases having the highest priority or being technically complicated. Another thing to do in the elaboration phase is setting up the development and test environment and preparing the customer organization for testing, accepting, and using the software. The goal of the service specifications is to allow the developers of different components to work independently during the software development project; all the information they require for the interaction with other components is documented in the service specifications. Very good specifications were required because the two development teams worked for two different IT services providers and on two different locations (Rotterdam and Groningen).

8.3 Case Study Results

Like stated earlier we gathered data during five months through participation-observation, i.e. we specified services ourselves using the framework, through direct observations, i.e. we have looked what 'happened' and through interviewing, i.e. we have evaluated the structure and contents of service specifications with a domain expert, a business analyst, four IT architects, and four software developers. In this evaluation we were looking for the answer to the following question: "To what extent do the aspects of the service specification framework cover the information needs of the project members?". For each aspect we want to indicate whether it was useful as is, whether it was useful in an adapted form, or whether it was considered not applicable to the project. Also, we want to find if the framework is missing any aspects required for service specification. In this section we first give an overview of the situation by presenting some of the identified services (subsection 8.3.1). After that, we discuss the evaluation results for each part of the framework (subsection 8.3.2 to 8.3.5).

8.3.1 Identified Services

Table 8.1 exhibits a selection of the services identified during the first iteration of the RUP elaboration phase (this is not a limitative list). Some of the services are related to searching information (SearchShipVisits, SearchBerthVisits, SearchShipMovements, and SearchInspectionTasks), one is related to extracting information (GetShipVisit), and other services are related to dealing with human tasks (ClaimInspectionTask, CompleteInspectionTask, and ReleaseInspectionTask). In this project DEMO was not used for business modeling. Traceability between IT services and business processes was realized by specifying a relation between an IT service and a use case. This use case is related to a business process.

Service name	**Service description**
SearchShipVisits	Search for ship visits using several input parameters
SearchBerthVisits	Search for berth visits using several input parameters
SearchShipMovements	Search for ship movements using several input parameters
GetShipVisit	Read the complete ship visit tree
SearchInspectionTasks	Search for inspection tasks, either assigned to the current user, available tasks, or, assigned to other users
ClaimInspectionTask	Assign an inspection task to current user or other user (depending on authorization level)
CompleteInspectionTask	Register the results of a ship visit inspection task
ReleaseInspectionTask	Undo the assignment of a certain inspection task

Table 8.1: Services identified during first iteration.

8.3.2 Service Executor Part Evaluation

The service executor part (see Table 8.2) was not particularly useful in the project, because only one service provider existed at that moment. It would make little sense to specify the actor role or name of this person in all service specifications. The architects argued that service provider information will be required in a later phase of the project when the number of services grows. When this occurs, they would indeed prefer to specify a certain role or department instead of a person, since a person can work part time, become sick etc.

Service Executor Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Actor Role	During the case study period, it was unnecessary to specify the service provider information explicitly because there was only one person fulfilling the role of service provider.	x		
Contact Information	The same finding holds for the service provider contact details as for the actor role.	x		

Table 8.2: Evaluation of the Service Executor Aspects

8.3.3 Service Production Part Evaluation

The project members all agreed that the service production part covered the aspects of the externally visible behavior of the service necessary for this project. Table 8.3 exhibits the results per aspect of the service production part.

Let us have a closer look at the SearchShipVisits service as an example. Its production act is defined as follows: ‘to provide a list of ship visits conforming to given search criteria. It returns all attributes of the ship visit, all attributes of the ship belonging to the ship visit, a selection of the attributes of the agent representing the ship visit, and zero or one ship movement identifiers’. Its production fact is ‘searching for ship visits *vis* has been done’, where *vis* is the set of ship visits conforming to the criteria of the production information used. The production kind of this service is ‘infological’.

Table 8.3: Evaluation of the Service Production Aspects

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Production Act	This aspect is used for giving a summary of what the service does. Its use is to get a quick picture of the behavior of a service without having to read all the details.			x
Production Information Used	We tried two ways of specifying the production information used (input), viz. specifying them using UML class diagrams and in graphical representations of the XML schema trees. It turns out that architects prefer the first way of representation because they are only interested in what information is exchanged and not in the precise structure of the XML messages. Developers prefer the schema trees, or, the textual representation of the XML schemas. We used additional descriptions in natural language to specify conditions on the input parameters, e.g. if parameter a is empty, then parameter b must be filled.			x
Production Fact	The same findings holds for the production fact as for the production information used.			x

Table 8.3: (continued)

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Production Kind	Because in the project DEMO was not used as a methodology, a distinction between ontological, infological, and datalogical services was not made. Another service classification was made, some example types were search services, read services, and task services. However, it was not (yet) required in this project because the number of services was still quite limited. Searching for potential reuse will be important when the number of services grows.	x		
Production World Semantics	To prevent inconsistency in the message usage for interaction with different services, we have designed a data model that specifies all possible data elements used in the interaction with the services. The production information used and the production fact of a services refer to data elements in this data model. We guarantee compliance between the messages and the data model by only allowing the service messages being specified in terms of *restrictions* on the data model.			x
Preconditions	We specified the preconditions in natural language. A discussion arose on how to deal with preconditions and postconditions. The architect of the providing component expected the consuming component to check them, and the architect of the consuming component expected an additional check by the providing component.			x

Table 8.3: (continued)

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Postconditions	The same findings holds for the postconditions as for the preconditions.			x

Table 8.3: Evaluation of the Service Production Aspects

The XML Schema and WSDL standards are insufficient for specifying the production information used (input) and production fact (output) parameters. For instance, a requirement is that wildcards can only be used at the end of a search term, e.g. 'Ship = HMS Beag*', and not in the center, e.g. 'Ship = HMS*le'. The asterisk is a placeholder for zero or more characters. To make sure queries do not take too much time not all queries are allowed. Also, some input data needs to be conditional, e.g. if input item 'Berth' is filled than also input item 'StartDateTime' and 'EndDateTime' should be filled. Because the XML Schema standard lacks the means to specify these details, we need some additional input and output descriptions in the service catalog.

Table 8.4 shows the explanation of three of the terms from the terminology aspect in the service production part. A class diagram (UML) was used to depict the relations between the concepts. Regarding semantics there was a discussion on the term 'tonnage' (of a ship). Its value is not always expressed in thousands of kilograms, like the name seems to imply. Sometimes the value is expressed in units of 1016 kg, 1.1 cubic meter, or 2.8 cubic meters depending on the type of ship and on which maritime organization delivers the data. A business analyst proposed to "just use an integer", because the end user has enough domain knowledge what the value means in a specific context. It would not lead to problems because the only consuming component is the GUI. However, it can lead to serious errors when another consuming components start making calculations with the tonnage value having its own assumptions on its semantics.

This service has no pre- or postconditions, but some of the other services did. In the service catalog the pre- and postconditions were specified

Ship visit	a stay of a ship in a certain geographical area defined by the harbor master.
Ship	a water vessel, including water airplane, hydrofoil, hovercraft, rig, production platform, dredger, floating crane, elevator, pantoon, and every floating tool, object and installation.
Port	An area for receiving ships and transferring cargo. It is usually found at the edge of an ocean, sea, river, or lake.

Table 8.4: The production world semantics aspect of the service production part

in natural language, since the software engineers are not familiar with more formal approaches like UML OCL, Z, or Rule-ML. Despite the lack of precision of expressions in natural language, both the software engineer of the providing component and the software engineer of the consuming component have the same understanding of the semantics of the expressions themselves. However, an architect of the providing component and an architect of the consuming component started a discussion about how to deal with these pre- and postconditions. The first architect argued that in a Design by Contract$^{\text{TM}}$ the caller is completely responsible for checking the preconditions. Not fulfilling them leads to undefined behavior (the service may or may not carry out its intended work). The second voted against taking this approach. He opted for a double check at the service provider side, making sure an unfulfilled precondition always results in an error message returned by the service provider. His motivation was that an undefined output can jeopardize the functioning of the complete HaMIS system. Though he sees the double work for implementing condition checking at both sides, he sees undefined output as an unacceptable risk. So it is not only important to specify the pre- and postconditions, but also whether or not unfilled pre- or postconditions result into an error message or undefined output.

8.3.4 Service Coordination Part Evaluation

Table 8.5 shows the results per aspect of the service coordination part. From the interviews we found that the aspects of the service coordination part sufficiently addressed the information needs of the architects and software developers the service consumer and service provider. The other project members did not need the service coordination part as they were only interested in the production part of the service.

Table 8.5: Evaluation of the Service Coordination Aspects

Service Coordination Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Coordination Acts	Specifying the error situations, i.e. decline coordination acts, was regarded as one of the most important aspects of the service specification. The provider and consumer need to create an understanding of what can go wrong. When getting a certain error message the consumer needs to know whether it is useful to call the service again with the same input parameters, whether he needs to change his input parameters, or whether there is nothing he can do about it (in that case he needs to contact the service provider). In the case study we did not deal with cancellation acts.		x	
Coordination Kind	Since all services in scope were IT services, this aspect was not applicable.	x		

Table 8.5: (continued)

Service Coordination Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Protocol	For accessing the service we used the WSDL standard. A web service can be built using RPC or document style binding. A few members of the project team, viz. two architects, one developer, two integration specialists, and the service specifier chose to apply document literal style web services for their message validation capabilities, the possibility to define XML schemas externally (outside the service interface description) and their WS-I compliance (conformation to standards) (Akram et al., 2006). Though a drawback may be a lesser performance, we thought the benefits outweigh this drawback. Because all services used a document style binding we did not need to make this information explicit in the individual service specifications. In our project we used two types of transport layers: HTTP (for synchronous service calls) and JMS (for asynchronous service calls). Though the difference could be seen in the binding part of the WSDL, we also specified this in the service specification for making this information available to other stakeholders than developers.			x
Location	In this project specifying only one location for a service was deemed insufficient. Instead, we needed a Development, Test, Acceptance, and Production location.		x	

Table 8.5: Evaluation of the Service Coordination Aspects

Error code	Error name	Cause	Error message
F00000001	NoSearchElement	None of the input search elements are filled [unfulfilled condition on input].	"The service requires at least one search element as input"
F00000002	CharNotAllowed	A string search element contains characters that are not allowed, e.g. a wildcard in the middle of the string or Russian characters	"The name input element contains a character that is not allowed"
F00000003	UserNotAuthorized	A task is being assigned to a service that has no authorization to perform this task.	"The user is not authorized to perform this task"

Table 8.6: The errors codes of the coordination acts aspect of the service coordination part

In automated systems, error situations lead to decline and reject coordination acts. Table 8.6 shows examples of the errors that can occur when calling the service.

The SearchShipVisits service used SOAP and HTTP (as transport layer) as protocols, because for searching for information we did not need asynchronous communication or guaranteed message delivery. For the service ClaimInspectionTask, for instance, we would need guaranteed message delivery and we would choose Java Message Service (JMS) as a transport layer.

The architects and software developers proposed a change to the service location aspect. In our original service specification template 'location' referred to just one location, in the project we needed to specify multiple locations. In software development projects in general a clear distinction is made between the following type of physically separated environments: Development, Test, Acceptance, and Production. They each have their own purpose within the software development process. Something new or a change to existing software should be developed in the Development environment. It should be tested in the Test environment by the project members. After that it should be tested by a selected group of end users in the Acceptance environment. If this group of end users accepts the software, it can be propagated to the Production environment in which the software is used by the actual

Environment	**URL**
Development	exampleserver1.portofrotterdam.com:5050/ShipVisit-v1/Service?wsdl
Test	exampleserver1.portofrotterdam.com:5060/ShipVisit-v1/Service?wsdl
Acceptance	exampleserver2.portofrotterdam.com:80/ShipVisit-v1/Service?wsdl
Production	exampleserver3.portofrotterdam.com:80/ShipVisit-v1/Service?wsdl

Table 8.7: The location aspect of the service coordination part

end users. When tests fail in either the Test or Acceptance environment, the software is demoted back to the Development environment and the process restarts. Since the services play a role in all these environments, the locations for all these environments need to be specified. Table 8.7 shows an example specification for the different environments.

8.3.5 Service Contract Option Part Evaluation

Table 8.8 shows the evaluation of the service contract option part and Table 8.9 shows some of the QoS constraints to which the service conforms. Only the quality of a service was specified and not its price, since there was no mechanism for charging the use of services.

8.3.6 Overall Evaluation

All in all, the architects and software engineers from the providing as well as the consuming party expressed their enthusiasm about the service specifications. The software engineers saved time because both parties agreed upon the external behavior in an early stage. This enabled both parties to work in parallel; the consumer used stubs of the actual service. Replacing the stub by the actual service led to no or very few problems. When problems did occur, it was always immediately clear by looking at the service specification which party has caused the problem(s).

According to the interviewees not only the framework itself contributed to the prevention of errors and early error discovery, but also the structured specification process and the separate role of the ‘neutral’ service specifier. By neutral we mean that the service specifier does not work on the design or implementation of either the providing or the consuming components. Be-

Service Contract Option Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Price Quality Combination	QoS constraints are an essential part of the service specification in the HaMIS project. We have used the Extended ISO model (van Zeist and Hendriks, 1996), an extension to the ISO 9126 model, as a basis for specifying the QoS constraints. Not all elements of this model are relevant, e.g. the usability element only applies to user interfaces and not to services. For this reason we have made a selection of elements of this model that need to be specified. Time behavior, availability, and security elements were regarded as the most important QoS constraints.		x	

Table 8.8: Evaluation of the Contract Option Aspects

Characteristic	**Sub characteristic**	**Constraint**
Efficiency	Time behavior	The maximum response time in 90% of the calls is 1,75 seconds.
Efficiency	Resource behavior	The service can be called 50 times a minute.

Table 8.9: Quality part of price quality combination aspect of the contract options part

cause of this we prevent component-specific constructs in the specification ('shortcuts') for making the implementation easier.

We have seen that we can very well use the framework for determining what aspects of a service need to be specified. However, when specifying IT services it is insufficient to only specify information that we derive for the ontological model of the enterprise. We need to add many additional details such as information that is only relevant for processes at the infological or datalogical level, such as data required for making calculation or data required for making checklists and letters. Like we explained in chapter 7 our service specification framework is a generic one that can also be used for specifying human services. Some details are less relevant for human services. For instance, a human can easily interpret 1 pond (Dutch measurement for mass) as 0.5 kilograms, so one does not have to specify this fully in advance in this case. We can use the framework for determining *what* aspects to specify, but we need additional standards to determine *how* to specify the information (in detail).

Our framework was lacking two aspects that were needed in this case study. The first aspect was the possibility to specify multiple locations of a service. If an enterprise develops its own software, either with or without the assistance of an external IT service provider, it in general requires four types of locations; the development, test, acceptance, and production environment. The first environment is used for developing the software. This environment is used by software developers. The second environment is used for testing the software (e.g. unit tests, integration tests, and stress tests) by software developers themselves and testers. Some enterprises make an additional distinction between the test environment for developers and for testers. The acceptance environment is required for user acceptance testing the software. Only end users are allowed to access this environment. Lastly, the production environment is used for actually using the software. End users are allowed to use this environment and only software administrator can access it to solve problems. When an enterprise buys its software (i.e. COTS), it usually only requires an acceptance and production environment.

Secondly, we require a version aspect in our framework. Because during a software development project services change quite often, a good service versioning mechanism is crucial. DEMO does not give us any help on how to deal with versioning. For this project we have applied the backwards compatibility strategy as mentioned by Erl et al. (2008) to the data model as well as the messages. This is a well-known versioning strategy, also used before the days of service-orientation. This results into the following type of version numbers: “x.y”, in which x represents a major version number and y a minor version number. The terms major and minor relate to compatibility with previous versions, for instance: version 5.3 is compatible with version 5.1, but not with version 4.8. This was considered to be sufficient.

8.4 Improvements to the Service Specification Framework

As said in the previous section we need to add two things to our framework: the possibility to specify multiple locations of our framework and the possibility to specify versioning information. We have to change the 'location' aspect to 'locations'. We do not want to add the specifics about the development, test, acceptance, and production environment, because the types of location can differ per enterprise and these locations are not relevant for human services. When adding these location types to our framework we would be jeopardizing the genericity of the framework. We have to add the versioning aspect to the contract options part. Before using the service, the consumers want to be sure that the version of the service is the one he expects. Though we also recognize the concept of versioning in human services, the term is not often used in this context. A new version of a human service can, for instance, be changing the protocol in a restaurant (e.g. wearing a tie), delivering faster response times (e.g. package delivery in one day instead of one week), or introducing a new pricing structure.

8.5 Conclusions

In chapter 7 we presented a generic service specification framework that is derived from the Ψ-theory, a theory that regards an enterprise as a purposefully designed and engineered social system. To demonstrate the framework we applied it to an insurance company case. Though this insurance company case is based on a real world company, we then did not apply the framework in practice at this company. So we only *demonstrated* the framework and did not *evaluate* it yet. In this chapter, we sought to evaluate our service specification framework in a real-life case study at the Port of Rotterdam. This case study contributed to our ultimate goal, i.e. creating a generic service specification framework that is both founded on a sound scientific theory and evaluated in several real-life projects. We position this case study as the evaluation step of the design science research approach (Hevner and March, 2003). In this case study we focused on the use of the service specification framework in a component-based software development project.

The main contributions of this framework in the HaMIS project consisted of early error discovery and awareness of important service aspects that are often overlooked. These errors mainly included semantic errors and conditions to the input messages that cannot be specified using XML schemas and WSDL. The QoS constraints aspect in the service specification framework made people aware that this is also externally visible behavior of a service. It lead to negotiations about for instance response time, availability, and security between service provider and consumer. Also, people became aware that it is not only important to specify the 'happy scenario', but to also take into account the specification of different types of error situations or exception handling.

Overall the service specification framework fulfilled the needs of the HaMIS project, though some findings require improvements in our framework. These findings include the following: (i) the need for specification of multiple service locations, and (ii) the need for a versioning aspect. Another thing we found is that for IT services plain XML schemas are insufficient as a standard for specifying the production information used (input) and production fact (output) of a service as they cannot specify conditional input and output parameters. Next to this, it is important to make explicit how to deal with unfulfilled preconditions.

Chapter 9

Case Study 2: De Lage Landen

Abstract This chapter presents a case study conducted at De Lage Landen. Data gathering took place in the months October and November 2009 by means of documents, semi-structured interviews, a workshop supported by group decision software, and questionnaires. The goal of the case study is twofold; on the one hand we derive criteria for the modularization of service-oriented systems, on the other hand we evaluate our service specification framework. We asked the participants to give both the criteria and the service specification framework aspects an importance rating. We found that criteria that base the delimitation of the modules on organizational structures got relatively low scores, e.g. 1.1 and 1.6. The criteria 'maximum cohesion and minimal coupling' (8.5), 'no functional overlap' (9.3), 'process clusters' (8.7) were valued the most by the participants.

In the evaluation of the service specification framework most of the specification aspects got a rating of 7.4 or higher on a scale from 1 to 10. The ones that scored lower were 'production kind' (6.4), 'coordination kind' (6.6), and 'price quality combination' (6.6). An interesting finding emerged from the interviews and questionnaires. We presumed that software systems are mainly production-centric. It turned out De Lage Landen did value the aspect of coordination acts, especially for services that reach beyond the boundaries of the organization itself. Currently they have a lot of manual work in order to correct the state of information systems after a cancellation or to ask for the status of a process because promise acts are not explicit and therefore interpreted in different ways. Nevertheless, they hardly specify or implement these coordination acts (e.g. promise or state messages) in services at the moment because they do not have a clear vision on how to do it.

9.1 Introduction

The second case study has been performed at De Lage Landen. De Lage Landen is a global provider of leasing, business and consumer finance solutions, including vendor finance and factoring (De Lage Landen, 2009). Its main goal is to help its customers grow market share, enhance profitability and achieve strategic goals by offering them asset-based financing programs. De Lage Landen is a fully-owned subsidiary of the Rabobank Group. The company operates internationally and its Dutch office is based in Eindhoven. In 2008 De Lage Landen reported a net profit of €235 million.

De Lage Landen has been working on SOA for a couple of years and it is currently further introducing SOA governance within the organization. Rens Voogt, Team Lead, Enterprise Integration Services (EIS), expects this research to give a valuable insight into how different people within De Lage Landen think about the delimitation of autonomous environments and into what aspects of a service should be specified. The latter question is especially of interest to De Lage Landen as they are currently looking into a migration of the specification of services in various different Word templates to a service registry/repository.

The remainder of this chapter is structured as follows. Section 9.2 presents the results of the data gathering from documents. Next, section 9.3 provides the results of the interview phase. The workshop, as discussed in section 9.4, forms the last part of the data gathering for this case study. Finally, we conclude in section 9.5 with a summary of the findings of this case study.

9.2 Document Data Gathering

From the Lage Landen we received the following documents:

- Integration Reference Architecture 1.1 - Marker Report, Date: Thursday, January 24, 2008
- A Single Business Service Catalog Shall Be Implemented, Date: not provided

The first document contains a set of principles, the second document describes what information of a service should be specified. In the following two subsections we discuss the contents of these documents.

9.2.1 Integration Reference Architecture

The integration architecture explicitly mentions SOA as one of the basic concepts for integration. It defines SOA as a distributed computing concept in which software functions are standardized in such a way that they can be shared by dissimilar applications and technologies as services, both inside and outside the company. According to De Lage Landen, there are four other classifications that constitute the concept of SOA. First, within an SOA a service exposes functionality through an abstract interface hiding the inner complexity and implementation details. Second, an SOA provides business agility by increasing reusability on a macro level; services can be composed, in a loosely coupled fashion, into higher-level workflows. Third, *autonomous environments* acting in an SOA as consumer and/or provider of services become collaborative while independent (we will define the notion of autonomous environment later). Fourth, communication between consumer and provider in an SOA is in general bidirectional. These classifications also conform to our vision on SOA provided in previous chapters; the autonomous environments are modules of service-oriented systems containing only IT services.

De Lage Landen introduces the notion of a *business service*, defined as follows.

Definition 4. *A business service offers functionality of an autonomous environment dedicated to support a certain well demarcated business function and has the following properties:*

- *clearly defined input, output, business functionality and error handling defined in a contract (scheme, specification of service levels, interface etc. in machine readable or documented format)*
- *a business service is capable to act on an individual request once and once only*
- *service levels are clearly defined; availability hours, response times, costs per transaction, incident handling procedure are minimum requirements*
- *based on the message interface*
- *can be invoked through common communication protocols, providing interoperability and location transparency*
- *defined with explicit interfaces, independent of service implementations*

- *accesses business data, or facilitates a change in business data, from one valid and consistent state to another*

In chapter 7, in which we defined the notion of service, we deliberately did not use the term 'business service' as it often leads to confusion. In general, business people use this term for commercial propositions of the enterprise, while IT people mean automated functions that directly support the business process. In the definition above the latter is meant. This is more or less synonymous to our notion of IT service as defined in chapter 7. As a difference we see that the Ψ-theory does not elaborate the notion of business functions, but only transactions and business processes. We clearly recognize the concept of an interface (described in different wording multiple times), the hiding of the implementation, and the agreements about Quality-of-Service.

