Web Service Detection in Service-oriented Software Development: A Semantic Syntactic Approach

Carolin Letz

2007

Informatik

Web Service Detection in Service-oriented Software Development: A Semantic Syntactic Approach

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich
Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von
Carolin Letz
aus Münster
-2007-

Dekan:	Prof. Dr. Dr. h.c. Joachim Cuntz
Erster Gutachter:	Prof. Dr. Gottfried Vossen
Zweiter Gutachter:	Prof. Dr. Herbert Kuchen
Tag der mündlichen Prüfung:	06.12.2007
Tag der Promotion:	06.12.2007

Abstract

Service orientation has become a buzzword in research and practice during the last years. Enterprises consider investing in service-oriented architectures to increase the reuse of their existing software and to decrease development time and costs. On the one hand, standardization committees have advanced to define many layers of technical standards that are to become the technical backbone of a service-oriented infrastructure. On the other hand, the term service-oriented architecture is not precisely defined and a mature development methodology does not exist, as the numerous offers of software consulting services in this field show.

This thesis makes a contribution to the ongoing discussion on the organization of service-oriented application development. First, the needs of a top-down development process for service-oriented architectures are examined by using a development scenario from financial industry. The ability to detect existing Web services during the design process is revealed to be the key to reusability. To support Web service detection, a relational Web service operation repository is introduced to store and search for existing Web services at the level of operations.

The relational representation in the repository is then used to express the Web service detection problem as a relational schema matching problem on a syntactic level. The match between a Web service operation sought and an existing Web service operation is described with extended relational algebra expressions as a new transformation Web service. This representation is used to define a match hierarchy.

The syntactic approach is extended to express semantics by adding semantic annotations to the relational Web service operation repository. The annotations are based on a taxonomy expressed in description logic. An algorithm is presented, that uses the taxonomy to rewrite SQL queries over the repository for operation retrieval. The algorithm supports the search for different match types. This approach combines Web service standards with semantic Web standards to provide structured algorithmic support for design time detection of Web service operations.

Acknowledgments

I am grateful to my adviser, Prof. Dr. Gottfried Vossen, for his support and constructive criticism concerning the content and structure of this thesis. His opinion that database theory provides solid results that can be reused in many other application contexts was the starting point for this work.

I would like to thank Prof. Dr. Herbert Kuchen for his willingness to act as second reviewer and Prof. Dr. Sergej Gorlatch to act as third examiner during the defense of this thesis.

I am grateful to my colleagues Dr. Jens Lechtenbörger and Stephan Hagemann for proof reading, constructive advice, long hours of discussion, literature hints, and many helpful comments. Further, I thank my colleagues Dr. Peter Westerkamp, Joachim Schwieren, Gunnar Thies, and Daniel Dahl for a lively exchange of opinions as well as Barbara Wicher and Ralf Farke for their organizational and technical help. All of them have encouraged me and cheered me up many times. I also extend my thanks to Frauke Fuchslocher for proof reading.

My deepest thanks go to my parents, my sister, and my godfather who were always reliable companions and have given me moral support whenever I needed it, and to Martin for his patience, understanding, and unconditional loving belief in me.

Contents

Chapter 1

Introduction

Service orientation has become a buzzword in research and practice during the last years. On the one hand, many different areas such as computer networks (e.g., [LZ05]), distributed systems (e.g., [WLWC05]), software engineering (e.g., [BBHN05]), semantic Web technologies (e.g., [BBZ$^+$05]), grid computing (e.g., [HSHG06]), or even bio-informatics (e.g., [HGHHC05]), e-learning (e.g., [Wes05]), and information systems research (e.g., [BHB05]) adopt service orientation as a new research perspective. On the other hand, many software vendors offer new tool suites, such as IBM WebSphere[1], that promise to solve most enterprise application integration (EAI) problems owing to their support of service orientation and service-oriented architectures (SOA). Even more Internet applications describe themselves as service-oriented or claim to be a Web service provider; Amazon[2] is a well-known example. These examples show that the notion of service orientation is widespread, but diverse and dominated by a terminology not commonly defined.

Despite this lack of precision, many companies have embraced the paradigm of service-oriented architectures. Especially the years 2005 and 2006 have seen an increased interest and rising investments in service-oriented technology, infrastructure, and software engineering among companies of many different industry branches as a survey from GCR Research by BEA Systems, Inc. of October 2006 shows [RE06].

Toyota Australia uses a service-oriented architecture to achieve a better and more flexible integration of their own ERP systems with the IT systems of their suppliers and dealers. Before switching to a universal message exchange format and standardized interfaces, information coming from one system had to undergo customized transformation functions or had even to be keyed in

[1]http://www-306.ibm.com/software/websphere/

[2]http://www.amazon.com/

manually [IDC04].

The Dutch airport Schiphol has started the introduction of a new service-oriented system. The main goal of this project is to replace the old pascal-based information and workflow system with a message-oriented system, which is based on broadly accepted XML standards to react flexibly to customer needs and to provide information in real-time to a large number of different consumers such as information monitors, the airport's Web pages, or the Dutch television [BEA06].

Hewlett Packard (HP) has turned its storefront for business-to-business (B2B) shopping into a Web service. The service exposes the complex pricing and configuration logic that resides in an SAP module. Thus, this logic is now available in many different storefronts and is not limited to HP's own B2B solution any longer [BEA03].

The Wachovia Bank, fourth largest bank holding company in the United States based on assets, started a SOA development project in 2004. Each of their nine business units, e.g., leasing or trading, is highly specialized and offers customized financial products. This has led to complex and data-intensive applications that are unique within each business unit, although they encapsulate functionality that might also be useful to other units if they knew about its existence. With the identification of services and a large scale SOA development project, such redundancies were discovered and the reuse of software was increased. For example, the Foreign Exchange division uses a foreign exchange calculator application, which has been reused by the International Payments and Trade division [Mar06].

These few examples reveal that SOA is increasingly used in enterprise settings on a larger scale. The main focus of SOA projects lies on information integration, service provisioning to internal and external customers, and internal software reuse.

In most cases, the key driver behind this interest in SOA is the hope to save IT costs. First, a service-oriented architecture promises an ease of integration efforts. Particularly in larger companies with a long IT history, the IT landscape is diverse. Different hardware from mainframe to pocket PCs and applications by different vendors as well as in different programming languages co-exist causing either high integration efforts or high redundancy in data and functionality. SOA is seen as a new attempt to overcome this heterogeneity.

Second, a service-oriented architecture promises to increase the reuse of software components, thus increasing the programmers' and testers' productivity. In a survey from November 2006 by [BH06], 90% of all participants regarded reuse as the most important key driver for investments in a service-oriented architecture. At the same time, more than 50% of all participants that had already started building a SOA were not experiencing software component reuse. This indicates that there is a gap between the perception of opportunities that service orientation promises and the experience gone through during the implementation of a SOA.

In the survey, half of the participants expect to deploy at least 50 services within one year from starting SOA implementation efforts. More than three thirds intend to reuse between 11% and 40% of the available services. Most participants expect to reuse the services that come from applications built in-house rather than from packaged applications or over the Web. This also reveals a discrepancy: service orientation is seen as an integrating technology, but the reuse does not encompass third party components. This is in contrast to the hopes of the standardization comittees, such as the W3C[3], that have always promoted technological standardization as a bridge between businesses on a global scale.

1.1 Service Reuse

In everyday life, a service is as diverse as the daily delivery of a newspaper, the flight booking or trip planning executed by a travel agent, the approval of a loan in the branch of a local bank, the information provided by the helpdesk in a train station, or the weather forecast provided by an Internet application. All these examples have in common that a service is regarded as the provision of an added value in terms of convenience, comfort, or amusement to customers who seek the service and, in many cases, also pay for it. As such, a service is intangible. This can be regarded as the economic perception of service orientation: providing added value for external customers through intangible goods [ZB96].

This perspective has also been adopted within enterprises. For example, the human resource unit looks for new employees on behalf of other organizational units. The IT infrastructure team provides Web access, e-mail accounts, access to information systems and keeps the middleware and information systems running. All this ultimately contributes to the companies' business goals and is also regarded as service orientation in terms of added value for internal customers.