As stated earlier the notion of an autonomous environment fulfills an important role at the Lage Landen. The notion is very similar to a component in Component-Based Development and modularity in general; the wish to decompose a complex environment into separate major building blocks that can be changed (developed, replaced, reallocated) independently (in terms of speed, technology) without requiring changes in other areas. These environment are called 'autonomous', because they are implemented using separate databases, using separate application server instances, and using separate servers.

Let us have a closer look at how De Lage Landen defines the notion of an autonomous environment;

Definition 5. *An autonomous environment is a clearly outlined and coherent set of related IT functions that is offered using a determined collection of IT-components and a determined set of corresponding system management activities. It has the following properties:*

- *one service owner is responsible for all functionality within the environment and fulfilling agreements with other environments and the outside world in regard to generic services provided and used*
- *the environment has clearly defined interfaces*
- *the environment is location independent and must be transferable from one location to another*
- *the environment moves to a next release as a whole*

- *a single team is responsible for the design and coordination on development activities in the autonomous environment*
- *there is only one production instance of an environment in the company*
- *the environment interacts with other environments by loosely coupled mechanisms*
- *the environment supports one or more generic services*
- *the data is in the environment is kept private, encapsulated behind business logic, read and write access is controlled by business logic*
- *the environment does not trust outsiders, it inspects requests, validates fields, authenticates and possibly authorizes requests*
- *database transactions (2PC) are only used within the environment and not across environment boundaries*
- *the environment implements a policy to cope with conflicts and inconsistency of incoming data related to the same objects entering the environment via different interfaces (e.g. an incoming message and user input via the user interface update the same customer data).*

An important property is that interaction across autonomous environments shall be done in a loosely coupled fashion. In the document 'Integration Reference Architecture' loose coupling refers to restricting the knowledge that requester and provider need to know about each other. Furthermore De Lage Landen provides the following definition on coupling:

Definition 6. *Any aspect that is coupled between requester and provider means that if change occurs it will imply a ripple of change across both parties; changes to the aspect by one party requires corresponding changes by other parties involved in interaction. Decoupling means that changes can be done in consumer (i.e. requester), provider or intermediating infrastructure with no subsequent consequences to other parties; changes to the aspect by one party never require corresponding changes by other parties involved in interaction. Loosely coupled means achieving decoupling for the maximum number of interaction aspects.*

9.2.2 Business Service Catalog

The Lage Landen specifies the different aspects for specification of business services in the document "A single business service catalog shall be implemented". According to this document, service specifications should provide a 'black box' view of a service: internals of the service are not relevant for the catalog nor potential consumers. This is in accordance with our view on service specifications as presented in chapter 7. For each version of a service, i.e. when a change to a service implicates changed behavior in interactions, there needs to be a separate description. Next to describing the services themselves, De Lage Landen wants the business service catalog to provide insight into (i) the relation of composite services using atomic services (please note: this difference is not visible to consumers of the composite service!) and (ii) functional dependencies between consumers and providers. In our view these elements should not be part of a service catalog. The first aspect deals with the internal design of the composite service and the second aspect deals with the contracts between consumers and providers instead of the behavior of the service itself. Table 9.1 depicts the mandatory and optional content of a business service specification.

9.3 Interview Results

We scheduled the following interviews at the Lage Landen:

- 12 October 2009, 9.30 - 11.30, Arjan Jansen, Domain Architect Credit Services
- 12 October 2009, 12.30 - 14.30, Tomi Santic, Application Consultant
- 15 October 2009, 9.00 - 11.00, Elice van den Bogaert, Designer/Domain Architect Consumer Finance and Dirk van Gameren, Software Developer
- 15 October 2009, 11.00 - 13.00, Wido van Eersel, Enterprise Architect
- 15 October 2009, 13.30 - 15.30, Martijn Schiedon, Enterprise Architect

In the following subsections we describe the main findings from these interviews. These subsections mirror the structure of the interviews (see Appendix B for the questionnaire). First, we discuss the rationale for and status of the

Aspect	**Description**
	Mandatory
Id	This aspect is a unique identifier of the service.
Descriptive Name	This verb/noun/specifics construction describes the behavior of the service in a few words.
Version	No explanation given
Category	The category can either be 'generic' (business service generally reusable, no consumer specifics, SOA level) or 'specific' (business service has elements specific to a consumer, EAI level).
Purpose of the Service	This describes the functional why.
Functional Description	This describes the functional what, including preconditions and postconditions.
Technical Description	This describes the technical aspects such as: correlation, naming, headers, exception handling (error messages, fall back, compensation), call sequences, language, character set and addresses.
Owner	The owner is the job position that should be regarded business owner of the service.
Manager	The manager is the job position that should be regarded IT owner of the service.
Status	Values of the status can be: development, production, migration (no longer used for new developments, current consumers are to migrate to new version) or retired (no longer used).
Expiration Date	This date indicates when support on the service ends.
References to technical artifacts required by (potential) consumers	No explanation given
Available Test Facilities	This can be, for instance, XML samples or test scripts.
Provider	This can be an autonomous environment or an external party.
	Optional
Service Levels	This can be, for instance, availability windows, performance limits, or scale limitations.
Support Contacts	No explanation given
Charging	No explanation given
Frequently Asked Questions	No explanation given
Links to Data Dictionary	No explanation given

Table 9.1: Specification aspects for services at De Lage Landen

SOA implementation (9.3.1). Next, we discuss the criteria for the modularization of service-oriented systems that we found in conducting the interviews (9.3.2). Finally, we present the preliminary results of the evaluation of the service specification framework (9.3.3). These results will be further expanded when we elaborate on the importance of the specification aspects in section 9.4.

9.3.1 Context of SOA Implementation

As early as 1995 De Lage Landen started working on applying application integration in a structured way. They introduced a file-based bus using protocols like FTP, SMTP, X400, and SMB to prevent point-to-point coupling. Also, De Lage Landen has experience with COM+ of Microsoft and Online Transaction Processors (2 Phase Commit). More recently De Lage Landen introduced Microsoft BizTalk as an Enterprise Service Bus (ESB). The concept of a service was first introduced in a Business-to-Business (B2B) initiative called Smart Universal Services Interface (SUSI), 5 to 6 years prior to this case study and in 2006 the department Enterprise Integration Services was founded. This department deals with enterprise-wide application integration. In 2007 an important application called FREO that uses Web service technology was taken in production. The department Independent Credit Services (ICS) started building their first services in 2006/2007. Using these credit services De Lage Landen can receive information like credit reports from external parties. Also, Straight Through Processing (STP), which is processing without human intervention, is applied.

The Chief Information Officer (CIO) of De Lage Landen supports SOA and is very focused on limiting the number of exceptions in the architecture process. These exceptions are raised when designers/developers in a project do not conform to the architectural principles. However, many of the decisions are made in ‘small changes’. For these small changes no architect is involved. This results in a gap between the architects and the designers/developers. Business people within De Lage Landen mainly see SOA as a technical approach and prefer not to get involved. None of the interviewees says that De Lage Landen has reached full maturity, though some other companies (financials and consultancy firms) have told De Lage Landen that they are relatively mature regarding the implementation of SOA in the Dutch market.

Figure 9.1 shows the different motivations for applying SOA at De Lage Landen. Let us have a look whether or not these goals are met. Some of

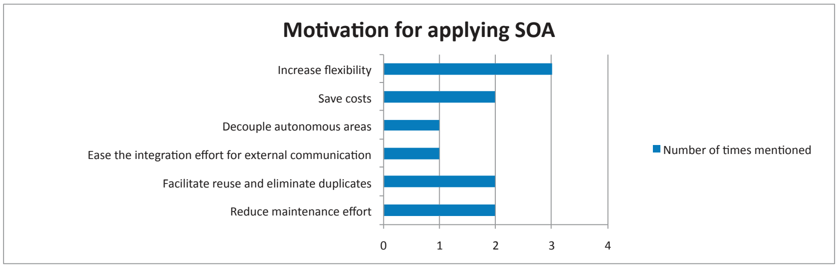


Figure 9.1: Motivation for applying SOA

the interviewees indicate that flexibility has increased by realizing a plug-and-play approach to some extent. By higher levels of standardization it is easier to roll out new functionality to different countries. Processes can be adapted quicker by using orchestration (compared to changing process logic hidden inside applications). Software maintenance (and in fact also development) became easier due to decoupling. For instance, in a project some problems with the front-end occurred. Because front-end and back-end were separated through a layer of services, the back-end team could go on with development without having to wait till the front-end problems were solved. Also, by making services autonomous it was easier to trace problems. Some services are also consumed by the Rabobank, e.g. the Credit Check Service and the Document Generation Service. The latter is a service for generating documents in a uniform way. This service led to direct cost savings. Also, the concept of SOA and the application of an Enterprise Service Bus provides insight into the usage of IT, e.g. the amount of traffic between consumers and providers and who uses what data. One of the interviewees did not think that SOA contributed (yet). He said the implementations were still too small-scale to see any real effects.

Of course the other interviewees also had some critical notes. One interviewee questioned the future performance of the services as the use of XML can cause more latency. But this has yet to be seen. Another note that was mentioned several times is that the business departments do not embrace the concept of SOA yet. According to the IT department they should see themselves as suppliers of business services. In practice, they were not always able to clearly define their semantics in data dictionaries. Also, they tended to think too much in terms of existing applications, e.g. “when contract identifier < 4, then the data originates from system x”. Furthermore, at his moment the granularity of the services is not uniform, which makes it difficult to really achieve the plug-and-play concept. Also, when the financial crisis of 2008 hit, it became hard to get budget for things that are only profitable in the long term. Another funding issue is posed because funding is usually project-based. Because of this the necessary steps towards SOA can only be taken in projects that realize new business requirements. Finally, for designing and changing services communication between many different parties is required. This takes a lot of time and the process is not really structured yet. Services tend to be fine-grained and too specific to a certain purpose. De Lage Landen is working on making services more generic; it is better to send more information and let the consumer decide which information he actually needs.

9.3.2 Criteria for Modularization and Service Design

The interviewees proposed several criteria for modularization. These principles are: “Maximum Cohesion and Minimum Coupling”, “Life Cycle Decoupling”, “Strategy and Technology Agnosticism”, “Value for Consumer and Commercial User”, and “No Single Orchestration”. We discuss the meaning of these criteria in the next section as we gained additional data in the workshop phase.

9.3.3 Service Specification Framework Evaluation

Tables 9.2, 9.3, 9.4, and 9.5 depict the evaluation comments of the interviewees regarding the specification aspects. There was little variation in the answers of the interviewees; they all gave more or less the same answers. The one thing that was mentioned as lacking in the framework was information about the life cycle (e.g. ‘in production’ or ‘in test’) and versioning (e.g. ‘version 2.3’) of the service.

Service Executor Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Actor Role			x	
	De Lage Landen does not specify the actor role, but only the actual owner. A distinction is made between a functional and technical owner. In practice, it is not always clear who the owner is or should be.			
	The functional owner is among others responsible for getting functional requirements from users, setting the price and the funding of the service. The technical owner is, for instance, responsible for the middleware.			
Contact Information				x
	End users only contact the service desk for problems they encountered. The service desk routes problems to the right service owner. So only the service desk would need this information for immediately problems. For inquiry purposes this information would also be required by developers, architects etc.			
	Contact information of every employee can be found in the business directory, so for internal owners this would not need to be specified separately in a service registry. For external owners the information is explicitly written down.			

Table 9.2: Evaluation of the Service Executor Aspects

Table 9.3: Evaluation of the Service Production Aspects

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Production Act	This is sometimes, but not always specified. It would be useful to have this for having a quick overview of what the service does in search screens in the service repository.			x
	At De Lage Landen, a consumer often needs to read the complete specification to get this overview.			
Production Information Used	The input of a service is specified (including complex constraints) in functional designs. This is sufficient according to the interviewees.			x
Production Fact	The output of a service is specified (including complex constraints) in functional designs. This seems to be sufficient. However, some of the participants would like to see a more formal and consistent way of specifying this information.			x
Production Kind	This is not specified as the organization does not use the notion of Enterprise Ontology nor does the organization propose any other service taxonomy. The number of services is not large enough at the moment to need such a taxonomy. The participants do see a need for it (for searching purposes) when the number of services grows.	x		

Table 9.3: (continued)

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Production World Semantics	The business departments at De Lage Landen use Excel-sheets for specifying semantics. The interviewees would prefer to not only specify the definitions of objects, but also their relationship. However, the more important and immediate problem is that business people have difficulties in making precise and unambiguous definitions.			x
Preconditions	Preconditions are written down informally and in different locations of specification documents. This seems to be sufficient to most of the interviewees. However, some of the participants would like to see a more formal and consistent way of specifying this information.			x
Postconditions	Postconditions are written down informally and in different locations of specification documents. This seems to be sufficient to most of the interviewees. However, some of the participants would like to see a more formal and consistent way of specifying this information.			x

Table 9.3: Evaluation of the Service Production Aspects

Service Coordination Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Coordination Acts	This is not specified, but according to the interviewees it would be useful to specify it.			x
Coordination Kind	Only IT services are specified in detail.	x		
Protocol	Protocols are usually not specified explicitly as they are almost always SOAP over HTTP for internal communication and SOAP over MQ for external communication.		x	
Location	The location is specified on an intranet website. Here all servers in the service development phase can be found (development, test, acceptance, and production).		x	

Table 9.4: Evaluation of the Service Coordination Aspects

Table 9.5: Evaluation of the Contract Option Aspects

Service Contract Option Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Price Quality Combination	For external services the price is usually documented.		x	
	Internally it is not always a wish to specify prices for commercial and political reasons.			

Table 9.5: (continued)

Service Contract Option Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
	Quality aspects are specified very basically. This is not always sufficient; sometimes consumer and providers get different expectations in this area due to lack of specification.			

Table 9.5: Evaluation of the Contract Option Aspects

9.4 Workshop Results

Our workshop took place on 21 October, 2009. As described in chapter 1 the goal of this workshop is to let interviewees respond to each other's ideas and to prioritize the criteria (for module identification) they propose. We chose to use *Group Decision Software* instead of a regular flip over for two reasons. First, we want to prevent the 'biggest mouth wins' effect. People who are less talkative will probably be more likely to express their opinion when they don't have to attract attention to say something, but they can simply type in what they want to say using a computer system. Second, we want to enable people to express ideas that may be controversial by guaranteeing their anonymity.

Table 9.6 shows the criteria for delimiting autonomous environments proposed in the workshop. Figure 9.2 shows the importance rating the participants gave to each criterion by providing the average rating and the standard deviation. During the workshop it was not our intention to find anti-criteria (criteria that should NOT be followed). Nevertheless, we did encounter some criteria that got a very low rating from the participants, sometimes even also of the participant who originally proposed the principle.

As shown in appendix C our initial workshop design comprised two parts. The first part of the workshop took more time than expected. For practical reasons (follow-up appointments of participants) we decided to get the input for the second part in a different way; we sent questionnaires to the partic-

Criterion short name	**Criterion long name**
Maximum Cohesion and Minimum Coupling	Autonomous environments have maximum cohesion and minimum coupling.
Knowledge Domain	Autonomous environments reflect the structure of knowledge domains, i.e. sets of actors having more or less the same knowledge.
Business Disciplines	Autonomous environments reflect the structure of business disciplines, i.e. sets of actors working on the same business function.
No Functional Overlap	Autonomous environments do not have functional overlap.
Responsibilities	An autonomous environment falls under the responsibility of one business owner.
Process Clusters	An autonomous environment reflects the structure of autonomous process clusters, i.e. an autonomous environment delivers those services required by a certain (cluster of) business process(es).
COTS	An autonomous environment should be available on the market as a COTS system.
Life Cycle Decoupling	It should be possible to deliver services independently of the financial products in which they are used, i.e. services should not be tightly coupled to the product as a whole.
Strategy and Technology Agnosticism	The division of autonomous environments is independent of strategy and technology.
Budgeting	An autonomous environment reflects the budgeting structure of the enterprise.
Supporting Team	An autonomous environment reflects the organization structure of the software development and maintenance teams.
Long Term Strategy	The division into autonomous environments is based on the long term strategy of the enterprise.
Business Objects	Autonomous environments are built around business objects (base administrations).
Value for Consumer and Commercial Usage	The service has an added value for its consumers.
No Single Orchestration	If services always have to be called in the same order, then the service need to be offered to the consumer at a higher level of aggregation, i.e. the set of services should in that case be offered as a composite service.

Table 9.6: Criteria proposed by workshop participants

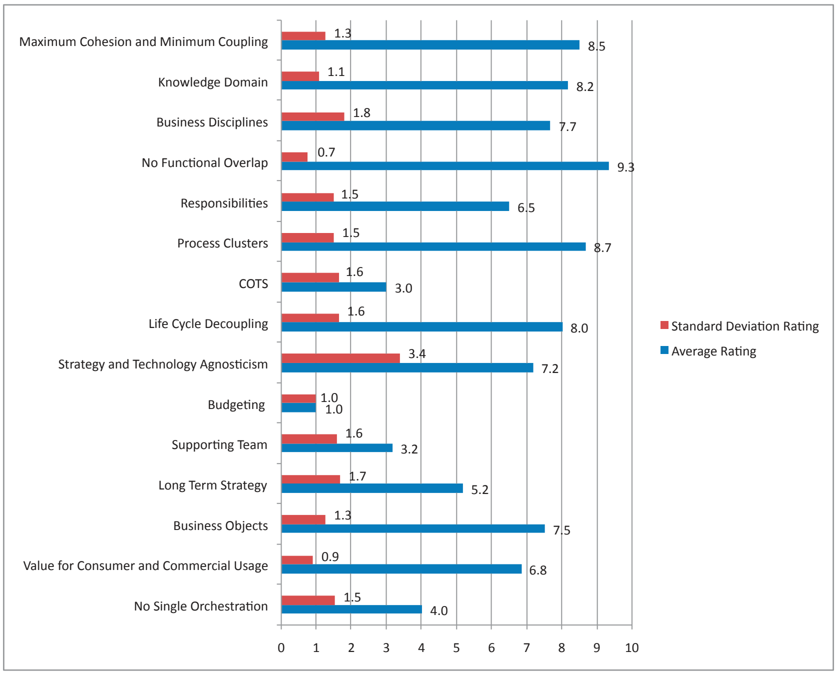


Figure 9.2: Rating of the proposed criteria

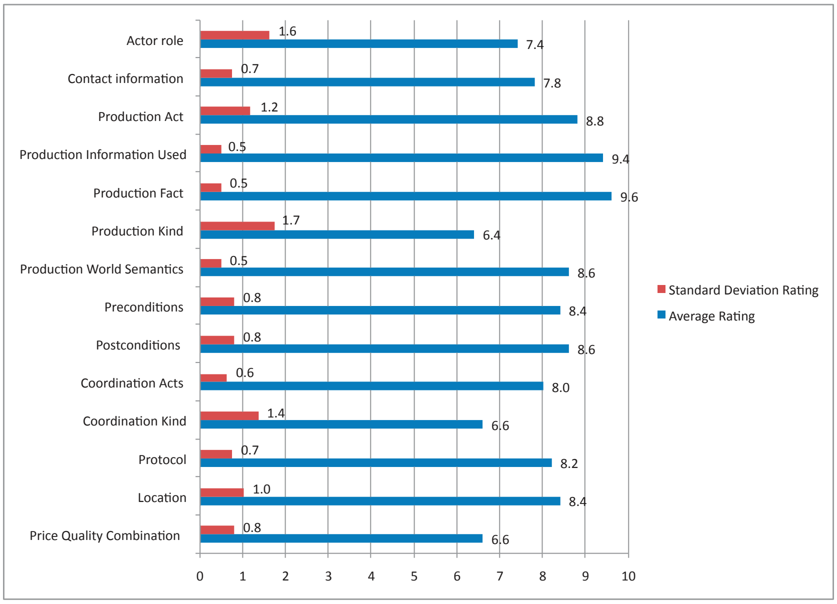


Figure 9.3: Rating of the service specification aspects

ipants to ask them about the rating of different aspects. Figure 9.3 depicts the importance rating participants gave to the different specification aspects. Unfortunately one of the participants did not return the form, so we only have 5 replies instead of 6. The average ratings per person did not differ very much (standard deviation of 0.5).

9.5 Conclusions

In this chapter we discussed a case study conducted at De Lage Landen. We found fifteen criteria for modularization of service-oriented systems. Seven of these criteria got a rating of 7.5 or higher, viz. ‘maximum cohesion and minimal coupling’ (8.5), ‘knowledge domain’ (8.2), ‘business disciplines’ (7.7), ‘no functional overlap’ (9.3), ‘process clusters’ (8.7), ‘life cycle decoupling’ (8.0), and ‘business objects’ (7.5). An interesting finding is that the participants of the workshop did *not* give criteria that base the modularity of service-oriented systems on organizational structures a high rating. For instance, the criterion ‘budgeting’ (basing modules on budgeting structures in the organization) got a score of 1.1 and the criterion ‘responsibilities’ (delimiting modules based on the responsibilities of a business owner and basing these responsibilities on the organizational structure) got a score of 1.6. Nevertheless, these criteria are in reality applied very often at De Lage Landen.

In this case study we also validated our service specification framework. The participants of De Lage Landen gave positive feedback about the service specification framework. Most of the information they required did fit in this framework. In fact, the only things that were really lacking was information about the life cycle and the version of the service. Another comment was that we should make a distinction between a functional and a technical owner and therefore specify two different actor roles. One aspect from the business service catalog that is not available in our service specification framework is the FAQ. These are Frequently Asked Questions about the services. This information is gathered during the time the service is actually used by the participants.

Chapter 10

Case Study 3: Air France/KLM

Abstract This chapter presents a case study conducted at Air France/KLM between August and November 2010. Data is gathered by document analysis, interviews, and a workshop supported by group decision software. Like the previous case study its goal is twofold. We used the case study to derive criteria for the modularization of service-oriented systems. Next to this, we evaluated our service specification framework. The participants gave an importance rating to the criteria for modularization as well as to the aspects of the service specification framework.

We found that Air France/KLM applies the notion of modularity in her business architecture by defining business domains. Business domains can be decomposed into subdomains. The smallest subdomains are called business areas. Interaction among domains and among business areas is realized by automated or manual business services. The highest rated criteria for the definition of domains are: (i) the delivery of results that are meaningful to the environment (7,6), (ii) a high internal cohesion (6.8), and (iii) a uniform definition of the syntax and semantics of information objects within a domain (6.0). In 2007 Air France/KLM started a large-scale SOA program. Originally this program started out with technical activities, namely the implementation of an Enterprise Service Bus (ESB). Later Air France/KLM also included activities for integrating this new technology in her Enterprise Architecture. This new technology enables Air France/KLM to build IT services. Business services that are automated and conform to several other criteria are called SOA software services. The highest rated criteria for the definition of services are: (i) the scope of the service is aligned with business responsibilities (8.8), (ii) the description is sufficient for use (8.0), and (iii) the service is independent of existing systems (7.6).

Air France/KLM has a service registry that provides an overview of all available IT services. Information about the services is mainly stored in two documents: the Business Service Description (BSD) and Software Service Description (SSD). The first document specifies information that is relevant from a business point of view, while the second document focuses on information relevant from a technical point of view. In the workshop we validated our own framework. Aspects of the service specification with a rating higher than 7.0 are: production information used (9.0), production fact (8.8), coordination acts (7.6), price quality combination (7.6), preconditions (7.2), and production act (7.2). The lowest rated aspects, having a rating of 6.0 or lower, are: production kind (5.0), coordination kind (4.2), and protocol (6.0). The participants proposed the following additional aspects that are currently lacking in the framework: versioning information, life cycle information, and an indicator whether the service has a request/reply or pub/sub interaction style.

10.1 Introduction

The third case study is conducted at Air France/KLM, a worldwide airline company based in France and The Netherlands. KLM was founded on 7 October 1919 and has Amsterdam Airport Schiphol as its home base. In 2004 KLM merged with Air France. In our case study we focused on the former KLM company, i.e. the part of Air France/KLM based in The Netherlands. In fiscal year 2009/2010 the KLM Group, which also includes KLM Cityhopper, transavia.com and Martinair, transported 22.5 million passengers and more than 540,000 tons of Air France and KLM cargo. The fleet comprises 205 aircrafts. In 2009/2010 the KLM Group employed more than 31,000 staff (KLM, 2010).

The KLM applies DEMO as one of the methodologies for business modeling and many of the employed architects are DEMO professionals. Also, KLM has multiple years of SOA experience in a joint Air France/KLM SOA program. For our case study we selected two parts of Air France/KLM for data gathering: Air France/KLM Cargo and Air France/KLM Ground Services. Cargo deals with the transportation and handling of cargo and Ground Services deals with ground handling of Air France/KLM flights and flights of partners. Activities of Ground services include check-in, baggage handling, de-icing, and refueling and cleaning planes. We selected these two parts of the organization for they have both DEMO experience and service identification experience.

We performed data gathering in three phases. First, we analyzed several documents of the SOA program. This analysis is described in section 10.2. Second, we interviewed business architects, technical architects, enterprise architects, and the service repository manager. The interview results are described in section 10.3. Third, we performed a workshop in which we gathered and evaluated principles for module and service identification and the importance of different aspects of the generic service specification framework. We describe the workshop results in section 10.4. We conclude this chapter in section 10.5 by summarizing the most important findings of this case study.

10.2 Document Data Gathering

Air France/KLM provided the following documents:

1. SOA Business Services - Identification Techniques V2.1b, Date: June, 2010
2. Overall Domain Structure, Ground Services, Common AF/KL V2.0, Date: May 2010
3. SOA Business Services Concepts Definition V2.4, Date: August 2010
4. Software Services Definition V2.4, Date: August 2010
5. BSD - Business Service Description V2.0, Date: not provided
6. SSD - Software Service Description, Date: not provided
7. SOA Repository Roles & Service Lifecycle, Date: not provided
8. SOA Repository Demonstration, Date: not provided
9. SOA introduction for KL CCC, Date: Spring 2010
10. Required Service States and its implementation, Request for Change for Service States in the AFKL SOA Repository, Date: not provided

The first two documents describe principles for domain and/or service identification. Documents three and four explain the notions of domain, business service, and software service. Documents five and six are the templates for respectively business and software service specifications. The *business service description* (BSD) template is intended for describing the aspects of a service that are relevant from a business point of view. The *software service description* (SSD) is intended for describing the aspects of a service that are relevant from a software engineering point of view. The last documents describe how the service registry is used at Air France/KLM.

10.2.1 Identification of Services Using Business Models

Air France/KLM defines the notion of an *SOA business service* as a business service for which the interaction between consumer and producer can be automated. Additionally, an SOA business service has the property that the interaction between consumer and provider is limited to a single interaction to avoid deep coupling. The documents describe three techniques for the identification of SOA business services: (i) identification from domaining, (ii) identification from business process modeling, and (iii) identification from functional detailed design. These techniques differ in the completeness of the service specification (i.e. how much information about the service is available) and the exhaustivity of the service identification (the amount of services that are identified). Which service identification technique is used depends on the project phase. Figure 10.1 shows graphically which technique Air France/KLM uses in which project phase. This is not a picture with formal semantics. What it intends to depicts is the following. In the first phase of a project (pre-study) Air France/KLM starts identifying services using domaining. Later in this phase identification from process modeling is used. This technique is also the main technique used in the feasibility study phase. At the end of the feasibility study phase the functional detailed design technique is applied. This technique is also used in the last two phases: architecture & specification and design & realization.

Let us first have a look at identification from domaining. This technique starts with the identification of a *business domain* or *domain* for short. Air France/KLM assigns the following characteristics to a domain:

- to a domain corresponds a specific mission, strategy and policy (internal much cohesion, external loose coupling)

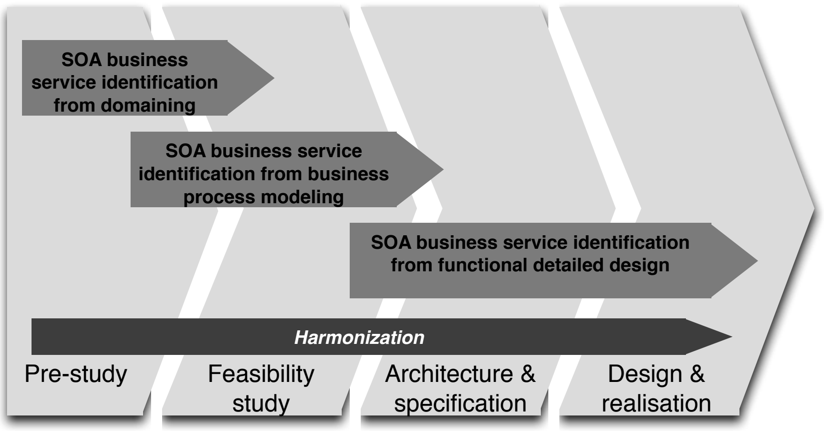

SOA business service identification from domaining
SOA business service identification from business process modeling
SOA business service identification from functional detailed design
Harmonization
Pre-study
Feasibility study
Architecture & specification
Design & realisation

Figure 10.1: Applicable techniques per project phase

- each domain contains a set of business processes contributing to its mission
- the decomposition of domains does not depend on organizational structures
- for a given type of business (e.g. airline, retail bank) a domain decomposition could be shared by several enterprises
- domains can be decomposed into multiple layers of subdomains and the smallest identified domain is called *business area*

Air France/KLM defines a *business service* as an added-value that contributes to fulfill part of the mission of the enterprise and that is provided by a business area to other business areas or that is reusable inside the same area. A distinction is made between the *black-box view* of a business service that only exposes its behavior (aimed at consumers) and the *white-box view* of a business services that describes its construction (exposes the internal actions). The business service is used by a *consumer area*, i.e. the business area that benefits from the result/output of the business service. The business service is offered by a *provider area*, i.e. the area which produces the business service. The business areas interacting through services can reside inside the same domain or in different domains. Also, an interaction between a domain (or actually a business area within a domain) and an external actor

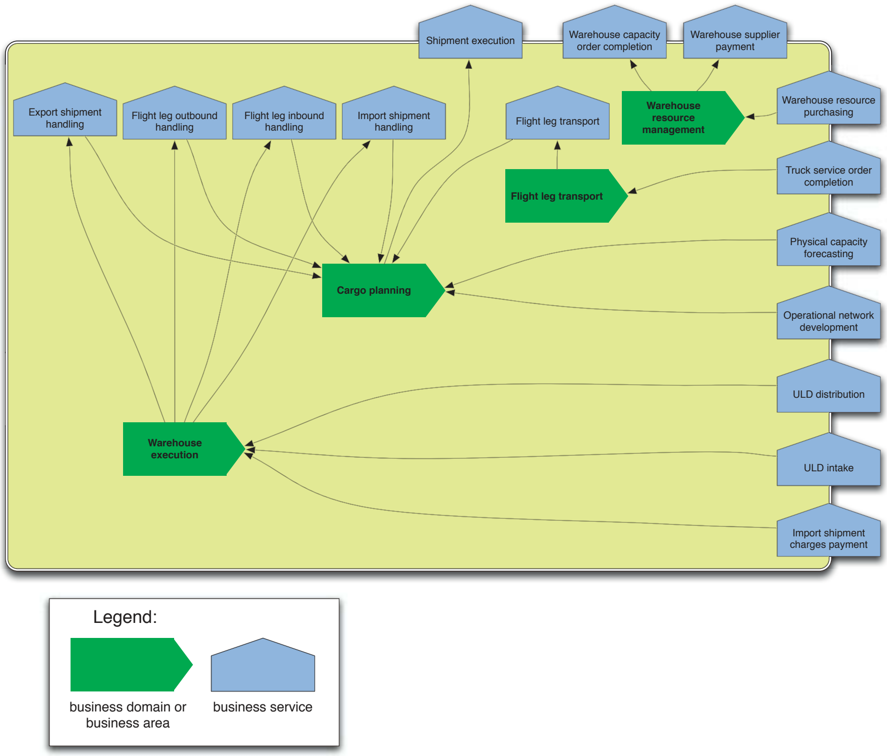

Shipment execution
Warehouse capacity order completion
Warehouse supplier payment
Export shipment handling
Flight leg outbound handling
Flight leg inbound handling
Import shipment handling
Flight leg transport
Warehouse resource management
Warehouse resource purchasing
Flight leg transport
Truck service order completion
Physical capacity forecasting
Cargo planning
Operational network development
ULD distribution
Warehouse execution
ULD intake
Import shipment charges payment
Legend:
business domain or business area
business service

Figure 10.2: Example of domain: operations

is defined as a business service. Figure 10.2 shows the operations domain as an example. As stated before, one of the characteristics of a domain is that it may be further decomposed into subdomains.

The second technique, identification from business process modeling, identifies SOA business services using the ARIS notation (the notation that is supported by the tool Air France/KLM uses). A *business process* is defined in this context as a structured chain of actions which is designed to produce a specific output with added-value for a particular customer or higher level process. Figure 10.3 depicts the Air France/KLM processes pyramid; a decomposition of business processes. A business process which has an interface with the current analyzed process is candidate to be provider of an SOA business service if (i) this process belongs to another domain and (ii) exchange of events with the current process takes place via IT systems. These services are called *event report type* SOA business services.

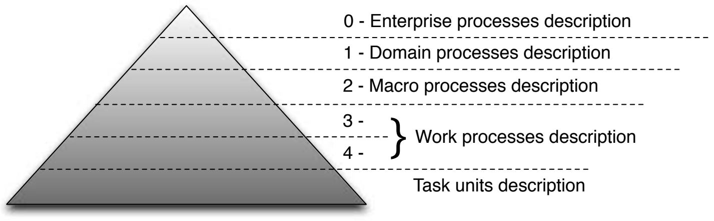

0 - Enterprise processes description
1 - Domain processes description
2 - Macro processes description
3 -
4 -
Work processes description
Task units description

Figure 10.3: Air France/KLM process pyramid

Also, process models can be used to define *request/reply type* of SOA Business Services. Task units (lowest level of process pyramid depicted in Figure 10.3) can be described in dedicated models. These are called *ARIS Function Allocation Diagrams* (FADs). These FADs describe among others the actors, used documents, associated risks, and read and/or updated business information. The criterion for determining whether or not a task unit uses a business service is the usage of external business information. Next to this, task units used in more than one work process are SOA business service candidates at the condition that: (i) they are fully automated and (ii) it is validated that the task unit is candidate to be (re)used by other business areas.

The last technique for identifying SOA business services is identification from functional detailed models. These services are on a lower granularity level than the services identified from the previously described technique. For this technique task steps within task unit are identified as SOA business services if the following conditions apply: (i) they are fully automated and (ii) it is validated that the task unit is candidate to be consumed by other business areas. These conditions are the same conditions as in the previous technique. The only difference is the granularity level of the services.

10.2.2 Terminology Related to Automated Services

Figure 10.4 provides an overview of the terminology used at Air France/KLM in relation to automated services. As we see a business area can offer *Business Software Services* (BSS). A BSS is defined as the software service implementing an SOA business service. A distinction is made between two types of BSS: those that executed after a request from a consumer and those that are executed after a notification from a publish/subscribe mechanism. In the latter case the party creating the notification does not require a response from the provider. An SOA business service may have one of these implementations or

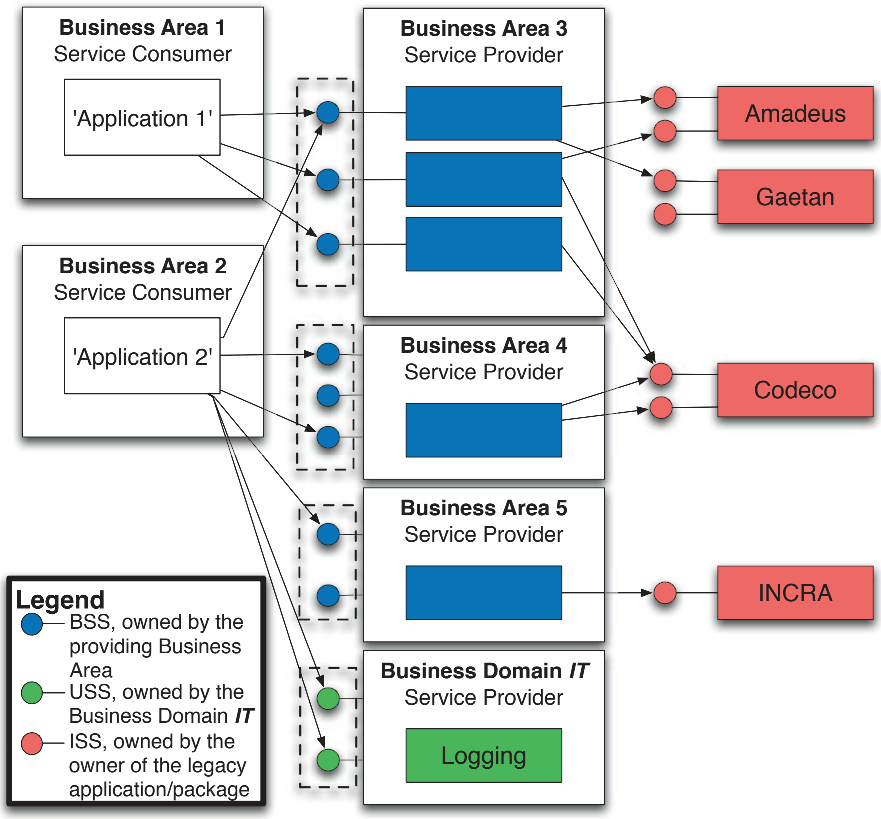


Figure 10.4: Different service types at KLM

both. *Internal Software Services* (ISS) are software services privately exposed by an area. The need for internal software services can come from legacy applications reuse, COTS, or IT optimization needs. *Utility Software Services* (USS) expose technical functionality (e.g. archiving, logging). These are BSS offered by the domain IT.

10.2.3 Specification of Services

Air France/KLM uses a service repository for storing information about services. This is a catalog used for adding, editing, and organizing services. The services are organized in a hierarchical way using the domain, subdomain, and business area decomposition. Another function of the repository is to

Aspects of business service
Name
Description
ARIS-Database
Business Process
Business Area
Owner

Table 10.1: SOA business service aspects stored in service repository

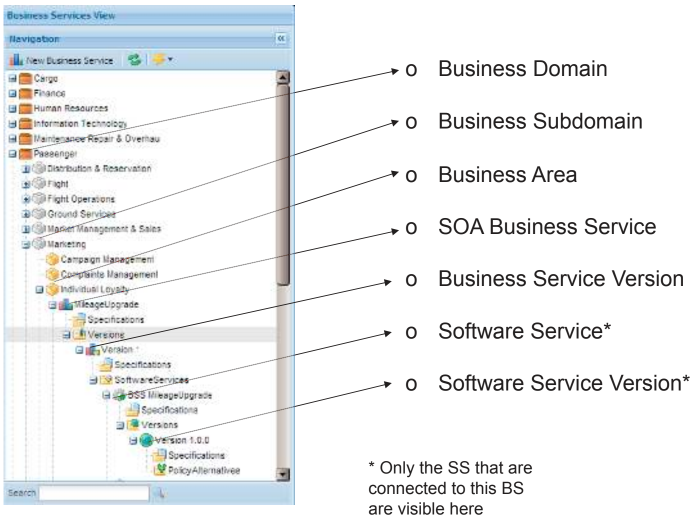
Aspects of software service
Name
Business Area
Application
Business Service
Owner
Name
Identifier
Visibility
External
Transmission (event or request/reply)
Technical Type
Type (Business/Internal/Utility)
Connector
Functionality
Securitic (Air France only)
URL

Table 10.2: Software service aspects stored in service repository

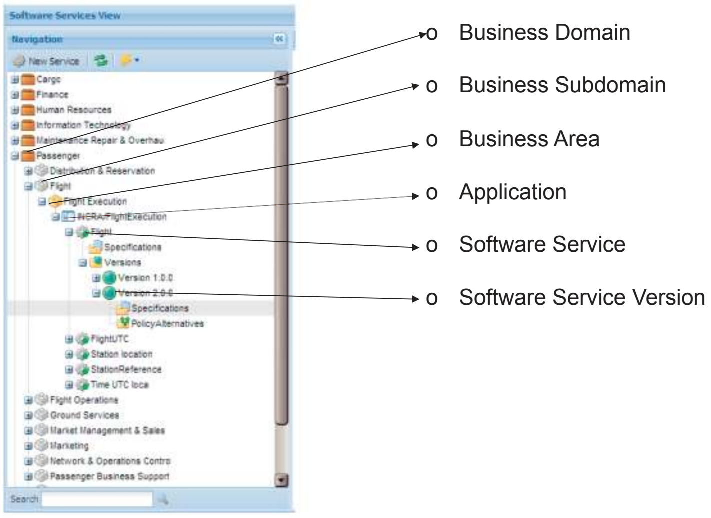
keep track of versioning information of services and different document versions. The latest version of a service specification is always available to the consumers and older versions can be made available if required. Thus it is a single point of information for services. Figure 10.5(a) shows a screenshot of the organization of SOA business services in the service repository. As we can see in the picture the relationship between business domains, business subdomains, and business areas is graphically represented by hierarchical folders. Figure 10.5(b) shows a screenshot of the organization of software services.

Table 10.1 shows the aspects of an SOA business service specified in the repository as structured searchable fields. The ARIS-Database is a link to the business models in the business modeling tool. Specification documents can be attached and a link to an *SOA business service version* can be made. A certain SOA business service version can be linked to a software service. Table 10.2 shows the aspects of a software service that are specified in the repository as structured searchable fields. Figure 10.6 shows a screenshot of a software service in the service repository. Again, specification documents can be attached. A link to a *software service version* can be made.

As stated before the repository contains links to documents that contain the detailed specifications of services. Air France/KLM specifies services using two different templates: the *SOA Business Service Description* (BSD) and the

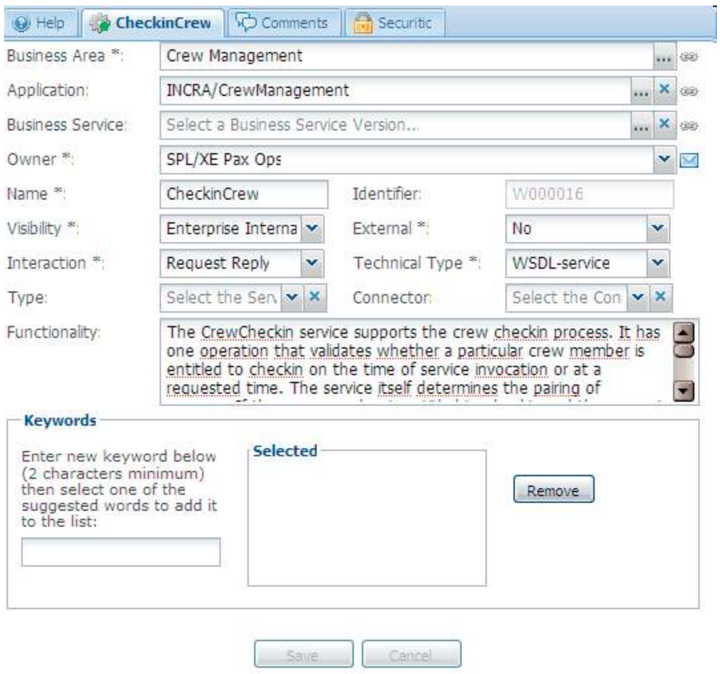


(a) Organization of SOA business services



(b) Organization of software services

Figure 10.5: Screenshots of organization of SOA business services and software services



Business Area *:	Crew Management		
Application:	INCRA/CrewManagement		
Business Service:	Select a Business Service Version...		
Owner *:	SPL/XE Pax Ops		
Name *:	CheckinCrew	Identifier:	W000016
Visibility *:	Enterprise Interna	External *:	No
Interaction *:	Request Reply	Technical Type *:	WSDL-service
Type:	Select the Serv	Connector:	Select the Con
Functionality:	The CrewCheckin service supports the crew checkin process. It has one operation that validates whether a particular crew member is entitled to checkin on the time of service invocation or at a requested time. The service itself determines the pairing of		

Keywords

Enter new keyword below (2 characters minimum) then select one of the suggested words to add it to the list:

Selected

Figure 10.6: A software service in the service repository

Software Service Description (SSD). The first template specifies the aspects of a service relevant from a business point of view. The second template specifies the aspects of a service relevant from a technical point of view. Table 10.3 shows the document template for SOA business services. Table 10.4 shows the document template for software services.