The enterprise perspective is further expendable. The way how organizational units work and cooperate is defined by business processes [VB96]. A business process defines the flow of activities, the actors involved, and the resources they need to execute the business process. Every activity can be associated with costs and duration time. Optimizing a business process can

[3]http://www.w3.org/

result in a re-ordering of activities, in a re-allocation of activities to different organizational units, or even in the outsourcing of activities to a subsidiary company or an external provider. Especially the latter presumes the existence of well-defined business processes that are precisely modeled in a modular way, e.g., in a formal model such as Petri nets [RR98], and well-understood among all participants within the application domain. For outsourcing certain activities, the different parties must agree upon the prerequisites and the outcome as well as the quality of the service result. From this point of view, service orientation can be seen as the abstraction of business processes or subprocesses that virtually belong to an enterprise, but may physically be executed in a different company, although this is not perceivable from the outside [WCL$^+$05].

The agile and flexible allocation of activities in business processes also effects the underlying IT systems. Loose coupling of system components is necessary to react to changing business requirements. This is an attempt with a long history in computer science.

Software and programming paradigms reflect the need for decomposition. Functions and packages in programming languages such as Ada [Bar06] or PL/SQL [UHMM04] are used for functional decomposition of a program. Object-oriented languages such as Java [AGH05] offer the class concept to achieve a semantic clustering of objects. Components as in J2EE [ACM03] provide a higher level of abstraction and also address application logic, e.g., transaction control or security issues.

In [ACKM04] even more examples are given: the evolution from one-tier architectures to n-tier architectures is an early example of the striving for modular software architectures beyond programming languages. The introduction of a middleware layer in system architectures serves as a bridge between heterogeneous software components and offers a layer of abstraction that helps the programmer to gain easy access. Remote procedure calls (RPC), for example, hide the communication channel from the programmer offering an interface to a remote system that looks just like a local procedure call. Transaction processing monitors (TP monitors) are a widespread middleware that combines RPC with transaction monitoring and is often used in enterprise application integration (EAI) scenarios. The queuing system used in TP monitors to support asynchronous transactional message passing has evolved into a middleware of its own, the so called message-oriented middleware (MOM) that allows persistent queuing and transactional access to queues. These technologies are mostly used in EAI scenarios developed in the closed software landscape of an enterprise. Bridging systems across enterprise boundaries, e.g., using the World Wide Web (WWW), without writing specialized adapters for every application is the next step in this evolution

[ACKM04].

From a software system perspective, service orientation can be seen as the attempt to provide access to existing business functionality offered by different systems irrespective of their location, their platform, and their implementation to be able to react flexibly to changing business needs [WCL$^+$05]. This is achieved by componentization of software applications so that every component within the architecture is dedicated to a specific aspect of the application, e.g., replication or authentication [Ber96]. This led to the paradigm of service-oriented architectures (SOA). Table 1.1 summarizes the different notions of service orientation as explained in this section.

Table 1.1: Notions of service orientation.

Perspective	**Service Orientation**
Everyday Life	Provision of added value to external customer
Company-internal	Provision of added value to internal customer according to business goals
Business Process-oriented	Ability to flexibly decompose complex business processes according to changing business needs
IT Infrastructure-centric	Provision of business functionality irrespective of location, platform, and implementation

The reuse of services in everyday life and within an enterprise is common, but often not standardized. Service detection follows common sense, human experience, or enterprise guidelines. For example, a taxi is ordered via the Internet, by telephone, or called for in the street. It depends on the customer's situation. If the taxi ride is planned, the service provider is chosen deliberately; if the ride is a spontaneous decision, the first available taxi is taken. This shows that service usage is very diverse, although the service itself remains the same.

Such a diversity of means of service detection prohibits the reuse of software services. Therefore, standardization committees have undertaken efforts to define technical standards for the interface definition, communication with and publication of services delivered via the Internet [OAS04]. By now, the standards have been defined, but surveys of the public Internet show that these standards are not actively used in public service registries

[FK05, BSFT06]. Publications on practical SOA implementations within enterprises reveal that additional meta-information management components are custom-built in addition to the registry standards that SOA vendors provide [TM07]. The meta-information repository contains, e.g., software documentations or internal classification systems to enhance the service description. The apparent gap between the high degree of technical standardization in service-orientated architectures and the low reuse effect in practical development projects shows that standards alone do not increase software reuse if they are not integrated into the software development process.

1.2 Thesis Objectives

Top-down software development starts with a survey of user needs and proceeds through various stages of abstraction to the physical system implementation. During this process, the need arises to compare the design of a new system to existing functionality and to decide if existing functions in the form of services can be reused for the new application. At the end of a development project newly built functionality needs to be evaluated to decide if it is suitable to become a new service.

Service detection support in service-oriented software development has not been the main focus of the Web service standardization efforts of the last years. As many standards already exist for various aspects concerning Web services (see Section 2.2.2), detection support for service-oriented software development ought to adhere to existing standards and use them, rather than generate a new standard. Furthermore, in this thesis, Web service detection in service-oriented software development is regarded as a discovery process. Searching is the central step within the Web service detection process, but the specification of the search target and the evaluation of the results returned is also part of the discovery process. Therefore, the term "detection" is preferred to the term "search" in this thesis. The envisioned detection process leads to the following central research question:

> How can an implemented Web service be found for reuse during service-oriented software development?

This lead question directly implies subsequent research questions which will be answered throughout Chapters 2, 3, and 4:

1. Which stages of service-oriented software development need service detection support?

2. Which kind of information is available and contributes to service detection at design time?

3. How can the information needed for searching be organized efficiently for design-time service detection?

4. How can the desired service be described at an appropriate level of abstraction that fits to the software development process, but does not imply a documentation overhead?

5. Can search strategies from other research areas be reused efficiently?

6. What is the nature of a match between a service specification that results from software design and an implemented service?

7. How can the match between service specification and service implementation be operationalized and assessed?

8. How can a match be categorized to express a gradually increasing degree of compliance between specification and implementation?

If meta-information is added to provide semantics that go beyond the technical standards, further questions arise:

9. How can additional semantics be attached to a service so that this additional documentation is generated at design time?

10. How complex is semantic searching based on semantically enhanced service specifications?

11. In which way do semantic extensions of existing standards enhance the quality of the service description?

12. What is the contribution to the software-development process?

As a practical application scenario for service-oriented software development, an online loan application with an underlying financial calculation kernel for product pricing is chosen.

1.3 Thesis Outline

The first four research questions from above are addressed in Chapter 2. First, fundamentals of Web services and the basic standards as far as they are relevant in this thesis are introduced. A methodology for service-oriented software engineering is presented to analyze which kind of service detection support is needed during the different design stages. It is shown that the crucial step for service reuse within the presented top-down approach is the mapping of designed services to existing services in the logical design step. The information known in this step allows for syntactic as well as semantic matching of service capabilities. For the organization of this information, a relational Web service operation repository is proposed. The subsequent chapters resume this repository structure to explore its applicability for syntactic and semantic Web service detection.

Research questions 5 to 8 are addressed in Chapter 3. A relational approach for syntactic Web service matching is developed and examined. Special emphasis is laid on the reuse of algorithms from relational schema matching. It is shown by example that already existing schema matching algorithms can be adapted to provide detection support for Web service operations. These approaches support syntactic search and are dependent on naming conventions.

The remaining research questions will be answered in Chapter 4. The relational approach is enhanced with a taxonomy to allow for semantic Web service matching. This makes syntactic matching independent of naming conventions. An algorithm for semantic Web service operation retrieval based on semantic annotations is presented. The combination of semantic and syntactic matching is examined to determine the complexity and flexibility of the chosen approach.

Chapters 2, 3, and 4 have a common internal structure: They start with a motivation of the guiding research questions for the chapter. The motivation is based on a running example from financial industry that serves as a case study. After the practical motivation, the theoretical foundations of each chapter are briefly summarized as far as they are relevant for the following problem analysis and solution presentation. Then, the research results are presented. Each chapter closes with a short summary. The complete case study is documented in the appendix.

In Chapter 5, related work from the areas of software component retrieval, Web catalogs, search engines, semantic Web research and Web service research is presented and discussed. Differences and similarities to the approach presented in this thesis are pointed out. This chapter complements the foundational sections of Chapter 2, 3, and 4 because it presents studies that treat related research aspects, which have not been used or mentioned before.

In Chapter 6, the results of this thesis are summarized, and an outlook on open issues in the field of Web service detection is given.

Chapter 2

Web Service Operation Repositories

In this chapter, the requirements, foundations, and challenges of Web service reuse in a service-oriented development project are analyzed and a Web service operation repository is introduced to help in the detection of existing Web service operations.

Software reuse in general is a long known idea to decrease development time and costs of software projects. It has already been the aim of procedural and object-oriented programming libraries. Despite this long tradition of reuse as desirable goal of software development, there are still many obstacles to overcome. [Rog05] has observed that software designers still prefer to build new applications with their own code instead of reusing existing components and gives the following reasons for this:

- The existence of reusable components is unknown or undocumented.
- Developers prefer to rely on their own code.
- Developers fear to be held responsible for other program designers' faults.
- Testing the integration of old and new components is perceived as more complex than testing an application built from scratch.

This observation contrasts the hopes of companies that introduce service-orientation in their IT landscape. They want to increase the reuse of their internal software components in the form of Web services [BH06]. Therefore, they must overcome the first obstacle for reuse: the existing services must be published and described in a way that is compatible with the development process to be discoverable. In this chapter, the major demands of service-oriented development within an enterprise setting are analyzed and compared to existing solutions to reveal gaps.

In Section 2.1, the issues of service detection in a service-oriented development project within an enterprise are motivated by a use case example. The detection needs are categorized and the guiding research questions of this chapter are derived, which will be answered in the remaining sections.

In Section 2.2, the foundations of Web services are introduced. Definitions and characteristics as well as technical foundations and related standards are presented to give an overview about the state of the art and standardization efforts, as far as they are relevant for this thesis. The means to make Web services available for reuse and the techniques to discover Web services are addressed in detail as they play an important role in the remainder of this thesis. Further, different search scenarios and use cases for different detection solutions are presented, categorized, and discussed.

In Section 2.3, the paradigm of service-oriented architectures is introduced. Software development approaches for service-oriented architectures and their implications for the software development process are analyzed. Special emphasis is given to intra-enterprise application development and the reusability of Web services within this environment. The running example is resumed to illustrate a service-oriented development methodology that encourages reuse and takes the service consumer and the service provider perspective into account.

In Section 2.4, the different search scenarios that occur during the design process of a service oriented-development project are compared to the technical support given by Web service standards. A repository at the level of Web service operations is presented to make the interface descriptions of Web services accessible for searching. This repository is proposed as an additional component within a service-oriented application landscape.

In Section 2.5, the results of this chapter are summarized while answering the first four questions of Section 1.2.

2.1 Motivation for Web Service Operation Repositories

In the following sections, the search for Web service operations in service-oriented software development is motivated by example. Section 2.1.1 introduces the running example that will be used throughout this thesis. The need to search for Web services at the level of operations will be explained.

Section 2.1.2 categorizes the different approaches towards Web service operation detection. Section 2.1.3 outlines the guiding research questions for this chapter.

2.1.1 Scenario Overview

The following scenario description from the financial industry is used to illustrate operation search in a service-oriented environment.

A fictitious bank is taken into consideration that has decided to specialize in consumer loans. This type of loan is offered to customers who want to buy consumer goods and pay by installments. Such a loan contract typically has a duration from 12 to 60 months, and the net value ranges from 1,000 to 60,000 Euros. Owing to the relatively low amount and short duration, this type of loan promises a profit only if the application process is completely standardized and human intervention is not needed except for the final decision if the loan is granted. Therefore, the bank has decided to offer a loan via the Internet[1].

The central functionality of this application is the calculation of the loan conditions based on customer input. The duration, the net value, and a score for the customers' credit worthiness are needed to calculate the interest rate and the monthly installments. By using personal information such as profession, income, and family size, the customer score is calculated.

If the implementation of such an Internet application is to be based on functionality already implemented, the software architects must be able to find existing operations that fulfill the given task. Here, the need arises to describe an operation before implementing it. This means an operation must be logically, not physically represented in a manner that allows for (semi-)automatic search and match computation. The operation is the search objective, independent of the physical location of the providing application.

Furthermore, using pre-implemented functionality implies that the developer is unable to change the internal implementation of the software component. It is to be used as a black box. Therefore, the only information available during the design process is the specification of the operation, its input and output attributes, which are defined at design time. At this level, also operations that are already implemented can be described. For example, an operation for the retrieval of loan conditions for the consumer loan as described above is offered by an application realized as Web service, called `Loan_Pricing`. The operation itself is called `get_condition`. It takes `Net_Value`, `Duration`, and `Score` as input attributes and returns `Interest` and

[1]A real life example for such an online product is given at www.easycredit.de.

Table 2.1: Simple schema for consumer loan pricing.

Web Service Name	**Operation Name**	**Attribute Name**	**Attribute Type**	**Data Type Domain**	**Data Type Range**
Loan_ Pricing	get_ Conditions	Net Value	IN	Currency	not null, [1,000,. . ., 60,000]
		Duration	IN	Integer	not null, $\{12, 24, 36, 48, 60\}$
		Score	IN	Integer	not null, $\{1, 2, 3, 4, 5\}$
		Interest	OUT	Percentage	not null, [0, . . ., 1]
		Installment	OUT	Currency	not null, ≥ 25

`Installment`. The `Net_Value` is a currency value between 1,000 and 60,000 Euros. The `Duration` is an integer value of 12, 14, 36, 48, or 60 months. The customer `Score` is an integer value ranging from 1 to 5. The `Interest` is a percentage value, the `Installment` is a currency value of at least 25.00 Euros. This operation is represented in Table 2.1.

To search for an implemented operation, the above information must be made available. This can be achieved through various means. A description of the operation can either be published in a central or a decentral registry for reusable components. Further, a software designer must be able to execute a search in this registry either manually or automatically. When the designer has found a reusable operation this operation can be permanently integrated into the new application or it is chosen anew each time the application is executed. These different aspects of operation detection are analyzed in the next section.

2.1.2 Operation Detection Needs

Approaches to operation detection within service-oriented development can be distinguished according to the degree of automation they offer for search execution and the setting in which they are applied as summarized in Figure 2.1.

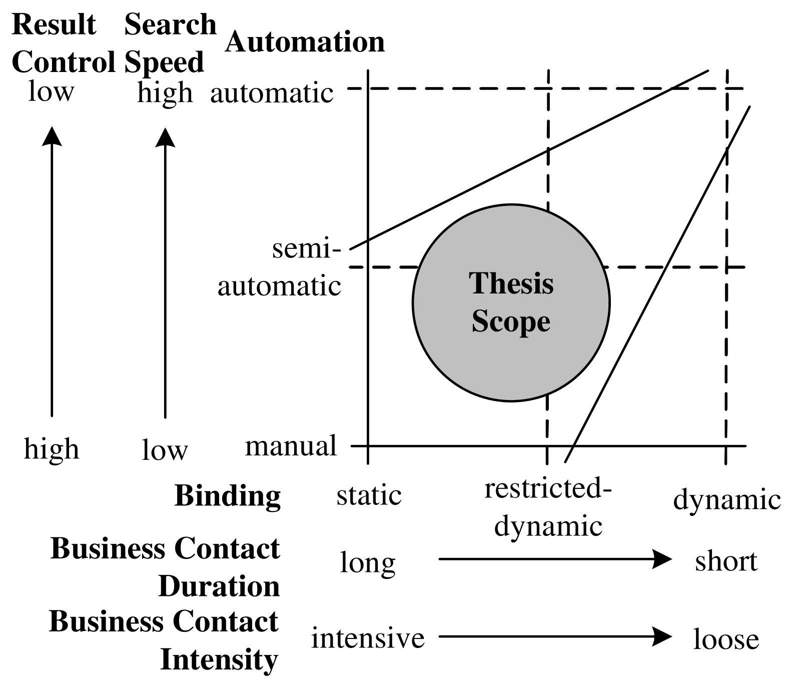


Figure 2.1: Thesis scope.

Detection can be executed fully *automatically* without human intervention, *semi-automatically* with human feedback, or *manually* by a human programmer. With increasing automation, the search speed increases. At the same time the control over the search result through a human programmer is given up and delegated to the search algorithms. Detection can either be used for *dynamic*, *semi-dynamic*, or *static* binding of a single service call. The more dynamic the binding, the shorter and less intensive is the business contact between provider and consumer.

It is obvious that choosing Web services manually is time-consuming if it has to be executed often. This is only worthwhile if the connection between Web service provider and consumer is tight. Then the service is chosen manually at design time. A semi-automatic search that reduces the search space automatically before a human programmer starts Web service inspection can also help in this situation.

If a pool of services is gathered statically to choose from it dynamically at execution time, manual search or semi-automatic search can be applied to assemble such a pool of available services. This results in a higher control over the service pool at design time. The effort spent to assemble this service pool is needed only once. At run-time, flexibility is given because the choice is still possible among the preselected services.