Table 10.3: Specification aspects for SOA business services at Air France/KLM

Aspect	**Description**
	Document information
Name of the SOA business service	The name of the service, naming rules for services are available
Version number	The version number consists of two digits "x.y". The x is called the release number and is updated by the business enterprise architect who is accountable for the Business Service Description (BSD). The y is updated by the business analyst who is responsible for the BSD.
Status	The status of the specification, e.g. "Draft" or "Final"
Date of last update	The date of the last update of the specification
Document location	The location of the specification
Domain/area	The business area that is responsible for the SOA business service
IMO accountable	The name of the provider of the description of the SOA business service
Author	The author of the specification document specified by his name, function, and department code
Approvals	The persons who have given approval for the different versions of the specification
Business enterprise architect	The architect accountable for the SOA business service
Revision history	An overview of the changes made to the service
Distribution	A list of people to whom the specification is sent
Linked documents	A list of documents related to the specification document
	Overview
Goal	The objective fulfilled with this service in main lines
Result(s)	A textual description of the result, this should be in terms of added value to the consumer and part of the mission of the owning domain
Interaction type	Indicator whether the service is a request/reply services or an event report, sometimes both types are applicable to a service
Precondition(s)	All the predicates that should be met in order to be able to execute the SOA business service successfully

Table 10.3: (continued)

Aspect	**Description**
Postcondition(s)	The condition(s) of the environment that should be met after the SOA business service is executed
Invocation	
Request	If the service is a request/reply service, this aspect needs to be specified. This is done as a table in which the business object, its description, whether its mandatory or optional, and remarks are specified.
Triggers	If the service is an event-based service, this aspect needs to be specified. This is done as a table in which the trigger name, a description of the trigger, and remarks are specified.
Other information	
Result	A complete description of the result, this is a table including the business object, its description, an indicator whether it is mandatory or optional, and remarks
Process steps	The process steps performed by the SOA business service
Business rules	The business rules that are relevant for consumers of the SOA business service
Business exceptions	The behavior of the SOA business service when an exception occurs (also called business fault)
Policies	The Quality-of-Service aspects of the SOA business service, this includes business criticality, business volume, business use, confidentiality, integrity, availability, and accountability

Table 10.3: Specification aspects for SOA business services at Air France/KLM

Table 10.4: Specification aspects for software services at Air France/KLM

Aspect	**Description**
Document information	
Name of the software service	The name of the service, naming rules for services are available
Type of software service	Indicator whether the service is a Business Software Service (BSS), an Internal Software Service (ISS), or a Utility Software Service (USS)
Version number	The version number consists of two digits "x.y". The x is called the release number and is updated by the business enterprise architect who is accountable for the Business Service Description (BSD). The y is updated by the business analyst who is responsible for the BSD.

Table 10.4: (continued)

Aspect	**Description**
Status	The status of the specification, e.g. “Draft” or “Final”
Date of last update	The date of the last update of the specification
Document location	The location of the specification
Author	The author of the specification document specified by his name, function, and department code
Approvals	The persons who have given approval for the different versions of the specification
ICT enterprise/domain architect	The architect is accountable for the SOA business service related to this software service (if applicable)
Revision history	An overview of the changes made to the service
Distribution	A list of people to whom the specification is sent
Linked documents	A list of documents related to the specification document
Business alignment	
Business domain	The name of the business domain, the business subdomain, the business areas and optionally the department code and/or name of the business enterprise architect
Business service	The name of the business service
Software service structure	
Goal(s)	Description in main lines when and for what this software service is used
Description of the result(s)	A short textual description of the added value of the software service
Preconditions	These are conditions that should be checked by consumers before invoking the software service. For important service the check may (also) be done by the provider at the start of service execution. Choices must be made clear in the service contract between consumer and provider.
Postconditions	The indirect results of the service (which is not part of out parameters and for which the consumer may need to take some actions after this service has been executed)
Application	The name of the application that implements the software service
Interaction pattern	Indicator for the way of interaction with the service, this can be request/reply or event report
General constraints	Any constraint applicable, these can be constraints for: (i) one attribute (e.g. date construct), (ii) one attribute of the same type (e.g. list of authorized values), (iii) between attributes (e.g. “valid until date” of credit card), (iv) between entities/classes (e.g. ticket has maximum of four coupons)

Table 10.4: (continued)

Aspect	**Description**
Operation	The name of the operation, a software service usually has only one operation
Main attributes	For the main attributes the following things are specified: goal, result(s), preconditions, and postconditions.
Input structure	A link to the input structure defined using XML Schema (XSD)
Output structure	A link to the output structure defined using XML Schema (XSD)
Return code - business fault	For request/reply types of services, this aspect explains the way return code (errors, warning, successful) are returned. Also, descriptions of the return codes are given. In case of event report types of services this aspect is empty.
Policies	
Intended use	The intended use of the software service, such as reuse within business domain, reuse within Air France/KLM, reuse within Air France only, reuse within KLM only, or unlimited reuse
Security	The appropriate security policy for authentication & authorization of the service, such as confidential, restricted, secret, internal Air France/KLM, internal Air France only, internal KLM only, or unrestricted
Usage requirements	Requirements, not yet mentioned in the above paragraphs, such as minimal and/or preferred requirements at consumer's side to use this service in an optimal way, for example, minimal version of specific client software and memory
Service implementation design logic	
Dependencies other services	A list of other software services need, the orchestration can be showed too using one or more UML sequence diagram(s)
Software service overall description	UML sequence diagrams or UML class diagrams that describe how the software service delivers the result. Please note: this is internal design of the service. These internals should not be exposed to the service consumer.

Table 10.4: Specification aspects for software services at Air France/KLM

10.3 Interview Results

We scheduled the following interviews at Air France/KLM:

- 27 August 2010, 8.30 - 10.30, Dick van Egmond, Business Architect Cargo
- 27 August 2010, 10.30 - 12.30, John van Velzen, Project Architect IS Development with focus on Cargo and Ground Services
- 27 August 2010, 13.00 - 15.00, Wouter Mellink, Enterprise Architect CIO Office
- 30 August 2010, 9.30 - 11.30, Daniël Burggraaf, Manager Service Repository
- 30 August 2010, 13.00 - 15.00, Hans Zonneveld & Paul Ensink, Manager Business Architectuur Ground Services & Enterprise Architect CIO Office

The interviews with Hans Zonneveld and Paul Ensink were combined because of scheduling and availability reasons. Appendix B shows the questionnaire used in these interviews. The following subsections contain the main findings from these interviews. These subsections mirror the structure of the interviews. First, we discuss the rationale for and status of the SOA implementation (10.3.1). Next, we discuss the criteria for the modularization of service-oriented systems that we found in conducting the interviews (10.3.2). Finally, we present the preliminary results of the evaluation of the service specification framework (10.3.3).

10.3.1 Context of SOA Implementation

Figure 10.7 shows the motivation for starting the SOA program at Air France/ KLM three years ago according to the interviewees. KLM as well as Air France have acquired middleware knowledge before the start of the mutual SOA program. KLM started using middleware about ten years ago using the Enterprise-Wide Messaging System (EWMS), a custom middleware system based on the IBM message queueing system MQ. Air France has around twenty years of experience. Before the introduction of the Enterprise Service

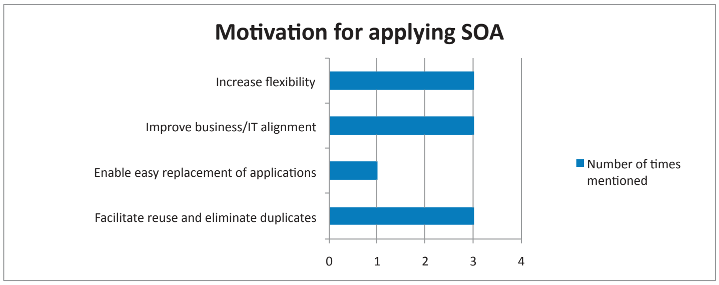


Figure 10.7: Motivation for applying SOA

Bus (ESB) Air France used a middleware system based on Remote Procedure Calls (RPCs) called Adhesion. Both environments had a focus on the technical aspects of system integration; not much effort was spent on the representation of business services to IT services.

From a technical perspective the current SOA environment is relatively mature. At the moment the case study was conducted, the service repository contained a few dozens services. Since business architects got involved later in the SOA program there is still work to be done to streamline the process from business analysis to service development. In the airline industry it is quite common though to think service-oriented on a business level. At Air France/KLM business people already use the notion of (business) building blocks that can be purchased from different partners.

Looking back at the original motivation for starting the SOA program, the interviewees see some of the advantages are met. Flexibility has already increased to a small extent. One of the examples mentioned was checking online. The front-end system could be built quicker because the back-end services were available and did not have to be developed in the online checking-in project. Though Air France/KLM is still working on better business/IT alignment, the notion of domains helped in making analyses for the merger of KLM and Air France. Because the organizational structure was not part of these domain models, discussions had much less political load. The advantage of easy application replacement is not yet met. According to the interviewees this is mainly a timing issue. First all services have to be realized. After that the applications can be replaced. This moment will probably come in the near future. Finally, on a small scale services are reused and duplicates eliminated. A negative experience that the interviewees encountered is the speed of the SOA program; things take longer than expected.

10.3.2 Criteria for Modularization and Service Design

The interviewees proposed several criteria for the identification of domains. These included “maximum cohesion”, “independence of organizational structure”, “sourceability”, and “semantics of business objects”. We discuss the meaning of these criteria in the next section as we gained additional data in the workshop phase. Some of the architects of Air France/KLM define these domains on logical groups of transactions of DEMO models to ensure confirmation to the maximum cohesion criterion.

10.3.3 Service Specification Framework Evaluation

Tables 10.5, 10.6, 10.7, and 10.8 exhibit the evaluation comments of the interviewees regarding the aspects of the service specification framework. The interviewees proposed the following additional aspects they required in a service specification: (i) versioning information, (ii) life cycle information, , and (iii) an indicator whether a service has a request/reply or pub/sub interaction style.

Table 10.5: Evaluation of the Service Executor Aspects

Service Executor Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Actor Role	Air France/KLM distinguishes between a business owner and a technical owner. The business owner is the one who pays for the development and maintenance of the service. The technical owner is the one who makes sure that the service keeps up and running. Next to this, Air France/KLM has a wish to also specify the functional owner. The functional owner makes sure the service offers the correct functionality and deals with functional changes.		x	

	Owners are specified on the level of a department to avoid trouble when people quit their job, get sick etc. Individuals fulfilling the role can be found in the repository.			
Contact Information	The contact information (email address and telephone number) is specified in the repository. Additional information about persons can be found on the corporate intranet.			x

Table 10.5: Evaluation of the Service Executor Aspects

Table 10.6: Evaluation of the Service Production Aspects

Service Production Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Production Act	This information is distributed over multiple fields. The repository has a field called 'description' and the templates have fields called 'overview' and 'goal'. Currently, not much uniformity exists in how these fields are filled. For business services Air France/KLM could specify the DEMO production act as many of the business services are ontological transactions.		x	
Production Information Used	This is specified as the input of the service.			x
	An additional wish of KLM/Air France is to specify the conditions under which certain parts of the input structure have to be filled or can be left empty.			
Production Fact	This is specified as the output of the service.			x

Table 10.6: (continued)

Service Production Aspect	Comments of interviewees	Not applicable	Adapted	Useful
	An additional wish of KLM/Air France is to specify the conditions under which certain parts of the output structure are filled or left empty.			
Production Kind	No distinction is made between ontological, infological, and datalogical services. Air France/KLM does distinguish different types of services, such as request/reply and event-based services, entity services, utility services, and composite services.		x	
Production World Semantics	Currently, semantics is not sufficiently specified, but Air France/KLM acknowledges its importance. The semantics is specified as a dictionary per business area.			x
Preconditions	Preconditions are specified in the template. These are specified in natural language.			x
Postconditions	Postconditions are specified in the template. These are specified in natural language.			x

Table 10.6: Evaluation of the Service Production Aspects

Table 10.7: Evaluation of the Service Coordination Aspects

Service Coordination Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Coordination Acts	Though Air France/KLM does not specify the coordination acts of the ontological model, they do specify different types of exceptions, namely (i) business faults (for expected situations that have meaning on a business level), (ii) technical faults (for several technical types of errors like wrong input formats), and (iii) SOAP faults (for connection errors). The interviewed business architects acknowledge that the complete transaction patters could form a good basis for defining faults.			x
Coordination Kind	Only IT services are fully specified in a service specification. Therefore the distinction between manual services and IT services is not required in the service specification (as it is always an automated service).	x		

Table 10.7: (continued)

Service Coordination Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
Protocol	For IT services this is only specified by exception. Most services use the Web services stack. When a service uses another protocol this is specified. Though non-IT services are not specified in a service specification, the business architects do think there is a need for specifying the protocol for manual services.		x	
Location	In the repository five types of locations are specified: development, customer acceptance, service acceptance, load test, and production.		x	

Table 10.7: Evaluation of the Service Coordination Aspects

Service Contract Option Aspect	**Comments of interviewees**	**Not applicable**	**Adapted**	**Useful**
			x	
Price Quality Combination	Air France/KLM only specifies the Quality-of-Service and not the price as the company does not apply an internal pricing mechanism. Price would be relevant though when external parties would offer or consume services. Also, no distinction is made between different quality levels; if one consumer requires a higher Quality-of-Services, every consumer will get the new higher Quality-of-Service.			
	One of the architects noted that Quality-of-Service can only be specified per component, as all services of a certain component are affected by each other's calls.			

Table 10.8: Evaluation of the Contract Option Aspects

10.4 Workshop Results

The last part of this case study was a workshop organized on 26 November 2010. Five of the six interviewees were able to attend this workshop. The goal of this workshop was to get a better insight into the criteria for service identification at Air France/KLM and to validate our generic service specification framework. We used group decision software to facilitate the workshop.

Table 10.9 shows the criteria proposed by the workshop participants. The letters 'D', 'S', 'BS', and 'SS' have the following meaning. 'D' means that the criterion applies to the identification of domains, 'S' to services in general, 'BS' to business services, and 'SS' to software services. These criteria are not orthogonal. Figure 10.8 shows the average rating of importance of the criteria given by the workshop participants. Also, the standard deviation is calculated so we can see to what degree the participants agree on the average rating.

The criterion having the highest ranking was *'ownership'*. This criterion

Criterion short name	**Criterion long name**
D: Object definitions	A domain has a uniform definition of the syntax and semantics of information objects.
D: Internal cohesion	A domain has a high internal cohesion.
D: Meaningful results	A domain delivers results that are meaningful to its environment.
D: Sourceability	A domain should be sourceable as a whole to an external party.
S: Service contract	A service has behavior that can be specified in a contract (including measurable results, price).
S: Reusable by right granularity	A service is reusable by defining it at a right granularity level.
S: Service description independent of implementation	A service has a stable description that is independent of its implementation.
S: Recognizable to airline industry	A service is recognizable to others in the airline industry; it should be compliant with industry standards.
S: Independence of other services	A service is independent of other services.
S: Interaction required	A service always requires interaction about the result.
BS: Ownership	The scope of the service is aligned with business responsibilities.
BS: Interaction between business areas	A business service is an interaction between business areas.
BS: Need of environment	A business service delivers a result that is wanted/required by its environment.
BS: Complete result (no partial results)	A business service delivers a complete result (i.e. no partial result).
BS: Part of one of more business processes	A business service is part of one or more business processes.
BS: Independence of existing systems	A business service is identified without looking at the existing IT systems.
BS: Description sufficient for use	A business service can be used by a consumer that does not have knowledge of its implementation.
SS: Automated version of business service	A software service is an automated version of an identified business service.

Table 10.9: Criteria proposed by workshop participants

means the service is aligned with business responsibilities. The main motivation for this criterion is that the owner is the one who knows best what can be delivered by his domain. He can make a trade-off between his own interests and that of his consumer. Also, he is the one the consumers can go to when the quality of the service is insufficient. Though these things are all true, the question remains how an owner is assigned to a domain. And when this is determined, the service owner will not define services out of the blue; he will need a set of criteria for service definition. Other criteria that have a rating equal to or higher than 7.0 are *'meaningful results'*, *'independence of existing systems'*, and *'description sufficient for use'*. *'Meaningful results'* should be delivered by a domain to its environment, because otherwise the domain does not provide any value and it should be discontinued. *'Independence of existing systems'* is important because Air France/KLM does not want a tight coupling between the functionality offered by the services and the current IT systems. This tight coupling would oppose the idea of SOA to create a more flexible IT environment. *'Description sufficient for use'* means the description of a service should be sufficient for using it; the consumer should not need to be aware of its internals. Surely, this is true, but this is already inclined in the definition of SOA.

Criteria with a medium score (equal to or higher than 6.0 and lower than 7.0) are *'object definitions'*, *'internal cohesion'*, *'service description independent of implementation'*, *'service contract'*, *'recognizable to airline industry'*, *'interaction required'*, *'reusable by right granularity'*, and *'need of environment'*. *'Object definitions'* means that the domain should specify the syntax and semantics of a certain object. According to the participants it is required that in a domain objects have the same structure and meaning. In an interaction between domains the semantics of the providing domain is used. It can be the case, for instance, that the term 'flight' has a different meaning for the cargo domain than for the ground services domain. This division of semantics is made to give domains freedom in their terminology and to prevent organization-wide discussions on semantics. One objection posed by a participants is that an object can have multiple meanings within Air France/KLM which makes reuse of services harder. That is why semantics needs to be specified very precisely for each domain. *'Internal cohesion'* refers to the principle of maximum cohesion and minimal coupling. The main motivation for this criterion is to reduce the impact of changes by limiting the number of dependencies to those that are absolutely required. *'Service description independent of implementation'* is important because of the notion of loose coupling. However, this principle got a low rating, because it is already captured in the definition of SOA. *'Service contract'* states that the service needs to have a contract that includes measurable properties of the service. *'Recognizable to airline industry'* was considered to be a desirable, but not a mandatory property for services. *'Interaction required'* is seen as a trivial property of a service. One remark was that communication acts in automated services are often implicit. The motivation for *'reusable by right granularity'* is that services should not be too specific to be reused. When a service does not offer enough functionality/information potential consumer will not use it. *'Need of environment'* refers to the property that the service should deliver a result that has value to its environment. If a service does not offer value, there would not be consumers for it. So the provider should not only look at what he has to offer, but also what the 'market' (internal or external) wants.

The lowest scoring criteria (lower than 6.0) are *'sourceability'*, *'interaction between business areas'*, *'independence of other services'*, *'complete result (no partial results)'*, *'automated version of a business service'*, and *'part of one or more business processes'*. *'Sourceability'* refers to the possibility of sourcing a complete domain to an external company. This criterion got a low score of the participants, because this criterion is seen as a derived criterion. Thus a domain is sourceable when it conforms to other criteria, such as maximum cohesion and low coupling and meaningful results. *'Interaction between business areas'* got a low score because it does not really help. It just raises another question, namely how domains should be defined. There were some differences in the motivation for the criterion *'independence of other services'*. Some participants agreed having as rationale that dependencies between services limit the flexibility. Others opposed because of the notion of composite services which does allow dependencies between services. The reason why the criterion of *'complete results'* scored low and had a high standard deviation was a discussion about semantics. One part of the group said that it is impossible to determine what a *part* and a *whole* is, because this depends on your perspective. Thus they disagreed with the principle. The other part of the group interpreted this criterion as the property of a service to keep the world in a consistent state after a call (guarantee integrity). This part of the group gave the criterion a relative high rating. The criterion that a software service is an *'automated version of a business service'* scored low. The business architects stated that this criterion was trivial and therefore did not help. A more technical participant mentioned that this criterion is not always true;

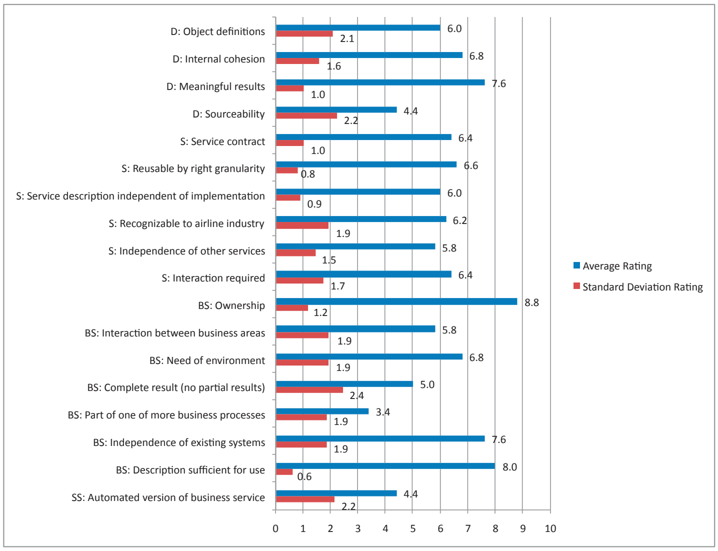


Figure 10.8: Rating of proposed criteria

IT services can also be utility services or internal software services that do not have a direct link with business services. That a service is *'part of one or more business processes'* received a low score, because it is not seen as a criterion by most participants. According to these participants a business process can be supported by services, but we cannot use the business process as a criterion for defining services.

Because of some software problems not enough time was left to discuss the service specification framework in the workshop. For this reason we sent all workshop participants a questionnaire about the service specification framework. In this questionnaire we asked to rate the importance of the different aspects in our service specification framework. Additionally, we asked them to provide a motivation for this rating. Figure 10.9 depicts the average rating and the standard deviation of importance of the service specification aspects. Let us have a look at the aspects with the highest standard deviations, since these are the aspects the participants disagree about. Aspects with a standard deviation higher than 2.0 are: contact information, production act, production world semantics, preconditions, postconditions, coordination acts, protocol, and location. Contact information got two low grades (4 and 5). According to these case study participants direct contact is not required when all information about a service is thoroughly documented and an approval mechanism for publishing and consuming services is put in place. The aspect production act overall received high grades. The high standard deviation is caused by one very low grade (2). This low-grading participants gave as an explanation that ‘it should be possible to alter the production act without altering the service’. We do not see, however, how a service can still be the same service when the production act is altered. A change to the production act would in our eyes lead to another service. Production world semantics also received very high grades. The high standard deviation was again caused by one very low grade (1). This low grade had as a motivation that semantics should not be specified for each service separately, but for the complete domain and all services offered by the domain. So, this person also thinks describing semantics is important, but not for every service separately. Preconditions and postconditions got two low ratings (preconditions 3 and 4 and postconditions 3 and 6). Objections by these persons included that they are often trivial and it is hardly possible to make a complete and consistent set of them. The aspect coordination acts received four high grades and one very low one (3). The low-grader found the coordination acts not relevant, because services at Air France/KLM at the moment have one interaction step. The protocol aspect had the largest standard deviation of all aspects (3,4), because everybody thought this aspect was relevant for technical service, but people disagreed whether or not the aspect was relevant for business services. The location aspect got very diverse ratings. For technical services this was seen as important, for business services there was disagreement. The given explanations for the rates did not really clarify this disagreement. The overall lowest scores were given to production kind and coordination kind. Two reasons for given production kind a low rate were given. First, the information in this aspect is redundant: it can be derived from the production act. Second, Air France/KLM does not use the classification of ontological, infological, and datalogical services. Coordination kind is seen as unimportant, because only IT service are fully specified in a service specification. Therefore it makes no sense to describe in a service specification whether it is a human or IT service.