In a highly dynamic setting, it takes too long to choose a Web service manually. With increasing autonomy in the choice of the service providers, the degree of automation must increase, too, to achieve an acceptable execution time. The control over which service is chosen is given up. Policies may still exist that guide the search. At the same time, the business relation between service provider and consumer is dissolved as it may happen that, with every service call, a new service from a different provider is chosen.

The setting taken into consideration in this thesis is motivated by service detection for service-oriented development. The service consumers are developers working in an enterprise setting. In this setting, manual to semi-automatic search support is appropriate. The services belong either to the enterprise's own services or to business partners with which the company cooperates. Choosing services fully dynamically is not intended. The integration of services must be reliable and therefore tests must be conducted to ensure application quality. This is only possible in a static or semi-dynamic setting where semi-automatic search support is needed.

2.1.3 Research Questions

The following research questions are derived from the introductory example and the first categorization of operation detection needs, which resume and deepen the questions of Section 1.2:

- Which stages of service-oriented software development need service detection support? Which requirements must Web service detection support fulfill to foster service-oriented design and development?
- Which kind of information is available and contributes to service detection at design time?
- How can the information needed for searching be organized efficiently for design-time service detection? To which extend do existing Web service detection standards and techniques cover the developers' needs?
- How can the desired service be described at an appropriate level of abstraction that fits to the software development process, but does not imply a documentation overhead? Which additional search support is realizable using the existing core standards?

To answer these questions, a use case study is conducted that extends the introductory example. Service detection needs are revealed by following a service-oriented development approach. Existing Web service detection mechanisms are described, categorized, and compared to the demands of service-oriented development. Finally, an additional component, a Web service operation repository, is presented to enhance search support, especially geared at service-oriented development.

2.2 Foundations of Web Service Detection

Despite the striving for standardization in Web technology industry, the term "Web service" as the implementation of the general service concept is vague and the features emphasized differ among vendors as well as research groups. A Web service can be anything from a printer device reachable within an intranet to an online shop on the Internet. Web service detection approaches must therefore first of all determine their notion of a Web service.

Leyman [Ley03] describes a Web service as a virtual component (see Figure 2.2) because it serves as an abstraction of an arbitrary piece of code or can even serve as a place holder for a service executed by a human being. An example for the latter is the Amazon Mechanical Turk (MTurk)[2] Web service. The service is executed by humans; the software simply establishes the contact to a person capable of the desired task. This section is dedicated to the disambiguation of the term "Web service", its technical foundations and the practical usage scenarios that require Web service detection support.

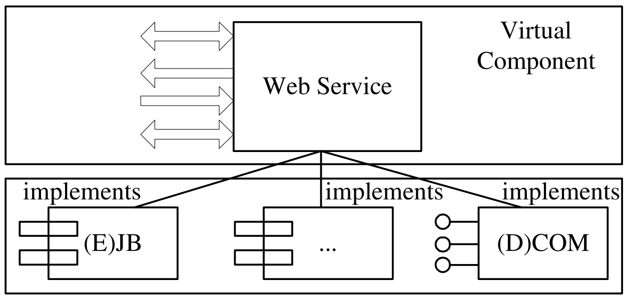


Figure 2.2: Web Service as virtual component according to [Ley03].

In Section 2.2.1, selected Web service definitions from industry and standardization committees are examined to achieve a more precise notion of a Web service. The section closes with a highlighting of the different characteristics that constitute a Web service as perceived within this thesis.

In Section 2.2.2, the technical foundations of Web services are introduced. From a distance, Web services are described by a large number of standards

[2]www.amazon.com

that are either horizontally or vertically related. To give an overview, the core standards are introduced. Further standards will be described in those sections where they are needed.

In Section 2.2.3, the support for Web service detection and usage is analyzed. To use a Web service and to make it available for reuse it must be discovered first. This can be done in different ways and with different use cases in mind.

2.2.1 Web Service Definitions and Characteristics

Many definitions of Web services addressed to developers as well as IT managers emphasize that a Web service is a "piece of business logic [CJ02]" that is "well-bounded, defined, and repeatable [Dev06]", that "can be invoked in a standard manner" [Dev06], and that is "located somewhere on the Internet" [CJ02]. This can be said about almost every Web application that uses standard Internet protocols, e.g., Web pages using CGI or PHP scripts. Such characterizations raise questions such as: what makes a "piece of business logic", and how can a Web service be distinguished from an ordinary Web site.

In an IBM white paper that explains the concept of Web service architectures, the definition is more detailed:

> "A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging ... The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written ... Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services ... [Kre01]."

This definition provides a high-level view on Web services for providers and consumers. From a user perspective, a Web service encapsulates arbitrarily simple or complex business logic in a standardized manner. The standardized interface hides the complexity of the implementation from the user; thus, the Web service becomes a black box. Providers must adhere to a standardized implementation and notation to provide all information necessary for interaction with the Web service.

Different W3C Working Groups use slightly different definitions of Web services. Here, the broadest version is used for analysis:

> "A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols [Wor04h]."

Here, a Web service is first of all a software system that has a unique identity and standardized XML-based interfaces. This makes them interoperable with other software systems. In order to be found by other systems a Web service must also be discoverable. This is the next important characteristic, but the definition does not lay down how to achieve this feature. The communication between different systems via Web services is message-based. Furthermore, the Web service is self-describing as its interface definition contains all information necessary for interaction.

From these observations and in accordance with [Wor04h, WKR$^+$05, ACKM04, WCL$^+$05] the following characteristics are identified that Web services have in common:

Self-describing: A Web service is registered with a description of the messages it accepts. This technical information is necessary for interaction.

Self-contained: A Web service offers an autonomous functionality that is independent of the state of other Web services or the context in which it is invoked. The Web service consumer does not need any specific software. Any programming language with XML and HTTP support is suitable to invoke a Web service.

Modular software components: A Web service offers standard business functionality that is well-defined and clearly distinguishable from other Web services. Being self-describing and self-contained contributes to the modular character of Web services.

Programmatic interfaces to existing applications: For cost-efficient and simple integration old applications remain in place. Their implementation, platforms, and access devices are not replaced, but a new standardized interface is added. It defines the identity of a service.

Published, located, and invoked across the Web: The Web as infrastructure for Web service implementation is used because it guarantees platform independence. Usually, three given standards are applied: SOAP for messaging, the "Web Service Definition Language" (WSDL) for interface description, binding, and technical reference as well as

"Universal Description, Discovery and Integration" (UDDI) for publication and location of services (see Section 2.2.2).

Based on broadly accepted standards: The standards named have been created in cooperation with major software producers such as IBM, Microsoft, and SAP. The consent of software vendors is a necessary prerequisite for acceptance to achieve a broad penetration of the software market.

Language and platform independent: The implementation of the service functionality is hidden behind its interface. From a consumer perspective, the implementation of the application behind it is not important. The consumer does not need any knowledge about the language in which the application is implemented or about the platform it runs on to interact with the Web service. Having programmatic interfaces, residing on the Web, and being built on accepted standards contributes to implementation independence.

Message-based: Web services operate message-based. Their interfaces wrap the underlying applications. Contact to the service requester or other Web services takes place via messages that are passed within a communication channel that supports standardized protocols. Being implementation-independent contributes to the ease of message exchange.

Interoperable: Interoperability is given because Web services are modular and based on messaging.

Composable: Web services can be composed to execute more complex tasks. Being interoperable contributes to composability.

These characteristics will serve as framework for the remainder of this work. Figure 2.3 summarizes the relationship between the different aspects.

2.2.2 Technical Foundations

The general requirements for Web service deployment and use will be explained, by looking at an information service provider such as Reuters[3]. Banks regularly import, e.g., the latest exchange rates or yields of government bonds from such providers. To achieve interoperability, the information service provider must offer a standardized interface that describes how to send

[3]http://customers.reuters.com/

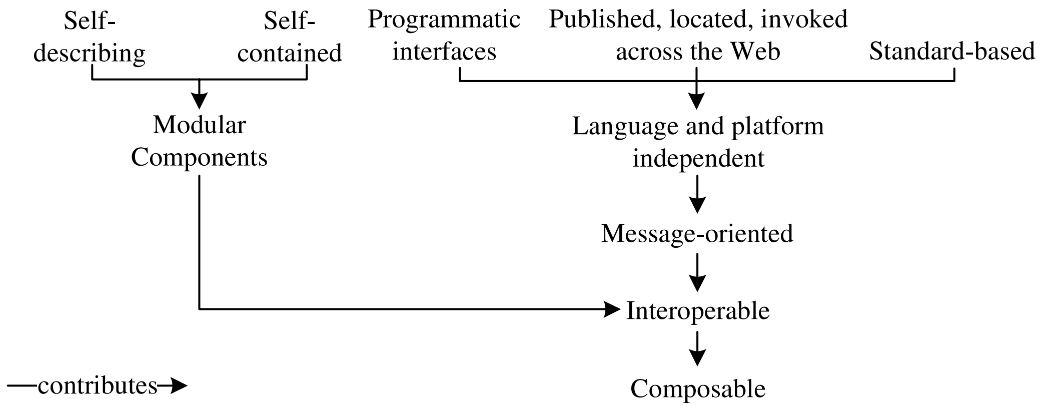


Figure 2.3: Web service characteristics.

and receive messages for information download. This interface is defined using WSDL, the Web services definition language. The messages also use a pre-defined format, defined in SOAP. For a better visibility of the service, the service provider publishes the functionality in a public directory, described by UDDI. The relationship between SOAP, WSDL and UDDI is summarized in Figure 2.4. SOAP serves as transportation standard for registration, search, and invocation. UDDI offers interfaces to publish and search for a service. The service itself resides with the service provider, may be published using UDDI, and is invoked using SOAP. The three standards SOAP, WSDL and UDDI are introduced in more detail in the next paragraphs.

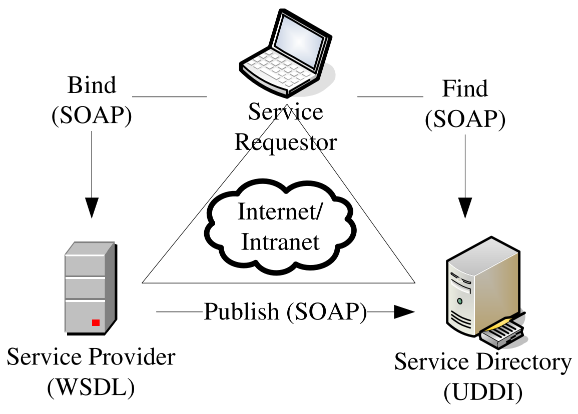


Figure 2.4: General Web services set up according to [WCL+05].

SOAP

SOAP 1.2[4] [Wor03a, Wor03b] is a lightweight protocol built to allow loosely coupled interaction between applications across the Internet. It is based on XML to convey information in a structured and standardized manner. According to [ACKM04], SOAP specifies in particular:

- a message format suitable for one-way message exchange,
- a mechanism to enclose an RPC in a SOAP message and to send the result back,
- all elements of a message that the receiving party ought to understand,
- a binding to HTTP or SMTP for synchronous and asynchronous communication.

A SOAP message consists of an XML envelope that contains an optional header and a mandatory body. Header and body may contain an arbitrary number of header blocks and body blocks respectively.

The header contains information for intermediary nodes that pass the message along to the receiver. The body contains the actual content of the message exchange. This can either be a simple document, or an RPC with the name of the called method and its input parameters, or the reply to such a call. Exchanging documents is called *document-style interaction* in contrast to using RPC, which is called *RPC-style interaction*. This is shown in Figure 2.5.

SOAP defines an encoding for data structures, but it does not impose this encoding. Any other encoding that the involved parties agree upon can be used. For example, IFX [5] is an XML standard for the exchange of financial data such as stock market orders or other kind of transactions. This standard has been recently extended to be used within SOAP messages. Thus, the XML structure of a SOAP message is used for transport, but the data structures used within the message conform to a domain specific standard that carries additional meaning for the sending and receiving parties.

To use SOAP, the message must be transported through a network. Different network transport protocols can be bound to a SOAP message. Common bindings are HTTP and SMTP. When SOAP is used with an HTTP binding the message is included in an HTTP POST or GET request. The recipient of the message is defined in the target URL of the HTTP command.

[4]At the time of writing, the current version of SOAP was SOAP 1.2. The current version can always be found under http://www.w3.org/TR/soap/.

[5]http://www.ifxforum.org/home

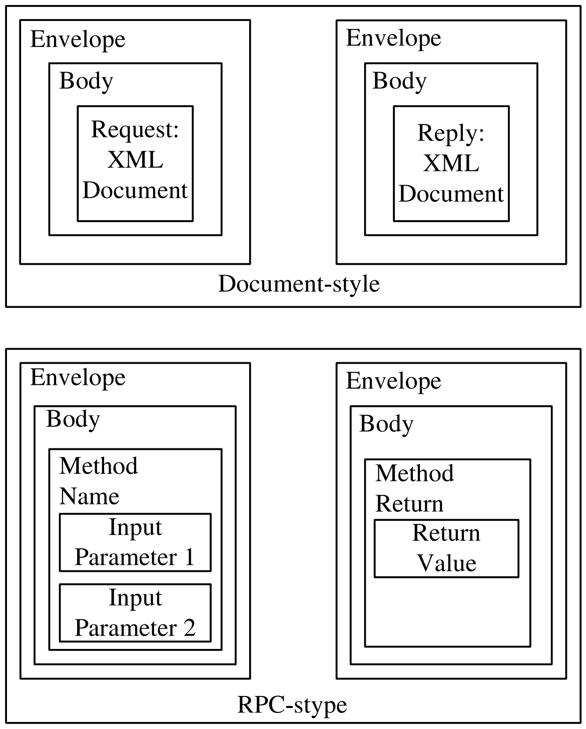


Figure 2.5: SOAP interaction styles according to [ACKM04].

WSDL

WSDL 1.1[6] [CCMW01] is a generic Web service description language that can be used for any type of Web service irrespectively of its underlying functionality. According to [ACKM04] a WSDL document describes:

- a Web service interface with service name, operation name, input and output parameters,
- access mechanisms because Web services can reside on different platforms,
- the location of the service, which is the destination of the SOAP message,
- interaction patterns to exchange several messages asynchronously or synchronously.

A WSDL specification consists of an *abstract part* and a *concrete part* as depicted in Figure 2.6.

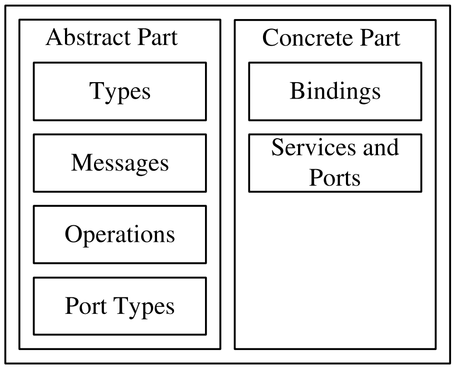


Figure 2.6: WSDL service specification according to [ACKM04].

The abstract part contains port type definitions that form a logical grouping of related operations. Each operation defines the exchange of messages, usually SOAP messages. With the current WSDL version, there are four basic operation primitives: *one-way*, *notification*, *request-response* and *solicit-response*. *One-way* and *notification* are used for asynchronous messaging.

[6]At the time of writing the current recommendation of WSDL was still version 1.1. WSDL 2.0 became a candidate recommendation in March 2006. The current version can always be found under http://www.w3.org/TR/wsdl.

Request-response and *solicit-response* messages are for synchronous communication. These operation types show that a Web service can either act like a service provider, answering a request or, like a client, initiating a service call. The data types used in the messages are also defined in the WSDL document. Usually, the XML type system [Wor04i] is used, but other type systems are also allowed.

The concrete part supplies interface bindings, ports, and services. Interface bindings specify the message encoding, e.g., RPC-style interaction with SOAP encoding of the abstract WSDL types, and the protocol bindings for the operations and messages of a given port type, e.g., SOAP for communication, and HTTP for transport.

The separation of abstract interface from concrete bindings is intended to make the abstract part of a WSDL specification reusable. The same port type can be used in combination with different bindings and can be made available at different addresses [ACKM04].

The W3C Working Group suggests three possible usage scenarios for WSDL:

1. as traditional service description language that defines a service contract,

2. as input information for stub compilers and development environments that automatically generate the stubs and skeletons to invoke the service,

3. as a vehicle to describe the semantics of a service.

The first two usage scenarios are already very common. Usually, a development suite automatically generates a WSDL document for a given API of, e.g, a Java class or a stored procedure in a database system. From this WSDL file, the development environment can also generate the client stub and the server stub. The last usage scenario, concerning semantics, is still very vague. WSDL has been designed to describe the interface of any service, irrespectively of its application domain [ACKM04]. This idea will be resumed in Chapter 4 of this thesis.