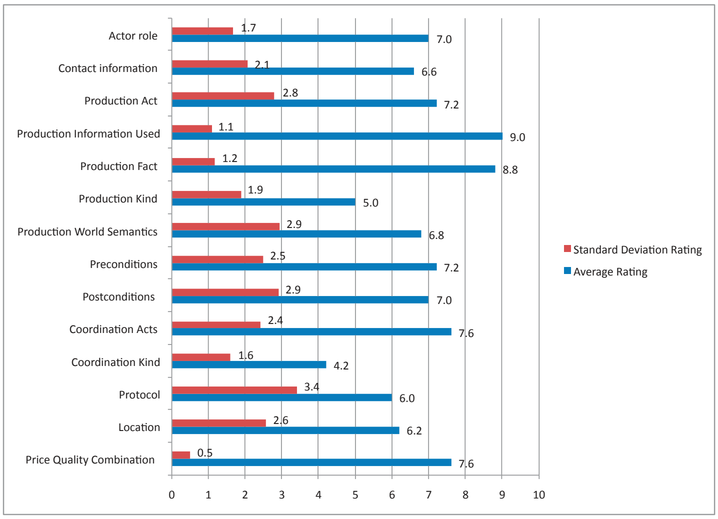


Figure 10.9: Rating of the service specification aspects

10.5 Conclusions

This chapter presented a case study conducted at Air France/KLM. We asked business architects and technical architects what criteria they value for modularization of service-oriented systems and service definition. Additionally, we validated our service specification framework. An interesting finding in relation to modularity in this case study is the application of domaining. Air France/KLM applies the notion of modularity on a business level by defining coarse-grained modules containing people as well as IT systems. These modules are defined independently of organizational structures and interact with each other through business services. Domains are often delimited by clustering related transactions of DEMO models. Criteria for defining domains and their services considered important are: a domain delivers results that are meaningful to its environment (by means of its services), a domain has a high internal cohesion, a business service has an owner, a business service is independent of existing systems, and the description is sufficient for using the service. Not all criteria help us in defining domains and services. Some of them are already included in the definition of SOA (e.g. 'the description is sufficient for using the service') and some raise additional questions (e.g. 'a business service has an owner').

The workshop participants rated the importance of eight of the fourteen aspects of the service specification framework with a 7.0 or higher on a scale of 1 to 10. We can conclude that the participants regard these aspects as crucial in a service specification. For the aspects with a score lower than 7.0 we performed a further analysis to find out the reason for the medium or low score and whether the aspect belongs in a service specification framework or not according to the participants.

Aspects with a score of 6.0 and higher, but lower than 7.0 are: contact information, production world semantics, protocol, and location. Contact information got a medium average rating due to two low grades (4 and 5). As reported by these participants direct contact is not required when all information about a service is thoroughly documented and an approval mechanism for publishing and consuming services is put in place. So this aspect may be optional; it is only required in case consumers need to contact the providers directly and no intermediary takes care of communication. The medium average rating of production world semantics was caused by one very low grade (1). This low grade had as a motivation that semantics should not be specified for each service separately, but for the complete domain and all services offered by the of a domain. So, this person also thinks describing semantics is important, but not for every service separately. When an organization applies the notion of domaining it makes sense to specify the semantics for each domain instead of for each service. The protocol aspect had a very diverse rating and therefore a medium average. This was the case because most people thought this aspect was relevant for technical services, but people disagreed whether or not the aspect was relevant for business services. Another argument for not specifying the protocol for IT services was that the same protocol (Web service stack) was used for each service. Only when IT services used another protocol it was specified. So we see another way in which the protocol aspect in the service specification framework can be used. Instead of specifying the protocol for each IT service a guideline for all services can be put in place that specifies to which protocols the IT services have to conform. The protocol aspect is only filled when the IT service for some reason cannot conform to the guideline and uses another protocol. The location aspect got very diverse ratings. For technical services this was seen as important, for business services there was disagreement. The given explanations for the rates did not really clarify this disagreement.

The lowest scoring (below 6) were production kind and coordination kind. The main causes for the low rating of production kind are (i) that the information is redundant (it can be derived from the production act) and (ii) Air

France/KLM uses another taxonomy for services. Surely, it is true that the information is redundant. The type of service can be derived from the other information in the service specification. However, this classification can be valuable for searching services. Because of the limited amount of services at KLM (dozens) and the clear hierarchical way of structuring them in the service repository, this was not an important requirement. The participants did not think coordination kind was a relevant aspect to specify as they only make complete service specifications for IT services and not for manual services.

The consequences of this case study for the service specification framework are as follows. First of all, some aspects need to be added to the framework. These are: versioning information, life cycle information (different locations for different stadia of the lifecycle of the service), and an indicator whether the service has a request/reply or pub/sub interaction style. Also, based on the importance rating of the different aspects one can define which aspects are most crucial to understanding the behavior of the service. The rating can be used as a means for prioritizing the service specification activities.

Part IV
Conclusions

Chapter 11

Reflections on Case Studies

Abstract This chapter presents reflections on the three conducted case studies. In two of the case studies the participants proposed criteria for identifying coarse-grained modules of service-oriented systems and in all three case studies the participants evaluated our service specification framework. The criteria for module delimitation can be used to define design principles for guiding the application of BCI-3D. BCI-3D is a method that delimits modules based on the maximum cohesion and minimal coupling principle. One of the inputs of the method is a business model of the enterprise (for which we use the ontological model). Besides this, it requires design principles to set the weights of the different types of relationships. We revisit the different types of modularity as described in the theoretical part of this dissertation, i.e. product modularity, process modularity, and organization modularity. We look into the criteria proposed in the case studies and how they are related to these types of modularity.
According to the interviewed practitioners the service specification framework covers the vast majority of aspects that are required in practice. But according to them the framework needs to be extended with some aspects that cannot be derived from the Ψ-theory. First of all, for IT services it is recommended to specify multiple locations instead of one (the development, test, acceptance, and production location). Also, versioning information should be included. The notion of Enterprise Ontology does not give us any help on how to deal with versioning. A versioning schema of "x.y", in which x represents a major version number and y a minor version number was considered to be sufficient in all case studies. Also, we have seen that some aspects are not always required, so they should be optional rather than mandatory.

11.1 Introduction

Chapter 3 of this dissertation presented one of the most widely accepted definitions in general systems theory of modularity, proposed by Baldwin and Clark (2000). They define a module as a unit whose structural elements are powerfully (i.e. strongly) connected among themselves and relatively weakly connected to elements in other units. Unfortunately, the terms 'powerfully' and 'weakly' are open to interpretation. In computer science literature we usually read about the modularity of software systems. Depending on the paradigm applied for software development these modules can be, for instance, functions, objects, or components. In literature from the organizational sciences we find a broader view on modularity. Not only product modularity is described, but also organizational modularity, process modularity, and knowledge modularity. In section 11.2 of this chapter we evaluate the criteria for the delimitation of coarse-grained modules of service-oriented systems as proposed in the case studies. Next, we elaborate on how these criteria can be integrated in BCI-3D in section 11.3. Another goal of the case studies was to evaluate our service specification framework. We reflect on the results of this evaluation in section 11.4.

11.2 Analyzing the Criteria

Both De Lage Landen and Air France/KLM see service-orientation as a means to achieve organizational flexibility. They are well aware that it is not sufficient to only take into account software systems for being able to create products or to change existing products quicker and with less effort. The products we are talking about in this context are intangible products, like loans and lease products for De Lage Landen and flights for Air France/KLM. In the answers on questions about how to structure their service-oriented system, we often heard terms like product, process, organization, and responsibilities in the interviews as well as the workshops. So a number of the criteria proposed were directly related to other types of modularity. Table 11.1 provides an overview.

Let us have a closer look at the criteria. First, some 'criteria' were mentioned that are not really criteria: *'service contract'*, *'service description independent of implementation'*, *'independence of other services'*, *'interaction required'*, *'description sufficient for use'*. Rather, they are part of the definition of the service notion. A criterion that was proposed by multiple people at De Lage Landen was that of *'maximum cohesion and minimal coupling'*. At Air France/KLM this criterion was also proposed by multiple people using the name *'internal cohesion'*. The criterion *'interaction between business areas'* is very much related to maximum cohesion and minimal coupling as these business areas are the modules that are required to communicate through services. Though the workshop participants valued this principle highly and they could also provide the motivation, i.e. changes in one part have little impact on another part, they could not provide a complete method on how to achieve conformation to this criterion. The most important criterion for the division of autonomous environments according to the participants at De Lage Landen is that they do not have any *'functional overlap'* as this can lead to extra maintenance effort and more error situations. However, because of a 'buy before build' policy conformation to this criterion is not always feasible. Also, the participants stated that this criterion may require a lot of analysis work upfront. There is always a trade-off between the modeling effort and the maintenance effort/dealing with error situations. A very low scoring principle at De Lage Landen was *'an autonomous environment should be available on the market as a COTS system'*. The participants opposed to the idea that an external vendor determines what the boundaries of their autonomous environments should look like. Air France/KLM formulated the criterion the other way around as *'independence of existing systems'* and gave it a high rating. So both companies agreed on this criterion. In general, the people at the Lage Landen agreed that the modules and their services should not be dependent on the specific business strategy or implementation technology (the criterion *'strategy and technology agnosticism'*) as these are both highly subject to change. The opposite criterion that the division of modules should depend on the business strategy (*'long term strategy'*) got a relatively low grade. The *'business object'* (De Lage Landen) and *'object definitions'* (Air France/KLM) criteria mean that modules (autonomous environments) are built around a certain business object, e.g. there should be an autonomous environment dealing with client information, one for car information etc. We see this criterion often in practice, for instance in the Dutch Government Architecture NORA (ICTU, 2010) in which they are called 'basisadministraties'. The criterion of *'no single orchestration'* does not deal with the division of autonomous environment, but only with the services that the autonomous environment offers. It says that when the services of an autonomous environment are always called in the same order, these services should be hidden to the consumer and a composite service at a higher aggregation level should be offered for communicating with other environments. The criterion *'reusable by right granularity'* does not help us in defining services as the term 'right' is very subjective and not measurable. The criterion *'recognizable to airline industry'* is of course very specific to Air France/KLM, but it could be formulated as a more general criterion, i.e. *'recognizable to other enterprises operating in the same organizational network'*. The criterion *'complete result'* means that the service needs to leave the world in a consistent state after it is called. This is a general principle for designing IT functions.

The first criterion that has a link with another type of modularity is *'life cycle decoupling'*. This criterion refers to the wish to use IT services for supporting the creation of multiple products and also to put these IT services in the market as separate products. Because De Lage Landen offers intangible products, the difference between its products and its information systems is often less obvious than when speaking about physical products. A subproduct can be exposed easily as a new product by offering it as an IT service, generating additional income. The web service standards make it possible for customers to integrate this service in their own software systems. So the real wish here is to create more modular business products and not only sell the complete product, but to also sell the modules separately to allow *customers* to use these building blocks. Baldwin and Clark (2000) make a distinction between *modularity in production* and *modularity in use*. The first refers to using the notion of modularity to bringing advantages to the production process of a product, e.g. to be able to manufacture car parts in different locations and then assemble them. The latter refers to using the notion of modularity to enable customers to assemble their own products. As we just explained both notions are important to De Lage Landen. For the design of modules that offer IT services this means that it may be more sensible to make multiple small services that are aligned with the product structure of the business product and to assemble these services through service composition if required. Only in this way the IT services can be exposed separately to the market. The criteria *'automated version of business service'*, *'value for consumer and commercial usage'*, *'meaningful results'*, *'need of the environment'* all entail that an IT service has to be directly linked to a business need, i.e. the IT service is aligned with the product delivered by the enterprise. *'Sourceability'* refers to the property of a module that it should be sourceable as a whole to an external party.

The criteria of *'process clusters'* and *'part of one or more business pro-*

cesses' refer to aligning the IT services with the business processes. In other words, services are 'found' through the decomposition of processes until an activity is small enough to support it with an IT service.

An interesting observation was that at De Lage Landen a lot of criteria for determining the boundaries of autonomous environments based on the organization construction scored quite low (e.g. *'budgeting'*, *'supporting team'*, and *'business disciplines'*). At Air France/KLM the criterion *'ownership'* did get a high rating. The motivation for this high rating is that the owner knows best what can be delivered by his domain. He can make a trade-off of his own interests and that of his consumer. However, as explained in the Air France/KLM case study domains are not based on organizational structures (though they may overlap). Basing module boundaries on organizational structures was considered to be a bad idea in both case studies, because, for instance, budgeting structures are often based on internal politics, which can lead to decisions that do not contribute to the enterprise as a whole. Nevertheless, currently the boundaries of the modules and the services they offer are often based on such organization structures. A plausible explanation is given by Conway's Law (Conway, 1968), which states that "...organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations". An interesting note of Conway is the statement that as long as a manager's prestige and power are tied to the size of his budget, he will be motivated to expand his organization. Conway calls this way of reasoning an inappropriate motive in the management of a system design activity. However, he also states that once the organization exists, of course, it will be used. He concludes his 'plea' with the following statement: "Probably the greatest single common factor behind many poorly designed systems now in existence has been the availability of a design organization in need of work." These are forces that play an important role in practice.

Criterion short name	**Relation with other type of modularity**
Service Contract (AFK), Service Description Independent of Implementation (AFK), Independence of Other Services (AFK), Interaction Required (AFK), Description Sufficient for Use (AFK), Maximum Cohesion and Minimum Coupling (DLL) and Internal Cohesion (AFK), Interaction between Business Areas (AFK), No Functional Overlap (DLL), COTS (DLL), Independence of Existing Systems (AFK), Strategy and Technology Agnosticism (DLL), Long Term Strategy (DLL), Business Objects (DLL) and Object Definitions (AFK), No Single Orchestration (DLL), Reusable by Right Granularity (AFK), Recognizable to Airline Industry (AFK), Complete Result (AFK)	Not applicable
Life Cycle Decoupling (DLL), Automated Version of Business Service (AFK), Value for Consumer and Commercial Usage (DLL), Meaningful Results (AFK), Need of Environment (AFK), Sourceability (AFK)	Related to the modularity of the product delivered by the enterprise.
Process Clusters (DLL), Part of One or More Business Processes (AFK)	Related to the modularity of the process executed by the enterprise.
Knowledge Domain (DLL), Business Disciplines (DLL), Responsibilities (DLL) and Ownership (AFK), Budgeting (DLL), Supporting team (DLL)	Related to the modularity of the organization of the enterprise.

Table 11.1: Criteria and relationship with modularity type

11.3 Enterprise Ontology and Alignment of Different Types of Modularity

In the second case study we have seen that De Lage Landen somehow wishes to base the delimitation of its autonomous environments, i.e. the coarse-grained modules of IT services, on product structures and process structures. As seen in the third case study Air France/KLM aims at finding coarse-grained modules of human services and IT services by applying the notion of domaining. But the interviewees of both organizations could not precisely define how these modules can be delimited. Problems arise when the delimitation is directly based on organization structures, because usually these organization structures are not created using a *design approach* and they tend to be unstable. Also, existing IT system boundaries do not provide a good starting point as we want to be able to easily replace systems. Though at both organizations some explicit criteria are available, these criteria together do not provide sufficient means to identify modules in an objective way. Thus the process of module and service identification is not fully defined and the blank spots are filled in by the experience and gut feeling of architects. Though this does not automatically mean that this leads to 'bad' modules (architects often know what works and what not based on their experience), but it results in a subjective and non-traceable process.

Design principles, which together form the *Enterprise Architecture* of the enterprise, influence the value of the weights allocated to the different types of relationships. In our case studies we asked participants what criteria they value for the identification of modules of service-oriented systems. Some of the proposed criteria can be used as a starting point for formulating design principles. Let us give two examples. First, we found a criterion *'autonomous environments are built around business objects (base administrations)'*. Such a principle is often applied if business processes that are structured the same way use different types of information. For instance, an application process for getting a building permit for a house consists of more or less the same steps as an application process for a company license for dealing with toxic chemicals. Both processes consists of the activities of registering the application request, checking the details of the applicant, checking the application itself, dealing with objections etc. Because the processes use different types of information it makes sense to structure this information in different modules. Example modules could be a civilian module, a company module, a house design module, a license module, and an objection module. The proposed criterion implies that relationships among information objects have a high weight; the relationships between process steps and information objects are less relevant than the relationships among information objects. A second example is the criterion *'life cycle decoupling'*. This criterion entails that it should be possible to deliver services independently of the products in which they are used, i.e. services should not be tightly coupled to the product as a whole. The motivation for applying this criterion is the possibility to generate additional income by applying a fee-for-service model. Usually the services offered by a module are only used for making certain products, but sometimes these 'building block services' can also be put on the market independently.

An example of such a module is a credit rating module. This service was required originally for making decisions on loan applications, but it can also be offered to other enterprises for a fee. The life cycle decoupling criterion implies that child transactions should not be allocated to the same module as their parent transactions. Instead, a decomposition structure of modules similar to the transaction tree should be defined. This can be realized by giving a very low weight to relations between mandatory calls among transactions. Especially if one transaction is called by multiple other transaction ('reuse of the transaction') the weights should be set low.

As shown in chapter 5 we can set the weights of different types of relations in the ontological model of the enterprise in a Design Structure Matrix (DSM). This weight defines the strength of the relation.

11.4 Evaluation of the Service Specification Framework

In all three case studies we evaluated the service specification framework. In the first case study at the Port of Rotterdam we applied the framework to specify services ourselves in a software development project. In this project two different companies were building software components for the Port of Rotterdam. One of these companies was working from Rotterdam, the other one from Groningen. The service specifications were used, among others, for work division. The developers of one component could design and implement its internal structure without requiring the other components to be fully designed and implemented yet. They could use the service specifications to understand the expected behavior of their own components and other components. Also, the service specifications could be used to create stubs for testing purposes. After the first services were implemented and tested, we interviewed architects and developers of both companies about their experiences with these service specifications.

In the second case study (De Lage Landen) and the third case study (Air France/KLM) we did not specify services ourselves. De Lage Landen and Air France both had multiple years of experience with SOA and already made service specifications themselves. We compared their service specification templates and actual service specifications to our service specification framework. Also, we interviewed business architects, technical architects, designers, and developers about whether or not they considered the aspects of the service specification framework to be applicable for specifying services in practice. Next to this, we asked them whether any aspects were lacking in the service specification framework.

We were only able to validate our service specification framework for IT services. At the Port of Rotterdam and De Lage Landen the focus of the SOA initiative was on IT services only. At Air France/KLM human services were also included in the SOA initiative. However, only for IT services service specifications were made. Though the business architects at Air France/KLM confirmed that the framework could also be used for human services, we could not compare service specification templates and actual service specifications for human services to our own framework (as they were not available).

In the interviews we asked the practitioners of all three companies about the usefulness of the aspects in our service specification framework. Table 11.2 shows an overview of the results. The rows of this table show whether a certain aspect was seen as not applicable, as useful in an adapted form, or as useful. The company names are abbreviated as follows: Port of Rotterdam to PoR, De Lage Landen to DLL, and Air France/KLM to AFK.

The following aspects are seen as useful by all companies: production information used, production fact, production world semantics, preconditions, and postconditions. Let us look into why the other aspects were not seen as useful by all companies, either because they need to be specified in an adapted form or because they are not applicable at all.

Actor role: In the Port of Rotterdam only a technical owner of the services was assigned. Since this person was responsible for all services, it did not make sense at the Port of Rotterdam to specify this technical owner in all service specifications. De Lage Landen and Air France/KLM stated that the actor role was useful in an adapted form, because they wanted to specify multiple actor roles. De Lage Landen made a distinction between functional ownership and technical ownership and Air France/KLM between business ownership, functional ownership and technical ownership. The business owner is the person who pays for the service, the functional owner is the person who specifies the functional behavior and decides about functional changes to the service, the technical owner is the person who makes sure that the service stays up and running and solves technical problems.

Contact information: The Port of Rotterdam did not specify the contact information for the same reason they did not specify the actor role. De Lage Landen and Air France/KLM did see contact information as a useful aspect to specify.

Aspect	**Not applicable**	**Adapted**	**Useful**
		Service Executor	
Actor Role	PoR	DLL/AFK	
Contact Information	PoR		DLL/AFK
		Service Production	
Production Act		AFK	PoR/DLL
Production Information Used			PoR/DLL/AFK
Production Fact			PoR/DLL/AFK
Production Kind	PoR/DLL	AFK	
Production World Semantics			PoR/DLL/AFK
Preconditions			PoR/DLL/AFK
Postconditions			PoR/DLL/AFK
		Service Coordination	
Coordination Acts		PoR	DLL/AFK
Coordination Kind	PoR/DLL/AFK		
Protocol		AFK	PoR/DLL
Location		PoR/DLL/AFK	
		Service Contract Option	
Price Quality Combination		PoR/DLL/AFK	

Table 11.2: Perceived usefulness of aspects in framework in three case studies

Production act: Air France/KLM did see this as a useful aspect to specify, but currently specifies it using multiple fields (description, overview, and goal). The Port of Rotterdam and De Lage Landen only used one field to describe it.

Production kind: Production kind was not seen as useful by the Port of Rotterdam and De Lage Landen. They did not think it was very useful, at least not at the moment, to make a distinction between different types of services. Also, they did not use the notion of Enterprise Ontology, so they did not know the difference between ontological, infological, and datalogical services. Air France/KLM agreed that it was useful to make a distinction between certain types of services, but they used another type of classification than ontological, infological, and datalogical.

Coordination acts: In the Port of Rotterdam a very basic way of dealing with coordination was seen as sufficient. The company only wanted to specify errors (not cancellations, promises etc). Both De Lage Landen and KLM did see the value of being able to specify all possible coordination acts.

Coordination kind: Coordination kind was not seen as important by any of the companies, because they only specified IT services.

Protocol: Air France/KLM usually does not specify the protocol of the services, because almost all services use the same protocol. Only when another protocol than the regular set of protocols is used, the protocol is specified. The same holds for De Lage Landen.

Location: All three companies agreed that one location was not enough for specifying IT services. Instead, they required a development, a test, an acceptance, and a production location.

Price Quality Combination: All companies specified the Quality-of-Service (QoS) of the services, but none of them specified a price. De Lage Landen and Air France/KLM valued information about pricing to some extent, but certainly not for every service.

Figure 11.1 shows the ratings of De Lage Landen and Air France/KLM for the different aspects of the service specification framework. We do not have ratings from the Port of Rotterdam, because we did not have the opportunity to organize a workshop during the SOA project in which we participated. The aspects are presented in the order of the average rating of the two companies from high to low.

A thing that was lacking according to the interviewees was versioning information. The notion of Enterprise Ontology does not give us any help on how to deal with versioning. A versioning schema of "x.y", in which

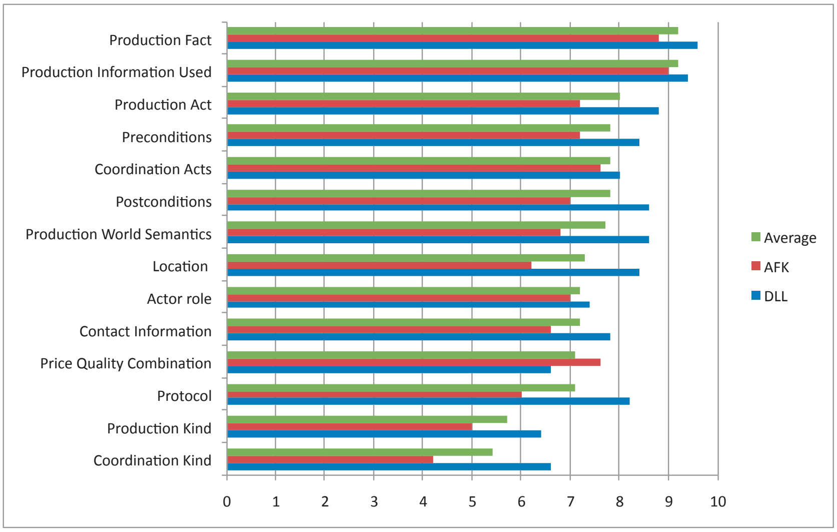


Figure 11.1: Rating of the service specification aspects by De Lage Landen and Air France/KLM

x represents a major version number and y a minor version number was considered to be sufficient in all case studies. In one of the case studies also the way of interacting with the service (request/reply or pub/sub) and policies that specify the obligation(s) of the consumer were seen as a necessary specification aspect.

What does this mean for our service specification framework? First of all, we see that most specification aspects are seen as useful. However, some aspects are not always required in a service specification. The coordination kind aspect is only required if an organization makes specifies human services as well as IT services. Otherwise it can be omitted (that is why it got a low rating in our case studies). The production kind aspect can only be specified in the way it is originally intended if the organization uses the notion of Enterprise Ontology and makes the distinction between ontological, infological, and datalogical services. Other types of service taxonomies are, according to the practitioners, only required with a large number of services. So this aspect does not have to be specified when an organization just starts with SOA, but when the number of services rises such a classification can provide assistance in searching for services. Protocol information is only useful if multiple (versions) of protocols are used. Because it depends on the situation whether or not the aspects of coordination kind, production kind, and protocol need to be specified, they are optional specification aspects.

As we have seen, price information is not always provided. However, this is not a problem in the framework. We can just specify one price quality combination having a price of zero.

Aspects in the framework that require changes are actor role and location. Instead of only one actor role, we need to specify multiple actor roles for IT services. Not only do we need to specify the actor responsible for the service from a business point of view. We also need to specify the actor roles of people who are responsible for specifying and making functional changes to the service (functional owner) and for dealing with the technical aspects of the service (technical owner). The same holds for the location. One location is not sufficient in the service specification. We need to be able to specify multiple locations, e.g. the development, test, acceptance, and production location.

11.5 Conclusions

In this chapter we presented reflections on the conducted case studies. The main premise of service-orientation is to achieve more organizational flexibility. But it is too simple to state that when an enterprise applies principles from the field of software engineering for modularization of service-oriented systems, it automatically gets more organizational flexibility. As we have seen in literature one of the main challenges for getting more organizational flexibility is to align product modularity, process modularity, and organization modularity. In our case studies we asked the participants to propose criteria that they value for the delimitation of modules of service-oriented systems. We found that many of these criteria are related to products, processes and organizational structures. Not all criteria proposed in the workshop help in delimiting coarse-grained modules of service-oriented systems. Some 'criteria' only provide a definition of service or module rather than assistance in delimiting modules or in finding the services through which modules interact. Some other principles, however, can be used as a basis for defining design principles. These design principles can be used for setting weights in the Design Structure Matrices (DSMs) used by BCI-3D (as shown in chapter 5).

Additionally, we presented the results of the evaluation of our service specification framework in the three case studies. We found that the service specification framework covers the vast majority of aspects that are required in practice. But according to the practitioners the framework needs to be extended with some aspects that cannot be derived from the Ψ-theory. First of all, for IT services it is required to specify multiple locations instead of one (the development, test, acceptance, and production location). Also, versioning information should be included. The ontological model of the enterprise does not give us any help on how to deal with versioning. A versioning schema of “x.y”, in which x represents a major version number and y a minor version number was considered to be sufficient in all case studies. What we have also seen is that some aspects of the service specification framework (coordination kind, production kind, and protocol) are not always required in the specification; they are optional rather than mandatory.

Chapter 12

Answers to Research Questions

12.1 Research Questions Revisited

In this dissertation we studied service-oriented systems. We defined a service-oriented system as an enterprise that offers services to its environment, that is structured in a modular way, and for which it does not matter whether a service is delivered by a human being or an IT system. Our wish was to find an approach for delimiting coarse-grained modules of such systems and to create a service specification framework to assist enterprises in describing the behavior of their services. To be able to do these things we needed to know what a service actually is. We encountered several definitions, but none of them was founded on a scientific theory. That is why we wanted to found the notion of service-orientation on Enterprise Ontology and Enterprise Architecture (as defined by the Enterprise Engineering community). After conducting the first case study in which we evaluated our service specification framework, we wanted to broaden the scope of our research and to dive deeper into the concept of modularity. In the second and third case study we therefore not only evaluated the service specification framework, but we also focused on deriving criteria for the delimitation of modules of service-oriented systems.

This led to our central research question: “How can service-oriented systems be constructed and specified in practice by founding service-orientation on the notions of Enterprise Ontology and Enterprise Architecture?”.

The answers to the subquestions posed in chapter 2 are:

RQ1: How can service-orientation be founded on the notions of Enterprise Ontology and Enterprise Architecture?

RQ1.a: How can the Ψ-theory which underlies the notion of Enterprise Ontology be used to define the concept of service?

Because the Ψ-theory describes the interaction between the requesting party and the offering party in a very formal way, it provides a basis for formalizing the notion of service. In chapter 4 we based the definition of service on the complete transaction pattern. Though a service has many similarities with a transaction in the Ψ-theory, they are not equal. While the transaction includes all acts of the initiator and the executor, the service concept only regards the executor side. We therefore defined a service as a part of a transaction rather than a whole transaction. So a service is a pattern of coordination and production acts, performed by the executor of a transaction for the benefit of its initiator, in the order as stated in the complete, universal pattern of a transaction. When implemented it has the ability:

- to get to know the coordination facts produced by the initiator and
- to make available to the initiator the coordination facts produced by itself.

We made a distinction between the following types of services: ontological human service, infological human service, datalogical human service, ontological IT service, infological IT service, and datalogical IT service.

RQ1.b: How can the notions of Service-Oriented Design (SoD) and Service-Oriented Architecture (SOA) be defined on the basis of the Generic System Development Process (GSDP)?

We elucidated the notions of SoD and SOA based on the GSDP in chapter 5. SOA has been defined as a consistent and coherent set of design principles that need to be taken into account in the development of service-oriented systems. We have defined SoD as the design part of a development process, according to the GSDP. It consists of producing successive conceptual models of the object system under consideration. In SoD, this object system is a service-oriented system.

We introduced the following phases in the development process for service-oriented systems: function design of the service-oriented system, construction design of the service-oriented system (including engineering), and implementation of the service-oriented system.

In chapter 6 we used these definitions for comparing several methodologies for service-orientation. Using the high-level distinction of phases we have determined the scopes of the investigated methodologies for service-orientation.

All in all, the GSDP has been very helpful in discussing the coverage of the investigated methodologies, and to elucidate and sharpen the core notions in service-orientation. Next, the application of the methodologies to the same case has shown the differences in depth in which the activities are described. None of methodologies combines full coverage with full depth.

RQ2: How can coarse-grained modules of service-oriented systems be delimited based on notions of Enterprise Ontology and Enterprise Architecture and on the needs of practitioners?

RQ2.a: How can coarse-grained modules of a service-oriented system be delimited by applying the principle of maximum cohesion and minimal coupling on its ontological model?

As a starting point for delimiting coarse-grained modules of service-oriented systems we used BCI-3D. BCI-3D, described in chapter 5, deals with identifying the modules of service-oriented systems and the services for interaction between these modules by using the ontological model of an enterprise as input. For this method the following relationships in the ontological model of the enterprise are important: the relationships among transactions, the relationships between information objects and transactions, and the relationships among information objects. Different types of relationships exist, e.g. conditional relationships and causal relationships between transactions and part-of, state-of, and related-to relationships between information objects. Since an organization may regard one type of relationship as more important than another, it can set weights to the different kinds of relationships.

An algorithm is used to cluster transactions from the business process model and information objects from the information model and thereby forming modules. The maximum cohesion and minimal coupling requirement for modules is an optimization problem for which a genetic algorithm has been developed. The algorithm starts with a predefined solution and generates better solutions through iterations. It generates the starting solution using a greedy graph-partitioning algorithm and improves this solution using the Kernighan and Lin graph-partitioning algorithm.

Summarizing, BCI-3D applies 'maximum cohesion, minimal coupling' as a principle. In applying this principle it uses business process models and information object models as input. The outcome of the algorithms can be influenced by design principles of the organization. BCI-3D does not state which principles are important.

RQ2.b: What criteria do practitioners regard as important for delimiting coarse-grained modules of service-oriented systems?

In two of the three conducted case studies (chapters 9 and 10) we asked business architects as well as technical architects what criteria they value for (coarse-grained) modularization. The criterion of *'maximum cohesion and minimal coupling'* was proposed in both case studies. Architects of both companies agreed that the boundaries of existing IT systems and current organizational structures do not pose good starting points for module identification. By basing module boundaries on existing systems an enterprise would create a tight coupling between its conceptual information system design and the actual implementation and would limit the possibility to easily replace one IT system by another. By basing module boundaries on the organizational structure an enterprise cannot easily get a stable set of modules as organizational structures tend to change often and are often influenced by organizational politics instead of rational enterprise design decisions. Because BCI-3D takes the ontological model of an enterprise as a starting point both problems are prevented. Surely, the ontological model of the B-organization is not sufficient for defining all types of modules. As we have seen in the definition of a service also infological and datalogical services exist. These types of services can only be defined if we have the infological and datalogical model of the enterprise. Some 'criteria' mentioned by practitioners were not really criteria: *'service contract'*, *'service description independent of implementation'*, *'independence of other services'*, *'interaction required'*, *'description sufficient for use'*. Rather, they are part of the definition of the service notion. We did find some criteria in the case studies that can be used as a basis for defining design principles that are input for determining the weights in BCI-3D. Examples are *'life cycle decoupling'* (parts may be used independent of the whole) and *'business object'* (related information should be in one module).

RQ2.c: How can the criteria proposed by practitioners be included in the proposed approach for delimiting coarse-grained modules of service-oriented systems?

As we have seen, BCI-3D can define coarse-grained modules based on the maximum cohesion and minimal coupling principle. The weights to determine what items have strong cohesion can be derived from design principles of the enterprise. Some of the criteria proposed in the case studies can be incorporated into BCI-3D, e.g. the criterion of 'life cycle decoupling'. This criterion entails that it should be possible to deliver services independently of the products in which they are used, i.e. services should not be tightly coupled to the product as a whole. This implies that child transactions should not be allocated to the same module as their parent transactions. Instead, a hierarchy of modules similar to the transaction tree should be defined. This can be realized by giving a very low weight to initiation relations among transactions. Especially if one transaction is called by multiple other transactions ('reuse of the transaction') the weights should be set very low. Though many of the criteria (e.g. 'process clusters', 'business objects', 'independence of other services', 'interaction between business area') can be used as a basis for formulating principles for BCI-3D, this is not the case for all principles. An example we encountered in our case studies was the criterion 'services are recognizable to the airline industry'. This criterion requires knowledge of other enterprises within the airline industry and specific standards used. This information can never be derived from a single ontological model. Another example of a criterion that cannot be incorporated in BCI-3D is 'value for consumer and commercial usage' as this requires an insight into the market of the enterprise.

RQ3: How can the external behavior of a service be specified based on the Ψ-theory and on the needs of practitioners?

RQ3.a: How can a service specification framework be derived from the Ψ-theory?

In chapter 7 we presented a service specification framework based on the Ψ-theory. When we recall the service definition, we see that for calling a service basically three things need to be known to the service consumer, namely information on (i) who provides the service (the executor), (ii) which production fact is brought about by the executor, and (iii) how to interact with the service executor by performing and dealing with coordination acts. We translate these information needs into three main areas of concern, i.e.: service executor, service production and service coordination, respectively. As transactions can have a commercial as well as a non-profit character, we add contract options as an additional area of concern; the consumer needs to know what he gets for which price. For each area of concern we define specification aspects based on the notion of Enterprise Ontology.

RQ3.b: Which aspects of the service specification framework derived from the Ψ-theory do practitioners specify in their own organization and/or regard as useful for getting an understanding of the external behavior of a service?

We evaluated our service specification framework in three case studies in chapters 8, 9, and 10. We interviewed business architects, technical architects and software engineers and found that they in general agreed that the framework is comprehensive enough to describe the externally visible behavior of a service. The following aspects are seen as useful by all companies: production act, production information used, production fact, production world semantics, preconditions, and postconditions. Aspects that were regarded as useful, but with changes are: actor role, contact information and location. Aspects that were seen as optional (depending on the situation are): production kind, coordination kind, and protocol. Furthermore, the Quality-of-Service was specified by the organizations, but the price of the service was not. This would lead to only one price quality combination (having a price of zero). There was some disagreement about the coordination acts aspect. In one case study (Port of Rotterdam) a very basic way of dealing with coordination was seen as sufficient. The company only wanted to specify errors (not cancellations, promises etc). In the other case studies (De Lage Landen and KLM) the practitioners did see the value of being able to specify all possible coordination acts.

RQ3.c What aspects is the service specification framework derived from the Ψ-theory lacking according to practitioners?

Our framework was lacking several aspects that were needed in practice. The first aspect was the possibility to specify multiple locations of a service. If an enterprise develops its own software, either with or without the assistance of an external IT service provider, it in general requires four types of locations: the development, test, acceptance, and production location. Some of the interviewees referred to this type of information as 'the lifecycle of a service'. Next to this, according to several interviewees our framework should include a versioning aspect. The ontological model of the enterprise does not give us any help on how to deal with versioning. For our first case study we applied the backwards compatibility strategy as defined by Erl et al. (2008) to the data model as well as the messages. This results into the following type of version numbers: "x.y", in which x represents a major version number and y a minor version number. The terms major and minor relate to compatibility with previous versions, for instance: version 5.3 is compatible with version 5.1, but not with version 4.8. This was considered to be sufficient in all case studies. In one of the case studies also the way of interacting with the service (request/reply or pub/sub) was seen as a necessary specification aspect.

Summarizing, the service specification framework covers the vast majority of aspects that are required in practice, but according to practitioners it needs to be extended with some aspects that cannot be derived from the Ψ-theory.

Central Research Question

The central question of our research was:

How can service-oriented systems be constructed and specified in practice by founding service-orientation on the notions of Enterprise Ontology and Enterprise Architecture?

We managed to define the notion of service-orientation on the concepts of Enterprise Ontology and Enterprise Architecture. In the answer on research question 1 we saw that we defined a service as the pattern of coordination and production acts, performed by the executor of a transaction for the benefit of its initiator, in the order as stated in the complete, universal transaction pattern of a transaction. When implemented it has the ability: (i) to get to know the coordination facts produced by the initiator and (ii) to make available to the initiator the coordination facts produced by itself. We clarified the notions of Service-oriented Design (SoD) and Service-Oriented Architecture (SOA) by basing them on the GSDP. We made a distinction between designing the function of (a module of) a service-oriented system and its services (black-box view) and the construction of the module and its services (white-box view). In BCI-3D we found an objective, scientific method to identify coarse-grained modules of service-oriented systems. In the answer on question 2 we explained the main principle of BCI-3D: maximum cohesion and minimal coupling. Also, we explained how we can incorporate principles that are based on criteria that we found in case studies. The answer on research question 3 shows that we were able to derive a service specification framework from the Ψ-theory that underlies the notion of Enterprise Ontology. We validated this framework in practice and we found that it covers the vast majority of required service specification aspects. Also, we proposed some changes to the framework based on this evaluation.

12.2 Outlook for Further Research

In this dissertation we did not see service-orientation as a technical paradigm, but as a paradigm for structuring the complete enterprise. Modules of service-oriented systems can be defined using BCI-3D as a method. We gathered criteria for the delimitation of coarse-grained modules and the identification of services in two case studies. Some of these criteria can be used for formulating design principles that define the weights between relationships. These weights are used by the algorithms of BCI-3D. This means that these criteria can be used to tune BCI-3D to the specific wishes of the enterprise. However, we cannot be sure whether the criteria we found are specific to the two enterprises we studied or whether they can be applied to a broader range of organizations. We need to study a large number of companies and see whether or not applying BCI-3D combined with the proposed criteria results in more flexibility for the enterprise.

Another important topic to study in future research is the sensitivity of the weights, i.e. do small changes in the weights lead to large changes in the outcome of the delimitation of the coarse-grained modules. This can be done by applying BCI-3D a large number of times to the same model using small differences in weights and comparing the results.

Furthermore, we designed a generic service specification framework. This framework can be used for specifying human services as well as IT services. It was our intention to determine *what* aspects should be described in a service specification. Future research should focus on *how* these aspects should be specified. How an aspect is specified will be different for human services and IT services. For instance, the location of a human service is usually a physical location or a telephone number, while the location of an IT service is usually a URL. The input of an IT service is usually specified in fields with a certain type (e.g. 'string' or 'integer') and a certain length, while a human service can interpret more free format input descriptions. Sometimes industry standards for human service specification exist. For example, in the Dutch healthcare sector Diagnose Behandel Combinaties (DBC's) are used to define a certain medical treatment and to allocate a price to it. Usually human services are specified using natural language. For IT services more formal descriptions are used. It does not make sense to define a complete new standard for specifying all aspects of the specification framework, because this would be a massive effort and already many standards are available. Instead, it is advisable to look into what existing standards can be used and how they can be combined (if possible). Some of the standards worth investigating are (non-limitative list): UML-OCL and Rule-ML for pre- and postconditions, OWL and ISO/IEC 11179 for semantics, WSDL-S for annotation of input/output structures with semantics, and WSLA and WS-agreement for service level agreements.

When a standardized way of specifying the aspects of the service specification framework (i.e. the 'how part') is available, quantitative research can be conducted. We would like to compare the time required to build services using our service specification framework and using other approaches to service specification. Also, we would like to measure the number of errors and the discovery time for potential consumers of our framework compared to others.

Part V

Appendices

Appendix A

Invitation Letter for Case Study

Dear Mr/Mrs <name>,

I would like to thank you for your interest in participating in my PhD study at the Delft University of Technology. Its goals are (i) to contribute to a better understanding of the principles involved in the identification of services in Service-Oriented Architecture (SOA) and (ii) to provide a specification framework for describing the externally visible behavior of services.
The study started in 2005 and is expected to be completed in Q4 2010 or Q1 2011. Currently, the theoretical foundations are laid by building on the theoretical foundation of the notions of Enterprise Ontology and Enterprise Architecture. An essential part of this study are a number of case studies that form a bridge between theory and practice. These case studies serve two purposes, i.e. getting input from practice to extend theory and validating artifacts derived from theory.

To participate in a case study we ask the following from you and your company:

1. access to documents related to the Service-Oriented Architecture (SOA) (e.g. the Enterprise Architecture, the service catalog, service design guidelines)
2. the opportunity to plan 2-hour interviews with key players in the SOA implementation (approximately 5 interviewees required)
3. the opportunity to organize a 4-hour workshop in which the results of the interviews are discussed and critically reviewed

As a sign of appreciation for your cooperation you get:

1. a report in which the results of your case study are documented
2. access to several technical reports produced in context of this research
3. a copy of the PhD thesis

Of course we will not publish any material about your company without your consent. Before anything is published in either the PhD thesis, technical reports or articles, you get the opportunity to review.

I will contact you by phone to discuss additional details. Looking forward to our cooperation!

Kind regards,

Linda Terlouw
PhD researcher Delft University of Technology

Appendix B

Questionnaire

The interview starts with an introduction about the goal, approach and status of the PhD study. Then we move on to the interview questions.

B.0.1 Questionnaire Interview - Part I

The first part of the interview focuses on the principles for the identification of services. It aims at getting information about the current principles that are used within the organization and the ideas of the interviewee of which principles should be used.

Context

The following questions are asked to get an idea of the context:

1. What are your reasons for applying SOA?
2. When did you start working on SOA?
3. With which earlier middleware concepts do you have experience (if any)?
4. What is the status of the implementation of SOA in the organization?
5. To which goals SOA contribute and how did it contribute (if any)?
6. How SOA contribute to these goals (if any)?
7. Which goals were not met (if any)?
8. Why were these goals not met (if any)?
9. What are the next steps for the implementation of SOA in the organization (if any)?
10. Which methodologies for SOA and/or Enterprise Architecture do you apply (if any)?

Data Gathering for Principles

The next phase of the interview is about gathering data about design principles for the delimitation of coarse-grained modules and the services these modules offer to each other. First, we explain (if necessary) the concept of an architectural principle. Then we asked the following questions:

11. In general, do you start by identifying coarse-grained modules after you identify their services or the other way around?
12. What principles do you use for delimitating coarse-grained modules (also called business components or functional components)?

    For each principle mentioned we ask the following questions:

    (a) What is the rationale (motivation) behind applying this principle?
    (b) What are the consequences of applying this principle?
    (c) Are you familiar with any reasons not to apply this principle?
    (d) If so, what are these reasons?
13. What principles are currently applied for the definition of services?

    For each principle mentioned we ask the following questions:

    (a) What is the rationale (motivation) behind applying this principle?
    (b) What are the consequences of applying this principle?
    (c) Are you familiar with any reasons not to apply this principle?
    (d) If so, what are these reasons?
14. What do you think is the priority ranking of these principles?
15. How do these principle influence each other (if they have an influence on each other)?

B.0.2 Questionnaire Interview - Part II

In this part, we move to the subject of service specification. We start with an open question on which aspects the interviewee thinks need to be specified (to prevent them being pushed into a certain direction). Then we explain our framework and ask questions related to our framework.

1. Which aspects do you think need to be specified?

   For each aspect:

   (a) Do you currently specify this aspect?
   (b) If so, why do you specify this aspect?

    (c) If so, what problems could occur if this aspect is not specified?
    (d) If not, do you see any reasons for specifying these aspects (what problems would it solve)?

2. Do you think our framework is lacking aspects that should be specified?

3. If so, which ones?
   For each mentioned aspect:

    (a) Why do you specify this principle (what problems could occur if this aspect is not specified)?