The latest W3C candidate recommendation as of time of writing is WSDL 2.0 [Wor07c]. The structure of the WSDL document has changed slightly in this recommendation compared to the afore-presented WSDL 1.1 standard. It still consists of an abstract and a concrete part. The concrete part has not changed much. It still contains bindings, services and ports. In the abstract part messages, operations, and port types have been combined as elements of an interface description as shown in Figure 2.7.

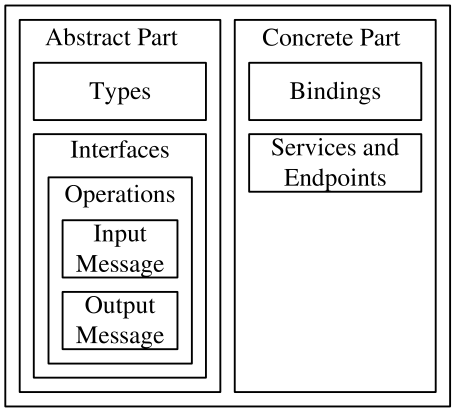


Figure 2.7: WSDL 2.0 service specification.

The operation primitives have been extended and renamed [Wor07d]. Four operation types *in-only*, *out-only*, *robust in-only*, and *robust out-only* are for one-way message exchange. The "robust" messages allow an error message as reply if the recipient cannot process the message as intended. The operation types *in-out*, *out-in*, *in-optional out*, and *out-optional in* are used for two-way communication.

UDDI

UDDI 3.0.2[7] [OAS04] is an XML-based framework that describes which data structures and APIs a Web service registry must implement and offer to support Web service publication and search.

The information that UDDI offers is often compared to the white and yellow pages of a telephone directory [ACKM04]. White pages list companies and organizations, their contact information, the services they offer, and a short human-readable description. Yellow pages categorize service providers and services according to externally defined classification schemes. Additionally, so-called green pages contain technical information about service invocation.

To store these different kinds of information, UDDI uses data structures that are schematically depicted in Figure 2.8. Each information type, *businessEntity*, *businessService*, *bindingTemplate* and *tModel*, has a unique key. A *businessEntity* contains one or more *businessServices*, a *businessService* comprises a list of one or more *bindingTemplates*. All three information types

[7] At the time of writing, the current version of UDDI was UDDI 3.0.2. The current version can always be found under http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

may contain references to one or more *tModels*. *tModels* may be referenced by more than one UDDI entry.

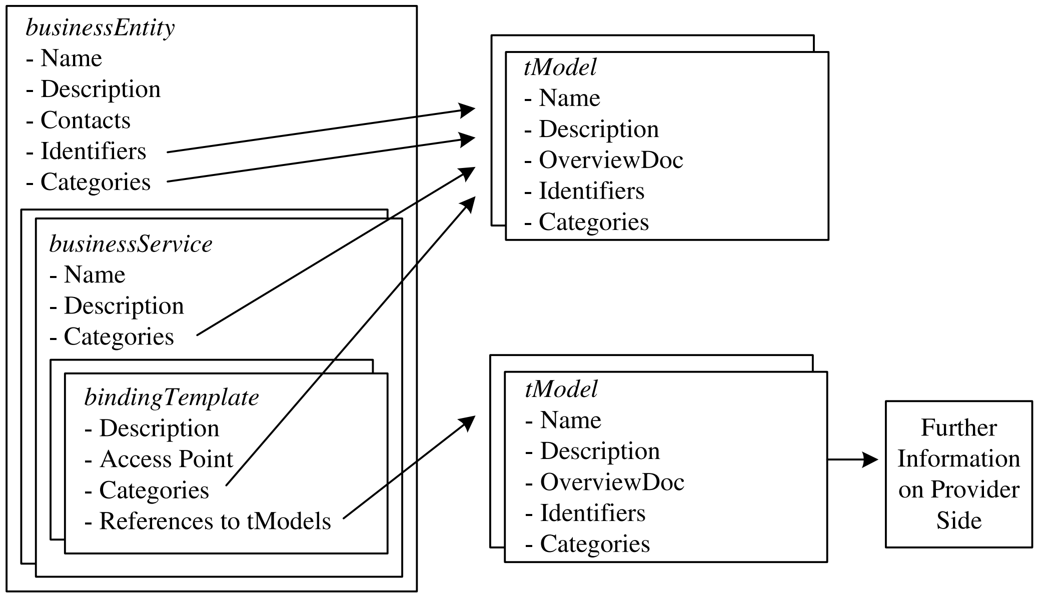


Figure 2.8: Schematic overview of UDDI data structure according to [ACKM04].

The businessEntity contains information about the service provider such as the name of the business, a short textual description, contact information (contact person name, telephone number, or e-mail address), business identifiers such as a tax ID or an ID based on another commonly accepted identification scheme, and a list of categories that describe aspects of the business entity such as industry branch, product category, or geographic region. To define which kind of identification scheme is used, pointers to further technical documentations, the *tModels*, can be included.

The businessService groups related Web services of a *businessEntity*. It will usually contain one Web service only that is available under different addresses, in different version or under different protocol bindings. The *businessService* is described by its name, a short textual description, and a list of categories that can be explained further by references to *tModels*.

The bindingTemplate discloses the technical details to use a particular Web service. This is essentially the access point to the Web service, a short textual description and references to *tModels* that contain further technical information.

The tModel is a generic container for any kind of technical model, e.g., a WSDL file or an industrial classification scheme. They can be referenced by multiple Web services and, thus, be reused. A *tModel* consists of a name, a human-readable description, references to further documentation (e.g., text documents or WSDL files), identifiers that conform to an identification scheme, and categories that follow a given classification.

The UDDI itself offers different APIs that are themselves described as WSDL interfaces. Therefore, a UDDI is a special kind of Web service that communicates via SOAP. The UDDI specification defines the following APIs:

The Inquiry API allows to find registry entries that fulfill given search criteria (*find_business*, *find_service*, *find_binding*, *find_tModel*) and to obtain details about a specific entry (*get_businessDetail*, *get_ServiceDetail*, *get_bindingDetail*, *get_tModelDetail*).

The Publisher API is intended for service providers who want to add, modify, or delete registry entries.

The Security API provides functionality to authenticate and authorize users based on security tokens.

The Custody and Ownership API enables the registry to transfer the custody and ownership to another publisher.

Subscription API offers functionality to track changes within the registry, e.g., new, modified, and deleted entries.

The Replication API supports the synchronization of distributed UDDIs.

Every UDDI provider must offer the Inquiry API and the Publisher API. The other APIs are optional.

2.2.3 Web Service Detection Support

Web Service Registry Scenarios

Different scenarios for Web service registries have been described by the standardization committee OASIS [Ste02], by software vendors such as IBM [Gra01] or Microsoft [Mic03b, Mic03a], and by software designers and researchers [DJMZ05]. The following paragraphs summarize and consolidate these scenarios.

Global Business Registry: The first and most general use case is to operate a UDDI as global, domain independent business registry. In this scenario, the UDDI is used as public registry service, open to every service provider and consumer, free of charge. The UDDI functions as registry as well as as search engine for establishing B2B contacts in an automated way [DJMZ05, Ste02, Gra01].

A proof of this concept was given by the initiating companies of the UDDI standard, IBM, Microsoft, SAP and NTT-Communications. They were operating a global UDDI distributed onto four nodes for five years as shown in Figure 2.9. The UBR was shut down in January 2006 because all companies participating regarded the proof of concept as successful and the UDDI standard version 3 as stable. Since then, companies have started operating separate UDDIs, e.g., uddi.sap.com.

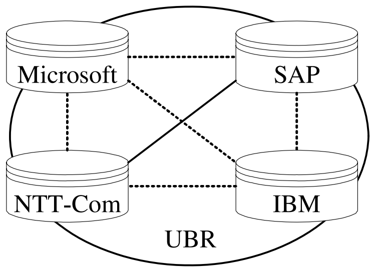


Figure 2.9: UDDI Business Registry according to [DJMZ05].

Domain Specific Business Registry: Instead of offering one global registry for all domains, it is also possible to establish specialized UDDIs that contain entries, all related to one industry branch [Gra01]. Such registries need to establish usage rules that must be adhered to by all users to ensure the quality of the entries and to avoid unwanted entries that are not domain-related. Therefore, such a registry needs an administration that supervises the users [DJMZ05]. This usage scenario is depicted in Figure 2.10.

In such a domain-specific environment, it is also possible to claim a membership fee, to rate the services, and to use established domain-specific taxonomies for service search.