Appendix C

Workshop Agenda

<date, time, location>

Part I

- Recap of PhD research goal
- Presentation of list of criteria extracted from interview
- Discussion of rationale and consequences of principles
- Prioritizing criteria
- Discovery of potential conflicts between criteria

Part II

- Presentation of service specification framework
- Presentation of feedback received during interviews
- Discussion of framework and feedback

Abbreviations

ATD Actor Transaction Diagram

BCI-3D Business Component Identification 3D

BPEL Business Process Execution Language

BPML Business Process Modeling Language

DEMO Design & Engineering Methodology for Organizations

DSM Design Structure Matrix

GSDP Generic System Development Process

IUT Information Use Table

P&H SoD and Development methodology

RUP Rational Unified Process

SCA Service Component Architecture

SMART Service-Oriented Migration and Reuse Technique

SOA Service-Oriented Architecture

SOAF Service-Oriented Architecture Framework

SoD Service-Oriented Design

SOMA Service-Oriented Modeling and Architecture

TRT Transaction Result Table

UDDI Universal Description Discovery Integration

WSDL Web Service Definition Language

xAF Extensible Architecture Framework

Bibliography

J. Ackermann, F. Brinkop, S. Conrad, P. Fettke, A. Frick, E. Glistau, H. Jaekel, O. Kotlar, P. Loos, H. Mrech, E. Ortner, S. Overhage, U. Raape, S. Sahm, A. Schmietendorf, T. Teschke, and K. Turowski. Standardized specification of business components, February 2002. `http://www.wi2.info/downloads/gi-files/MEMO/Memorandum-english-final-included.pdf`.

E. D. Adamides, Y. Stamboulis, and N. Pomonis. Modularity and strategic flexibility: a cognitive and dynamic perspective. In J. D. Sterman, N. P. Repenning, R. S. Langer, J. I. Rowe, and J. M. Yanni, editors, *Proceedings of the 23rd International Conference of the System Dynamics Society*, Boston, MA, USA, 2005. System Dynamics Society.

A. Akram, R. Allan, and D. Meredith. Best practices in web service style, data binding and validation for use in data-centric scientific applications. In S. J. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting*, pages 297–304, Nottingham, UK, 2006. National e-Science Centre.

A. Albani and J. L. G. Dietz. The benefit of enterprise ontology in identifying business components. In D. Avison, S. Elliot, J. Krogstie, and J. Pries-Heje, editors, *Proceedings of the 19th IFIP World Computer Congress*, pages 243–254, Santiago de Chile, Chile, 2006. Springer.

A. Albani, J. L. G. Dietz, and J. M. Zaha. Identifying business components on the basis of an enterprise ontology. In D. Konstantas, J. Bourriéres, M. Léonard, and N. Boudjlida, editors, *Interoperability of Enterprise Software and Applications*, pages 335–347, Geneva, Switzerland, 2005. Springer.

A. Albani, G. Hardjosumarto, L. Terlouw, and J. L. G. Dietz. Enterprise ontology based service definition. In *Proceedings of 4th International Workshop on Value Modeling and Business Ontologies*, Amsterdam, The Netherlands, 2009.

C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, 1964.

E. D. Arnheiter and H. Harren. A typology to unleash the potential of modularity. *Journal of Manufacturing Technology Management*, 16(7):699–711, 2005.

E. D. Arnheiter and H. Harren. Quality management in a modular world. *The TQM Magazine*, 18(10):87–96, 2006.

A. Arora and A. Gambardella. The changing technology of technological change: general and abstract knowledge and the division of innovative labour. *Research Policy*, 23(5):523–532, 1994.

A. Arsanjani. Best practices in service-oriented architecture, 2006. `http://www.ibm.com/developerworks/blogs/page/-AliArsanjani`.

A. Arsanjani and A. Allam. Service-oriented modeling and architecture for realization of an SOA. In *Proceedings of the IEEE International Conference on Services Computing*, pages 521 – 521, Chicago, IL, USA, 2006. IEEE Computer Society.

A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Gariapathy, and K. Holley. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008.

C. Y. Baldwin and K. B. Clark. *Design Rules: Volume 1: The Power of Modularity.* The MIT Press, Cambridge, MA, USA, 2000.

A. Bask, M. Lipponen, M. Rajahonka, and M. Tinnila. The concept of modularity: diffusion from manufacturing to service production. *Journal of Manufacturing Technology Management*, 21(3):355–375, 2009.

J. Becker, B. Niehaves, and D. Pfeiffer. Case study perspectives on design science research. In V. Vaishnavi and R. Baskerville, editors, *Proceedings of the 3rd International Conference on Design Science Research in Information Systems and Technology*, pages 1–6, Atlanta, GA, USA, 2008.

J. M. Bieman and B. Kang. Cohesion and reuse in an object-oriented system. *ACM SIGSOFT Software Engineering Notes*, 20(SI):259–262, 1995.

D. Birkmeier. Graphpartitionierungsalgorithmen zur Komponentenidentifikation. Master's thesis, University of Augsburg, June 2008.

J. Bloomberg. The role of the service-oriented architect. *The Rational Edge*, May 2003.

N. Boertien, M. van Steen, and H. Jonkers. Evaluation of component-based development methods. In J. Krogstie, T. Halpin, and K. Siau, editors, *Information Modeling Methods and Methodologies*, pages 323–343. Idea Group, 2005.

R. E. Bohn. From art to science in manufacturing: The evolution of technological knowledge. *Foundations and Trends in Technology, Information and Operations Management*, 1(2):1–82, 2005.

G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin–Cummings, Redwood City, CA, USA, 2nd edition, 1994.

L. C. Briand, S. Morasca, and V. R. Basili. Measuring and assessing maintainability at the end of high level design. In D. N. Card, editor, *Proceedings of the Conference on Software Maintenance*, pages 88–97, Washington, DC, USA, 1993. IEEE Computer Society.

L. C. Briand, S. Morasca, and V. R. Basili. Defining and validating high-level design metrics, Report No. UMIACS-TR-94-75. Technical report, University of Maryland Institute for Advanced Computer Studies, College Park, MD, USA, 1994.

L. C. Briand, J. W. Daly, and J. Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

T. R. Browning. The design structure matrix. In R. Dorf, editor, *Technology Management Handbook*, pages 103–111. Chapman & Hall CRC Press, 1999.

T. R. Browning. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. *IEEE Transactions on Engineering Management*, 48(3):292–306, 2002.

S. Brusoni and A. Prencipe. Unpacking the black box of modularity: Technologies, products and organizations. *Industrial and Corporate Change*, 10 (1):179–205, 2001.

M. A. Bunge. *Treatise on Basic Philosophy, vol. 4, A World of Systems.* D. Reidel Publishing Company, Dordrecht, The Netherlands, 1979.

C. Bunse, F. C. Freiling, and N. Lévy. A taxonomy on component-based software engineering methods. In R. H. Reussner, J. A. Stafford, and C. A. Szyperski, editors, *Architecting Systems with Trustworthy Components*, pages 103–119, Dagstuhl Castle, Germany, 2004. Springer.

S. R. Chidamber and C. F. Kemerer. Towards a metrics suite for object oriented design. *ACM SIGPLAN Notices*, 26(11):197–211, 1991.

S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

L. Clement, A. Hately, C. von Riegen, T. Rogers, T. Bellwood, S. Capell, J. Colgrave, M. J. Dovey, D. Feygin, R. Kochman, P. Macias, M. Novotny, M. Paolucci, K. Sycara, P. Wenzel, and Z. Wu. UDDI spec technical committee draft 3.0.2. OASIS committee draft, OASIS, 2004. http://uddi.org/pubs/uddi_v3.htm.

F. Colasuonno, S. Coppi, A. Ragone, and L. L. Scorcia. jUDDI+: A semantic web services registry enabling semantic discovery and composition. In *Proceedings of the 8th IEEE Conference on E-Commerce Technology and the 3rd IEEE Conference on Enterprise Computing*, San Francisco, CA, USA, 2006. IEEE Computer Society.

M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

De Lage Landen. De lage landen, September 2009. http://www.delagelanden.com/.

J. L. G. Dietz. A world ontology specification language. In R. Meersman, Z. Tari, and P. Herrero, editors, *Proceedings of the On the Move Workshops*, pages 688–699, Ayia Napa, Cyprus, 2005. Springer.

J. L. G. Dietz. The deep structure of business processes. *Communications of the ACM*, 49(5):58–64, 2006a.

J. L. G. Dietz. *Enterprise Ontology, Theory and Methodology.* Springer, Berlin Heidelberg, Germany, 2006b.

J. L. G. Dietz. *Architecture - Building strategy in design.* Academic Service, Amersfoort, The Netherlands, 2008.

J. L. G. Dietz and A. Albani. Basic notions regarding business processes and supporting information systems. *Requirements Engineering*, 10(3):175–183, 2005.

J. L. G. Dietz and A. Albani. Benefits of enterprise ontology for the development of ICT-based value networks, communications in computer and information science. In A. R. J. Cordeiro, B. Shishkov and M. Helfert, editors, *Proceedings of 4th International Conference on Software and Data Technologies*, pages 3–22, Sofia, Bulgaria, 2009. INSTICC Press.

J. L. G. Dietz and J. A. P. Hoogervorst. Enterprise ontology and enterprise architecture, how to let them evolve into effective complementary notions. *GEAO Journal of Enterprise Architecture*, 2(1):3–20, 2007.

J. L. G. Dietz and J. A. P. Hoogervorst. Enterprise ontology in enterprise engineering. In *Proceedings of the ACM symposium on Applied computing*, pages 572–579, Fortaleza, Ceara, Brazil, 2008.

F. Ding and L. Jie. An empirical study of flexible business process based on modularity system theory. In *Proceedings of the 3rd International Multi-Conference on Computing in the Global Information Technology*, pages 37–44, Athens, Greece, 2008. IEEE Computer Society.

J. Eder, G. Kappel, and M. Schrefl. Coupling and cohesion in object-oriented systems. Technical report, University of Klagenfurt, 1994.

T. Erl, A. Karmarkar, P. Walmsley, H. Haas, U. Yalcinalp, C. K. Liu, D. Orchard, A. Tost, and J. Pasley. *Web Service Contract Design and Versioning for SOA.* Prentice-Hall, Upper Saddle River, NJ, USA, 2008.

A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service-oriented architecture. In *Proceedings of the IEEE International Conference on Services Computing*, pages 257–260, Chicago, IL, USA, 2006a. IEEE Computer Society.

A. Erradi, S. Anand, and N. N. Kulkarni. SOAF: An architectural framework for service definition and realization. In *Proceedings of the IEEE International Conference on Services Computing*, pages 151–158, Chicago, IL, USA, 2006b. IEEE Computer Society.

F. Flores and J. Ludlow. Doing and speaking in the office. *Decision Support Systems, Issues and Challenges*, pages 95–118, 1980.

J. Gadrey. The characterization of goods and services: An alternative approach. *Review of Income and Wealth*, 46(3):369–87, September 2000.

F. Gallouj and O. Weinstein. Innovation in services. *Research Policy*, 26(4-5): 537–556, 1997.

R. B. Glassman. Persistence and loose coupling in living systems. *Behavioral Science*, 18(2):83–98, 1973.

G. Goldkuhl and K. Lyytinen. A language action view of information systems. In *Proceedings of the 3rd International Conference on Information Systems*, pages 13–29, Ann Arbor, MI, USA, 1982.

S. M. Goldstein, R. Johnston, J. Duffy, and J. Raod. The service concept: the missing link in service design research? *Journal of Operations Management*, 20(2):121–134, 2002.

P. Harmon. Second generation business process methodologies. *Business Process Trends*, 1(5):1–12, 2003.

C. W. Hart. The power of unconditional service guarantees. *Harvard Business Review*, 66(4):54–62, 1988.

B. Henderson-Sellers. *Software Metrics.* Prentice-Hall, Hemel Hempstaed, UK, 1996.

M. Henkel, J. Zdravkovic, and P. Johannesson. Service-based processes: design for business and technology. In *Proceedings of the 2nd international conference on Service-Oriented computing*, pages 21–29, New York, NY, USA, 2004. ACM Press.

P. Herzum and O. Sims. *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise.* John Wiley & Sons, New York, NY, USA, 2000.

A. R. Hevner and S. T. March. The information systems research cycle. *Computer*, 36(11):111–113, 2003.

A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of the International Symposium on Applied Corporate Computing*, 1995.

G. Hoetker. Do modular products lead to modular organizations? Working papers, University of Illinois at Urbana-Champaign, College of Business, 2002. `http://econpapers.repec.org/RePEc:ecl:illbus:02-0130`.

K. Hoffman and P. Eugster. Towards probabilistic assessment of modularity. In *Proceedings of 2nd Workshop on Assessment of Contemporary Modularization Techniques*, Nashville, TN, USA, 2008.

J. A. P. Hoogervorst. *Enterprise Governance and Enterprise Engineering.* Springer, 2009.

J. A. P. Hoogervorst and J. L. G. Dietz. Enterprise architecture in enterprise engineering. *Enterprise Modelling and Information Systems Architectures*, 3(1):3–13, 2008.

ICTU. NORA, 2010. `http://www.e-overheid.nl/e-overheid-2.0/live/binaries/e-overheid/architectuur/NORAv2_0.pdf`.

I. Jacobson. *Object-Oriented Software Engineering: a Use Case driven Approach.* Addison-Wesley, Wokingham, UK, 1995.

D. L. Jisa. Component based development methods: comparison. In *Proceedings of the 5th international conference on Computer systems and technologies*, pages 1–6, Rousse, Bulgaria, 2004. ACM.

S. Johnston. Modeling service-oriented solutions, 2005. `http://www.ibm.com/developerworks/rational/library/jul05/johnston/`.

A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

KLM. Klm, July 2010. http://corporate.klm.com/en/about-klm/profile/.

D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices.* Prentice-Hall, Upper Saddle River, NJ, USA, 2004.

P. Kruchten. *The Rational Unified Process: An Introduction.* Addison-Wesley, Boston, MA, USA, 2003.

A. Kusiak. Integrated product and process design: a modularity perspective. *Journal of Engineering Design*, 13:223–231, 2002.

R. N. Langlois and P. L. Robertson. Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries. *Research Policy*, 21(4):297–313, 1992.

Y. Lee, B. Liang, S. Wu, and F. Wang. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proceedings of International Conference on Software Quality*, pages 81–90, Maribor, Slovenia, 1995.

K. Levi and A. Arsanjani. A goal-driven approach to enterprise component identification and specification. *Communications of the ACM*, 45(10):45–52, 2002.

G. Lewis, E. J. Morris, D. B. Smith, and L. Wrage. SMART: The service-oriented migration and reuse technique, September 2005. Technical Note, CMU/SEI-2005-TN-029, http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn029.pdf.

G. Lewis, E. J. Morris, and D. B. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In G. Chastek, editor, *Proceedings of the Conference on Software Maintenance and Reengineering*, pages 15–23, Washington, DC, USA, 2006. IEEE Computer Society.

J. Luo, B. Montrose, A. Kim, A. Khashnobish, and M. Kang. Adding OWL-S support to the existing UDDI infrastructure. In *Proceedings of the IEEE International Conference on Web Services*, pages 153–162, Chicago, IL, USA, 2006. IEEE Computer Society.

M. W. Maier, D. Emery, and R. Hilliard. Software architecture: Introducing IEEE standard 1471. *Computer*, 34(4):107–109, 2001.

H. Mannaert and J. Verelst. *Normalized Systems, Re-creating Information Technology Based On Laws for Software Evolvability.* Koppa, Kermt, Belgium, 2009.

R. Mantovaneli Pessoa, E. Goncalves da Silva, M. van Sinderen, D. Quartel, and L. Ferreira Pires. Enterprise interoperability with SOA. In M. van Sinderen, P. Johnson, and L. Kutvonen, editors, *Proceedings of the 1st International Workshop on Enterprise Interoperability.*, volume WP08-05, pages 32–45, Munich, Germany, 2008. CTIT, University of Twente.

J. March and J. Olsen. Choice situations in loosely coupled worlds. Technical report, Stanford University, 1975. Unpublished manuscript.

E. Marks and M. Bell. *Service-Oriented Architecture, A planning and implementation guide for business and technology.* John Wiley & Sons, Hoboken, New Jersey, 2006.

J. McClelland and D. Rumelhart. *Parallel Distributed Processing.* The MIT Press, 1995.

J. McGovern, O. Sims, A. Jain, and M. Little. *Enterprise Service-Oriented Architectures: Concepts, Challenges, Recommendations.* Springer, Secaucus, NJ, USA, 2006.

M. D. McIlroy. Mass-produced software components. *Proceedings of NATO Conference on Software Engineering*, pages 88–98, 1968.

G. J. Myers. *Composite/structured design.* Van Nostrand Reinhold, New York, 1978.

OASIS. OASIS, 2006a. `http://www.oasis-open.org/`.

OASIS. Reference model for service-oriented architecture, committee draft 1.0., February 2006b. `http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf`.

P. O'Grady. *The Age of Modularity.* Adams and Steele, 1999.

OMG. Service-oriented architecture SIG, 2006. `http://soa.omg.org/`.

M. Op 't Land. *Applying Architecture and Ontology to the Splitting and Allying of Enterprises.* PhD thesis, Delft University of Technology, June 2008.

Open Grid Forum. Web services agreement specification (WS-agreement), 2007. `http://www.ogf.org/documents/GFD.107.pdf/`.

J. D. Orton and K. E. Weick. Loosely coupled systems: A reconceptualization. *Academy of Management Review*, 15(2):203–223, April 1990.

M. Papazoglou and W. van den Heuvel. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology 2006*, 2(4):412–442, 2006.

D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

J. Pries-Heje, R. Baskerville, and J. Venable. Strategies for design science research evaluation. In W. Golden, T. Acton, K. Conboy, H. Van Der Heijden, and V. K. Tuunainen, editors, *Proceedings of the 16th European Conference on Information Systems*, pages 255–266. National University of Ireland, 2008.

P. Rajasekaran, J. A. Miller, K. Verma, and A. P. Sheth. Enhancing web services description and discovery to facilitate composition. In J. Cardoso and A. Sheth, editors, *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*, pages 55–68, San Diego, CA, USA, 2004.

E. Ramollari, D. Dranidis, and A. J. H. Simons. A survey of service-oriented development methodologies. In *Proceedings of the 2nd European Young Researchers Workshop on Service-Oriented Computing*, Leicester, UK, June 2007.

Rational Software Corporation. Rational unified process: Best practices for software development team, November 2001. Technical Paper, TP026B, `http://www.therationaledge.com`.

J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall, Upper Saddle River, NJ, USA, 1991.

F. Salvador, C. Forza, and M. Rungtusanatham. How to mass customize: Product architectures, sourcing configurations. *Business Horizons*, 45(4): 61–69, 2002.

R. Sanchez and J. Mahoney. Modularity, flexibility, and knowledge management in product and organizational design. *Strategic Management Journal*, 17:63–76, 1996.

M. A. Schilling. Toward a general modular systems theory and its application to interfirm product modularity. *The Academy of Management Review*, 25 (2):312–334, 2000.

H. A. Simon. *The sciences of the artificial.* The MIT Press, 1969.

D. B. Smith, L. O. Brien, and J. Bergey. Using the options analysis for reengineering (OAR) method for mining components for a product line. In *Proceedings of the 2nd International Conference on Software Product Lines*, pages 316–327, London, UK, 2002. Springer.

M. Stal. Using architectural patterns and blueprints for service-oriented architecture. *IEEE Software*, 23(2):54–61, 2006.

J. te Winkel, D. Moody, and C. Amrit. Desperately avoiding bureaucracy: Modularity as a strategy for organisational innovation. In *Proceedings of the European Conference on Information Systems*, Galway, Ireland, 2008.

L. Terlouw. Towards a business-oriented specification for services. In J. L. G. Dietz, A. Albani, and J. Barjis, editors, *Advances in Enterprise Engineering I*, pages 122–136. Springer, 2008.

L. Terlouw and A. Albani. An enterprise ontology-based approach to service specification. Technical report, Delft University of Technology, 2011.

L. Terlouw and J. L. G. Dietz. Positioning methodologies for service-orientation. *Enterprise Modelling and Information Systems Architectures - An International Journal*, 1(5):26–43, 2010.

L. Terlouw and K. E. Maarse. A service specification framework for developing component-based software: A case study at the Port of Rotterdam. In A. Albani, J. Barjis, and J. L. G. Dietz, editors, *Proceedings of the 5th International Workshop CIAO! and EOMAS*, pages 100–114, Amsterdam, The Netherlands, June 2009. Springer.

The Open Group. Service-oriented architecture, 2006. `http://www.opengroup.org/projects/soa/`.

The Zachman Institute for Framework Advancement. Enterprise architecture: A framework, June 2007. `http://www.zifa.com/framework.pdf`.

G. Todorova and B. Durisin. Mixing and matching modularity: A study of strategic flexibility. Technical report, Carnegie Mellon University and Bocconi University, 2008. `http://mtei.epfl.ch/webdav/site/mtei/shared/mtei_seminars/2008/durisin_210408.pdf`.

K. Ulrich. The role of product architecture in the manufacturing firm. *Research Policy*, 24(3):419–440, May 1995.

B. van Zeist and P. Hendriks. Specifying software quality with the extended ISO model. *Software Quality Journal*, 5(4):273–284, 1996.

P. Vitharana, F. M. Zahedi, and H. Jain. Design, retrieval, and assembly in component-based software development. *Communications of the ACM*, 46 (11):97–102, 2003.

W3C. Web services description language, March 2001. `http://www.w3.org/TR/wsdl`.

W3C. Web services glossary, February 2004. `http://www.w3.org/TR/ws-gloss/`.

W3C. Web service architecture, w3C working group note, 2006a. `http://www.w3.org/TR/ws-arch/`.

W3C. W3C, 2006b. `http://www.w3c.org/`.

W3C. Web services policy 1.5 - framework, 2007. `http://www.w3.org/TR/ws-policy/`.

T. Wahl and G. Sindre. A survey of development methods for semantic web service systems. *International Journal of Information Systems in the Service Sector*, 1(2):1–16, 2009.

J. Walkerdine, J. Hutchinson, P. Sawyer, G. Dobson, and V. Onditi. A faceted approach to service specification. In *Proceedings of the 2nd International Conference on Internet and Web Applications and Services*, page 20, Morne, Mauritius, 2007. IEEE Computer Society.

R. A. Watson and J. B. Pollack. Modular interdependency in complex dynamical systems. *Artificial Life*, 11(4):445–457, 2005.