Intranet Service Registry: UDDIs are also used within the Intranet of an enterprise [DJMZ05, Gra01]. Figure 2.11 summarizes this scenario. Application developers use the UDDI to search for available services and as reference list that does not only contain the services, but that also helps to find the related technical documentation [Mic03a]. Further, services within

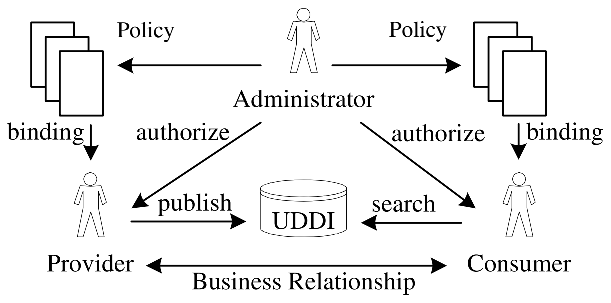


Figure 2.10: Schema of domain-specific UDDI according to [DJMZ05].

the registry can be approved of by the administrator to be used company-wide. This approval process may also be extended to include external services that the application developers are allowed to use [DJMZ05].

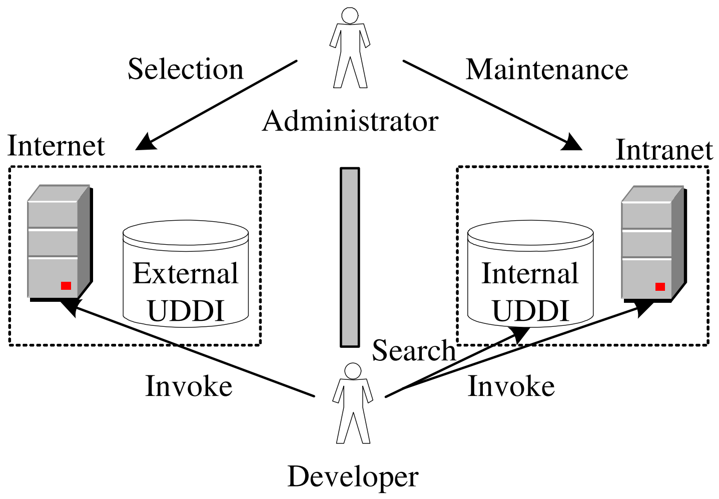


Figure 2.11: Schema of internal UDDI registry according to [DJMZ05].

B2B and EAI Service Registry: The Intranet use case can be extended so that the company does not only use its own and external services but also acts as an external service provider towards trading partners and customers [Ste02, Mic03a, Gra01]. In this scenario, it is necessary to ensure that not all services available on the Intranet UDDI are made public. Only a selection of services is suitable for external use and must be carefully chosen by an administrator[DJMZ05]. This is depicted in Figure 2.12.

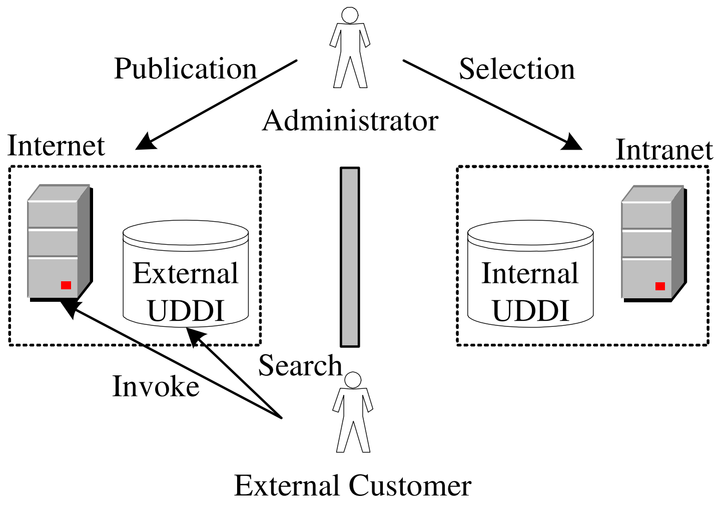


Figure 2.12: Schema of a B2B UDDI according to [DJMZ05].

UDDI in the Software Development Cycle: Throughout a structured software development cycle that follows a process model such as described by CMMI[8] or ITIL[9], three phases of software maturity can be distinguished: the development phase, the test phase, and the operational phase. The three phases are clearly separated. Often, this is also physically the case because the three phases take place on different, separate systems. At least the operational environment is logically and physically separated from the development and test environment as not to interfere with daily business [Ste02]. In many companies the development and test environment are also separated from each other.

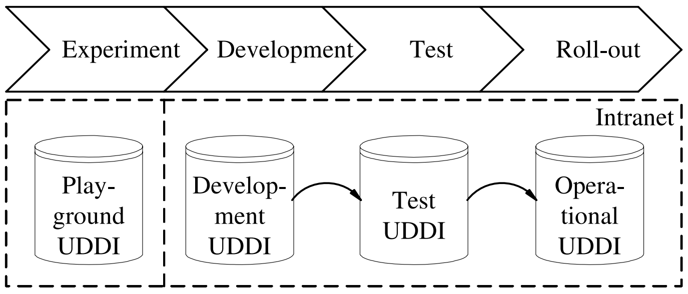


Figure 2.13: UDDIs in software development.

If an Intranet UDDI is used, the different environments imply that a

[8]http://www.sei.cmu.edu/cmmi/cmmi.html
[9]http://www.itil.org/

development, a test, and a production UDDI are needed [Gra01, Mic03a]. This is depicted in Figure 2.13. The development UDDI contains services still under development. The test UDDI contains all services that are to be tested. Finally, all services tested are moved to the UDDI in the operational environment. If developers need to get acquainted with Web services or if they merely want to test and play with new services, there may even be a "playground" UDDI. Such a scenario using four separate UDDIs has been successfully implemented by Qwest [Mic03b].

This last usage scenario is complementary to the Intranet and B2B/EAI scenarios. The focus of this thesis is on the Intranet usage scenario, accompanied by employment in the software development process.

UDDI Search and Invocation Scenarios

UDDI is designed to support different search patterns, the search by the human developer at design-time of an application and the search for services of an application at run-time.

Design-time Search: At design-time, UDDI can be used in a combination of browse and drill-down search patterns [OAS04]. The application designer calls a *find_xyz* function of the UDDI Inquiry API and gets a result list back. This list must be inspected manually or refined through a more specific search. To inspect a particular entry in the result list, a *get_zyxDetail* function is executed. Finally, the designer can invoke the service. This sequence of browse, drill-down, and invocation patterns is depicted in Figure 2.14. Most often, development tools support this exploratory search pattern with a GUI. The results of the search at design-time can be used to bind the service invocation statically into the application.

Run-time Search: Besides static binding the search functionality of the UDDI can also be used for dynamic binding. Three types of dynamic binding must be distinguished: the first type of binding is *fully dynamic*; this means, at run-time, the application searches for a service it has never used before that is offered by a provider that is unknown to the consuming application. The application chooses a service among the results returned by the UDDI, e.g., according to quality of service aspects, and binds to it on the fly. This scenario, although visionary, is considered to be unrealistic for service-oriented application development. The behavior of such an application would be unpredictable. At design-time, it is unknown which parameters need to be passed. The application cannot be thoroughly tested, it is undefined how to react if an exception is returned, and the service consumer does not even have

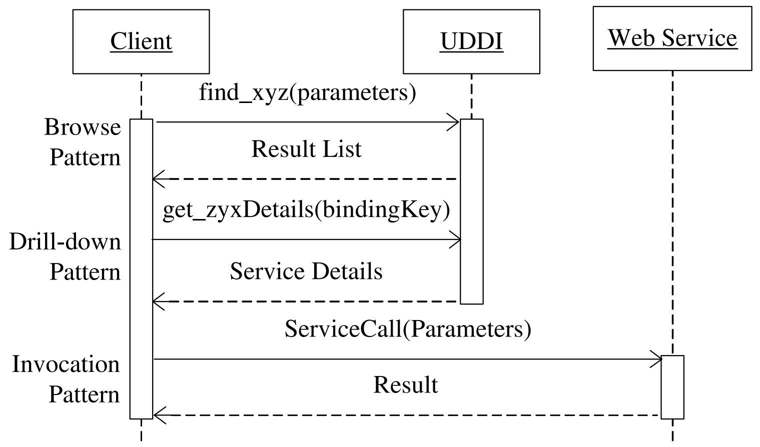


Figure 2.14: Patterns supported by UDDI inquiry API [DJMZ05].

a proper service level agreement with the service provider. [ACKM04] doubt that application designers will ever have use for such a degree of dynamism.