K. E. Weick. Educational organizations as loosely coupled systems. *Administrative Science Quarterly*, 21(1):1–19, 1976.

H. Weigand. The language/action perspective. *Data & Knowledge Engineering*, 47(3):299–300, 2003.

G. M. Weinberg. *An Introduction to General Systems Thinking.* Dorset House Publishing Company, 2001.

E. Wintzen. *Eckart's notes.* Lemniscaat, Rotterdam, The Netherlands, 2007.

WSMO. D10 v0.1 WSMO registry, June 2007. `http://www.wsmo.org/2004/d10/v0.1/`.

R. K. Yin. *Case Study Research: Design and Methods.* Sage, 3 edition, December 2002.

E. Yourdon and L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* Prentice-Hall, Upper Saddle River, NJ, USA, 1979.

V. A. Zeithaml, L. L. Berry, and A. Parasuraman. The nature and determinants of customer expectations of service. *Journal of the Academy of Marketing Science*, 21(1):1–12, January 1993.

Z. Zhang, R. Liu, and H. Yang. Service identification and packaging in service-oriented reengineering. In W. C. Chu, N. J. Juzgado, W. E. Wong, W. C. Chu, N. J. Juzgado, and W. E. Wong, editors, *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, pages 620–625, Taipei, Taiwan, China, 2005. Springer.

This dissertation is based on the following publications:

- L. Terlouw. Towards a business-oriented specification for services. In J. L. G. Dietz, A. Albani, and J. Barjis, editors, *Advances in Enterprise Engineering I*, pages 122–136. Springer, 2008

- L. Terlouw and K. E. Maarse. A service specification framework for developing component-based software: A case study at the Port of Rotterdam. In A. Albani, J. Barjis, and J. L. G. Dietz, editors, *Proceedings of the 5th International Workshop CIAO! and EOMAS*, pages 100–114, Amsterdam, The Netherlands, June 2009. Springer

- A. Albani, G. Hardjosumarto, L. Terlouw, and J. L. G. Dietz. Enterprise ontology based service definition. In *Proceedings of 4th International Workshop on Value Modeling and Business Ontologies*, Amsterdam, The Netherlands, 2009

- L. Terlouw and J. L. G. Dietz. Positioning methodologies for service-orientation. *Enterprise Modelling and Information Systems Architectures - An International Journal*, 1(5):26–43, 2010

- L. Terlouw and A. Albani. An enterprise ontology-based approach to service specification. Technical report, Delft University of Technology, 2011

SIKS Dissertations

1998
1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
1998-4 Dennis Breuker (UM)
Memory versus Search in Games
1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999
1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
1999-3 Don Beal (UM)
The Nature of Minimax Search
1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects
1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
1999-7 David Spelt (UT)
Verification support for object database design
1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000
2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.
2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.
2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001
2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA)
Learning as problem solving
2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002
2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003
2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach
2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
2003-08 Yongping Ran (UM)
Repair Based Scheduling
2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour
2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval
2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models
2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems
2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004
2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM)
Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005
2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM)
AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)
Adaptive Game AI
2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
2005-16 Joris Graaumans (UU)
Usability of XML Query Languages
2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks
2005-19 Michel van Dartel (UM)
Situated Representation
2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006
2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems
2006-04 Marta Sabou (VU)
Building Web Service Ontologies
2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines
2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling
2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering
2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
Aptness on the Web
2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlović (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

2007
2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
2007-07 Natasa Jovanović (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU)

Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations
2007-19 David Levy (UM)
Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns
2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems
2007-24 Georgina Ramrez Camps (CWI)
Structural Features in XML Retrieval
2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

2008
2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations
2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach
2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration
2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning
2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
2008-10 Wauter Bosma (UT)
Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.
2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM)
Adaptive Active Vision
2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven
2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
2008-22 Henk Koning (UU)
Communication of IT-Architecture
2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval
2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues
2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining
2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

2009

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU)
Understanding Classification
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services
2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense 2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System
2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making
2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

2010
2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter
2010-02 Ingo Wassink (UT)
Work flows in Life Science
2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to
Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic Relational Patterns
2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web
2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification
2010-37 Niels Lohmann (TUE)
Correctness of services and their composition
2010-38 Dirk Fahland (TUE)
From Scenarios to components
2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach
2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies
2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services
2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment
2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples
2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions
2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources
2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

2011
2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2011-02 Nick Tinnemeier (UU)
Work flows in Life Science
2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms
2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage
2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction
2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning
2011-10 Bart Bogaert (UvT)
Cloud Content Contention
2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective
2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining
2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling
2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval
2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity
2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games
2011-19 Ellen Rusman (OU)
The Mind's Eye on Personal Profiles
2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

Samenvatting

Modularisatie en specificatie van servicegeoriënteerde systemen

Motivatie en doel

Hedendaagse organisaties staan voor een dilemma. Aan de ene kant willen ze in staat zijn snel te reageren op marktveranderingen; ze willen flexibel zijn. Aan de andere kant willen ze (of beter gezegd: moeten ze) complexiteit in hun organisatie introduceren vanwege strategische keuzes als massacustomisatie van producten en diensten, de introductie van meer complexe producten en diensten en de deelname in interorganisationele netwerken. Het is bijvoorbeeld een grote uitdaging om een overzicht te krijgen van alle bedrijfsprocessen en hun ondersteunende softwaresystemen door hun grootte en hun vervlochten structuur. Een nog groter probleem is dat 'de regels' voor het ontwerp van een organisatie nog slecht begrepen of in ieder geval slecht gedocumenteerd zijn.

In de algemene systeemtheorie is modulariteit voorgesteld als middel voor het omgaan met complexiteit. Het doel dat we in dit proefschrift wilden bereiken is mensen uit de praktijk (met name bedrijfsarchitecten en informatiearchitecten) meer inzicht geven in serviceoriëntatie en modulariteit van organisaties en softwaresystemen. Op dit moment worden in de praktijk vaak beslissingen genomen op basis van intuïtie en ervaring. Onze intentie was ontwerpkennis op het gebied van modulariteit meer onderbouwd en expliciet te maken. We hebben ons onderzoek gebaseerd op literatuur uit de software engineering en uit de organisatiewetenschappen.

Allereerst hebben we gezocht naar criteria voor de decompositie van servicegeoriënteerde systemen in grofmazige modules. Deze modules kunnen mensen en/of softwaresystemen bevatten. We hebben ons dus niet gericht op het structureren van een enkel softwaresysteem, maar op het structureren van de complete set van services geleverd door de organisatie. Voordelen van het introduceren van deze modulariteit zijn onder andere het geheel meer begrijpbaar maken en het effect van een aanpassing in één module op de andere modules minimaliseren. Om de afhankelijkheden tussen modules te minimaliseren is het belangrijk het principe van *'maximale samenhang en minimale koppeling'* te hanteren. Dit heeft geleid tot de vraag *hoe* dit principe van maximale samenhang en minimale koppeling toegepast kan worden. En zijn andere criteria belangrijk voor het identificeren van modules? En zijn deze criteria alleen software engineering-principes of misschien (ook) organisationele criteria? We hebben deze vragen beantwoord in ons 'laboratorium': grote organisaties met complexe IT-omgevingen.

Ten tweede hebben we gezien dat huidige aanpakken voor servicespecificatie erg onvolwassen zijn. Dit levert een probleem op, omdat er verwarring ontstaat wanneer partijen een verschillende interpretatie hebben van elkaars syntax, semantiek of verantwoordelijkheden. Leveranciers en afnemers zullen bijvoorbeeld niet in staat zijn nuttige informatie uit te wisselen als één partij alleen Engels spreekt (of XML) en de ander alleen Nederlands (of EDIFACT). Ook kunnen problemen ontstaan wanneer de leverancier denkt dat de hoogte van een product in centimeters gespecificeerd is, terwijl de afnemer denkt dat dit in inches gedaan is. Misschien noemen leverancier en afnemer niet eens dezelfde kant van een product 'hoogte', omdat het product op verschillende manieren op de vloer gepositioneerd kan zijn. Een ander type probleem ontstaat wanneer de afnemer verwacht dat de leverancier een product tegen de hoogst mogelijke kwaliteitsnormen levert, terwijl de leverancier daadwerkelijk het goedkoopste product van lage kwaliteit levert. Om de informatie over een service te structureren wilden we een servicespecificatieraamwerk ontwerpen. Dit raamwerk ondersteunt de leverancier in het beschrijven van zijn services door aan te geven welke aspecten gespecificeerd moeten worden om een afnemer de service te laten vinden, de service te kunnen benaderen en te kunnen bepalen of de service al dan niet aan zijn behoeften voldoet. We hebben dit raamwerk afgeleid van de Ψ-theory om het te baseren op een geschikte, wetenschappelijke basis.

Samengevat hadden we twee doelen: (i) het vaststellen van criteria voor het afbakenen van grofmazige modules van een servicegeoriënteerd systeem en (ii) het ontwerpen van een raamwerk voor het specificeren van services.

Onderzoeksaanpak

Voor het ontwerp van het servicespecificatieraamwerk hebben we de design science-methodologie gevolgd. Het theoretische deel van ons onderzoek is gebaseerd op de noties van Enterprise Ontology en Enterprise Architecture, zoals gedefinieerd door de Enterprise Engineering community (zie voor meer informatie www.ciaonetwork.org). We hebben het servicespecificatieraamwerk gebaseerd op de Ψ-theorie, de theorie die ten grondslag ligt aan de notie van Enterprise Ontology. Het Generic System Development Process (GSDP) verheldert de notie van architectuur en geeft begrip van het ontwerpproces door de introductie van een duidelijke en consistente terminologie. We specialiseerden het GSDP voor serviceoriëntatie, het meest recente paradigma voor modulariteit van informatiesystemen, en gebruikten dit om een aantal verschillende bestaande methoden op het gebied van serviceoriëntatie met elkaar te vergelijken. We hebben ons servicespecificatieraamwerk gevalideerd door mensen uit de praktijk te vragen of het raamwerk volgens hen compleet is en of het geen irrelevante aspecten bevat. In onze case studies hebben we semigestructureerde interviews gehouden, aangezien vragenlijsten over complexe zaken vaak niet erg zorgvuldig ingevuld worden (‘om er maar vanaf te zijn’). Ook wilden we niet een te vaste structuur in de interviews hanteren om de geïnterviewden ruimte te geven aan te geven wat zij belangrijk vinden. Een laatste stap in het uitvoeren van de case studies was het organiseren van een workshop. Tijdens deze workshop konden de geïnterviewden reageren op elkaars ideeën.

We hebben onze case studies niet alleen gebruikt voor het evalueren van het servicespecificatieraamwerk, maar ook voor het afleiden van criteria voor modularisatie uit de praktijk. Ons doel was inzicht krijgen in criteria die belangrijk zijn voor het afbakenen van grofmazige modules van servicegeoriënteerde systemen. Hoewel we BCI-3D als uitgangspunt hadden, was ons doel niet het valideren van BCI-3D als identificatiemethode. In plaats daarvan wilden we criteria ontdekken die als invoer voor BCI-3D gebruikt kunnen worden (voor het vaststellen van de gewichten). We hebben onze case studies dus niet alleen gebruikt voor de evaluatiestap in de design science-methodologie: de case studies hadden ook een verkennende aard (exploratory case studies (Yin, 2002)).

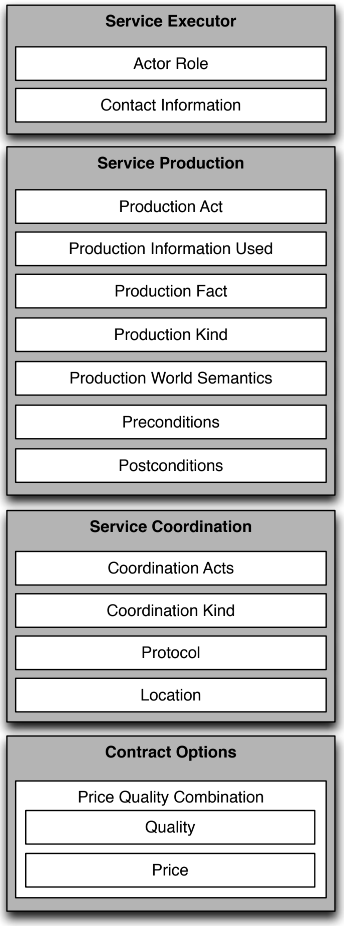
Resultaten

Aangezien de Ψ-theory de interactie tussen een afnemende partij en een leverende partij op een formele manier beschrijft, biedt deze een basis voor het formaliseren van het begrip service. We hebben een service gedefinieerd door gebruik te maken van het complete transactiepatroon als basis. Ondanks de overeenkomsten tussen een service en een transactie in de Ψ-theory kunnen we niet stellen dat ze gelijk aan elkaar zijn. Een transactie bevat namelijk alle acties van de initiator en de executor, terwijl een service alleen de executorkant omvat. Daarom hebben we een service gedefinieerd als een deel van een transactie in plaats van als de hele transactie. Een service is een patroon van coördinatie- en productieacties, uitgevoerd door de executor van de transactie voor de initiator in de volgorde van het complete, universele transactiepatroon. Wanneer de service geïmplementeerd is, heeft deze de mogelijkheid om:

- de coördinatiefeiten geproduceerd door de initiator te weten te krijgen en
- de coördinatiefeiten geproduceerd door zichzelf beschikbaar te maken aan de initiator.

We hebben een onderscheid gemaakt tussen de volgende typen services: ontologische menselijke services, infologische menselijke services, datalogische menselijke services, ontologische IT-services, infologische IT-services en datalogische IT-services. In twee case studies hebben we bedrijfsarchitecten en technisch architecten gevraagd welke criteria zij belangrijk vinden voor het afbakenen van grofmazige modules van servicegeoriënteerde systemen. In beide gevallen gaven de architecten aan te zoeken naar modules met *maximale samenhang en minimale koppeling*. In de eerste case study bij De Lage Landen werd gezocht naar grofmazige IT-modules. Deze werden *autonome omgevingen* genoemd. In de tweede case study, bij Air France/KLM, werd gezocht naar grofmazig organisatiemodules (die bestaan uit mensen en IT-systemen). Deze werden *domeinen* genoemd. Beide organisaties waren het erover eens dat grofmazige IT-modules niet gedefinieerd zouden moeten worden op basis van bestaande IT-systeemgrenzen of commerciële softwarepakketten, omdat dit de mogelijkheid om systemen te vervangen reduceert. Ook waren ze het erover eens dat de organisatiestructuur (organogram) geen goed startpunt is, omdat deze structuur doorgaans niet stabiel is en in grote mate beïnvloed wordt door organisatiepolitiek. BCI-3D kan gebruikt worden om grofmazige organisatiemodules evenals om grofmazige IT-modules af te bakenen, onafhankelijk van huidige IT-systemen en organisatiestructuren. Dit gebeurt door het toepassen van het principe van maximale samenhang en minimale koppeling. Echter, BCI-3D moet uitgebreid worden met ontwerpprincipes voor het bepalen van gewichten van verschillende afhankelijkheden. Deze gewichten zijn invoer voor de algoritmes die de modules vaststellen. In de case studies vonden we voorbeelden van criteria die deze gewichten kunnen bepalen. Bijvoorbeeld, het criterium 'autonome omgevingen zijn gebouwd rondom bedrijfsobjecten (basisadministraties)' leidt tot een relatief hoog gewicht voor relaties tussen bedrijfsobjecten. Het criterium 'levenscyclusontkoppeling' houdt in dat het mogelijk moet zijn services onafhankelijk van het product waarin ze zijn gebruikt te kunnen leveren. Dit wil zeggen dat services niet sterk gekoppeld moeten zijn aan het product als geheel. Dit kan gerealiseerd worden door een erg laag gewicht te geven aan relaties tussen aanroepen tussen transacties. Vooral als een transactie door meerdere andere transacties geïnitieerd wordt (hergebruik) moet het gewicht laag gezet worden.

De onderstaande figuur geeft het generieke servicespecificatieraamwerk dat afgeleid is van de Ψ-theory weer.



We hebben het servicespecificatieraamwerk gevalideerd in drie case studies (het Havenbedrijf Rotterdam, De Lage Landen en Air France/KLM). Het servicespecificatieraamwerk dekt het grootste deel van de aspecten die in de praktijk nodig zijn, maar volgens sommige geïnterviewden dient het uitgebreid te worden met enkele aspecten die niet afgeleid kunnen worden van de Ψ-theory. Allereerst is er niet slechts een enkele locatie nodig, maar moeten meerdere locaties gespecificeerd worden. Als een organisatie eigen software ontwikkelt, ofwel zelf, ofwel met de assistentie van een externe ITdienstverlener, dan zijn er doorgaans vier typen locaties nodig: de ontwikkelomgeving, de testomgeving, de accepatieomgeving en de productieomgeving. Sommige geïnterviewden noemden deze informatie 'de levenscyclus van een service'. Daarnaast gaven sommige geïnterviewden aan dat het raamwerk een versioneringsaspect zou moeten bevatten. In één van de case studies werd ook de manier van interactie met een service (request/reply of pub/sub) als noodzakelijk specificatieaspect gezien.

Toekomstig onderzoek

In dit proefschrift hebben we serviceoriëntatie niet benaderd als een technisch paradigma, maar als een paradigma voor het structureren van de hele organisatie. Modules van servicegeoriënteerde systemen kunnen gedefinieerd worden met behulp van de methode BCI-3D. We hebben criteria verzameld voor het afbakenen van grofmazige modules in twee case studies. Sommige voorgestelde criteria kunnen gebruikt worden als basis voor het formuleren van ontwerpprincipes die de gewichten van relaties bepalen. Deze gewichten worden gebruikt door de algoritmes van BCI-3D. Dit betekent dat deze criteria gebruikt kunnen worden om BCI-3D te conformeren aan de specifieke wensen van de organisatie. In toekomstig onderzoek moet bepaald worden of deze criteria ook voor andere organisaties gelden, aangezien het aantal uitgevoerde case studies te beperkt is om met zekerheid te zeggen dat ze algemeen toepasbaar zijn.

Een ander belangrijk onderwerp om te onderzoeken is de gevoeligheid van de gewichten. Hebben kleine veranderingen in de gewichten misschien grote veranderingen in de uitkomsten van de afbakening van grofmazige modules tot gevolg? Dit kan onderzocht worden door BCI-3D zeer vaak toe te passen op hetzelfde model met steeds een klein verschil in de gewichten en vervolgens de verschillen in uitkomsten met elkaar te vergelijken.

In het onderzoek naar servicespecificatie hebben we ons gericht op het ontwerp van een generiek servicespecificatieraamwerk dat gebruikt kan worden voor het specificeren van menselijke services evenals het specificeren van IT-services. Ons doel was te bepalen *welke* aspecten beschreven zouden moeten zijn in een servicespecificatie. Een vervolgstap is het bestuderen *hoe* deze aspecten gespecificeerd moeten worden. De manier waarop een aspect gespecificeerd moet worden is verschillend voor menselijke services en IT-services.

De locatie van een menselijke service is bijvoorbeeld doorgaans een fysieke locatie of een telefoonnummer, terwijl de locatie van een IT-service vaak een URL is. De invoer van een IT-service is normaal gesproken in velden van een bepaald type (bijv. 'string' of 'integer') en met een bepaalde lengte gespecificeerd, terwijl een menselijke service ook minder formeel gedefinieerde invoer kan verwerken. Soms bestaan er industriestandaarden voor het beschrijven van menselijke services. In de Nederlandse gezondheidszorgsector worden bijvoorbeeld Diagnose Behandel Combinaties (DBC's) gebruikt om medische behandelingen te definiëren en hier een prijs aan te alloceren. Meestal worden menselijke services in natuurlijke taal gespecificeerd. Voor IT-services zijn formele beschrijvingen nodig. Het zou niet verstandig zijn om een compleet nieuwe standaard op te stellen voor het specificeren van alle aspecten van het servicespecificatieraamwerk, omdat dat veel werk met zich mee zou brengen en er al veel standaarden beschikbaar zijn. Het is daarom verstandiger om te bekijken welke bestaande standaarden gebruikt kunnen worden en hoe deze gecombineerd kunnen worden. Sommige standaarden die hiervoor onderzocht kunnen worden zijn (niet-limitatieve lijst): UML-OCL en Rule-ML voor pre- en postcondities, OWL en ISO/IEC 11179 voor semantiek, WSDL-S voor de annotatie van invoer- en uitvoerstructuren met semantiek en WSLA en WS-agreement voor service level agreements.

Wanneer een gestandaardiseerde manier voor het specificeren van de aspecten van het servicespecificatieraamwerk (d.w.z. het 'hoe'-deel) beschikbaar is, kan kwantitatief onderzoek uitgevoerd worden. Een te onderzoeken vraagstuk is hoeveel tijd er nodig is voor het bouwen van services met behulp van ons raamwerk en met behulp van andere aanpakken. Een ander interessant vergelijkend onderzoek is het verschil in de benodigde tijd voor het vinden van services door potentiële afnemers.

Curriculum Vitae

Linda Terlouw (15-9-1980) is geboren in Tilburg en volgde de gymnasium-opleiding aan het Sint-Oelbert in Oosterhout (1992 - 1998). Ze studeerde Technische Informatica en Bedrijfsinformatietechnologie aan de Universiteit Twente (1998 - 2003) en startte haar carrière als consultant bij IBM. In 2005 maakte ze de overstap naar de SOA-adviesgroep van Ordina. In haar rol als solution architect adviseerde Linda grote organisaties over de stapsgewijze invoering van een servicegeoriënteerde manier van denken en het gebruik van ESB-technologie voor de technische implementatie. In hetzelfde jaar ging ze (part-time) terug naar de academische wereld voor een promotieonderzoek in de informatica aan de Technische Universiteit Delft. In 2009 richtte ze het bedrijf ICRIS B.V. op. De missie van ICRIS is het brengen van de laatste wetenschappelijke inzichten over informatiesystemen naar de praktijk. Linda richt zich op het geven van cursussen en het uitvoeren van consultancyopdrachten op het gebied van Enterprise Architecture (EA) en Service-Oriented Architecture (SOA).

Linda Terlouw (15-9-1980) was born in Tilburg (9-15-1980) and attended Sint-Oelbert grammar school in Oosterhout (1992 - 1998). She studied Computer Science and Business Information Technology at the University of Twente (1998 - 2003) and started her career working as a consultant for IBM. In 2005 she joined the SOA Consulting Group of Ordina. In her role as solution architect Linda advised large corporations about the gradual migration towards a service-oriented way of thinking and the use of ESB technology for its technical implementation. In the same year, she went back to academia (part-time) to pursue a PhD in Computer Science at the Delft University of Technology. In 2009 she founded the company ICRIS B.V. The mission of ICRIS is to bring the latest scientific insights in the field of information systems to industry. Linda focuses on giving courses and conducting consulting engagements in the area of Enterprise Architecture (EA) and Service-Oriented Architecture (SOA).