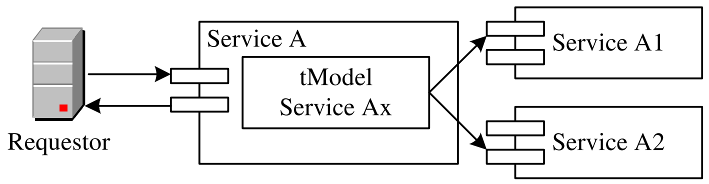


Figure 2.15: Dynamic service call via tModel.

The second type of binding is *semi-dynamic*, i.e., dynamic in a restricted sense. At run-time, the client application calls a service interface as defined by an existing *tModel*. This can be implemented by more than one Web service. Thus, the interface and the binding details are known to the requester a priori. Only the exact service and its provider are chosen at run-time as shown in Figure 2.15. [ACKM04] consider this to be a realistic scenario for Web service development.

A variation of dynamic binding is suggested by [Ley03, WCL$^+$05]: it uses a specialized infrastructure, an *enterprise service bus* (ESB), for service selection. The service requester sends a search request to the service bus and describes the necessary service by defining business goals and policies. All matching services are considered to be equivalent from the requester's point of view. They are represented by a *virtual service*. The service bus chooses an appropriate service, binds to it, invokes it, and delivers the result to the requester. The choice the service bus makes is guided by further policies such as execution costs or availability. This scenario is depicted in Figure 2.16.

The service bus is an infrastructure component that is delivered by software vendors; for example, IBM WebSphere[10] contains a service bus. However, the service bus idea is not widely adopted yet (see Section 2.3.1).

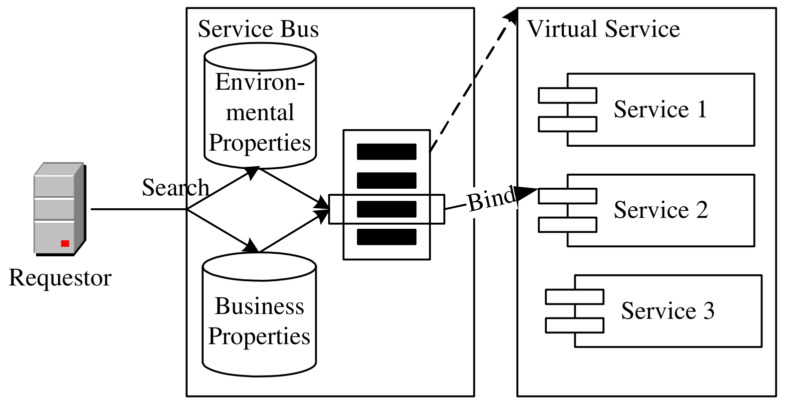


Figure 2.16: Dynamic service selection via service bus according to [Ley03].

The focus of this thesis is on design-time search, which is mostly static, but can also be semi-dynamic.

Implementations and Standards

The most important standard for Web service detection, UDDI, has already been introduced in Section 2.2.2. There are additional standards that address the Web service detection problem in absence of a central UDDI.

WS-Dynamic Discovery: WS-Dynamic Discovery [BKK+05] is a specification that defines a multicast discovery protocol to find services in ad hoc networks where Domain Name Services (DNS) or directory services do not exist. The protocol is not intended for Internet wide usage. It does not define any data model or any queries over rich meta-data. If a service consumer wants to find a service on the network, it sends a probe message to a multicast group. Matching services respond directly to the service consumer. The service consumer then sends a resolution request message to the same group and the matching service replies directly to the service consumer. If new services join the network, they send an announcement to the group to minimize polling. If a discovery proxy joins the network the multicast discovery protocol is automatically abandoned to use the proxy specific protocol. This protocol is suggested to be used for, e.g., discovery of low level resources such as printers within a network.

[10]http://www-306.ibm.com/software/websphere/

WS-Inspection: UDDI is based on the idea of centralized service registries that are in general open to every kind of business. In parallel, other ways of publishing Web services have emerged, e.g., amazon.com Web services are published directly on the provider's Web side. WS-Inspection addresses how to publish Web service in a decentralized, yet structured manner. Thus, clients can search Web services of providers they trust. An overview of the different strategies of UDDI and WS-Inspection is given in Figure 2.17.

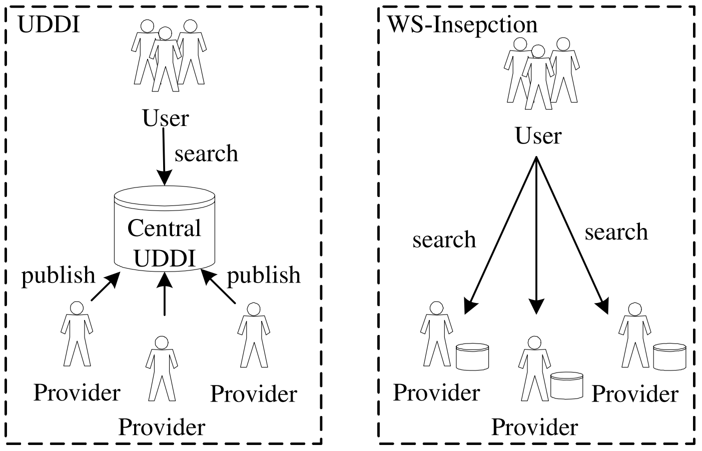


Figure 2.17: UDDI versus WS-Inspection according to [DJMZ05].

WS-Inspection is document-based: the service provider publishes a document with the name `inspection.wsil` in the home directory of the Web server. This document contains an arbitrary number of service elements and an arbitrary number of link elements. Service elements consist of a name and a description that contains technical information such as the namespace and the location of the WSDL document that defines the service interface. Link elements consist of technical information, e.g., the location of another WS-Inspection document. The WSIL document itself, the service elements, and the link elements may be described by short human readable abstracts.

The service consumer reads the `inspection.wsil` document via HTTP and receives a list of all available service and their WSDL files. The WS-Inspection standard is extensible, e.g., there are already pre-defined extensions to include UDDI and WSDL. The link elements permit to build hierarchies of WS-Inspection documents.

Electronic Business XML - ebXML ebXML[11] [OAS01a] is a standardization initiative from the e-business domain that started in 1999 to promote

[11] www.ebxml.org

an open XML-based infrastructure for world-wide, interoperable, secure and consistent exchange of information about electronic business. It supports trade between companies of different sizes and from different geographic areas by offering open XML standards for business-to-business (B2B) and business-to-customer (B2C) transactions that replace proprietary, vertical solutions by a horizontal solution spanning industry and trade. The standard consists of six specifications as depicted in Figure 2.18.

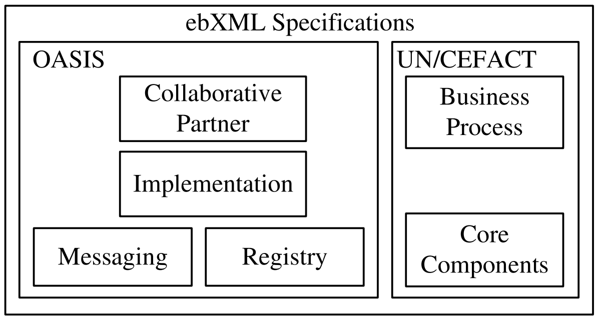


Figure 2.18: ebXML specifications.

These specifications are compatible with the basic Web service standards WSDL, SOAP, and UDDI and define the following standards for e-business [eJC06]:

Messaging Service defines a communication-protocol neutral method to exchange e-business messages over SOAP 1.1. It supports reliable messaging and digital signatures.

Collaboration Protocol Profile and Agreements defines a bridge between technical capabilities and partner expectations for business collaborations. It contains technical capabilities of a business partner and the capabilities and preferences of protocol aspects and properties for specific roles in component services. It enables the monitoring of sessions and verification of delivery channel features. The agreements contain data to configure shared aspects of business collaboration protocols.

Business Process Specification Schema defines a standard language to describe interoperable business processes. It allows to define transaction patterns, partner roles, documents and it helps to compose and monitor processes. The business process can also involve Web services.

Registry Service supports registering, locating, and accessing information resources in a distributed environment.

Core Components Technical Specification presents a methodology for developing semantic building blocks to represent the general business data types that are frequently used. This enables reusable and commonly understandable concepts and data models. It defines a fixed set of data types and naming conventions for consistent business representation.

Implementation, Interoperability and Conformance define standards to realize, e.g., test frameworks for automatic testing of standard compliance.

ebXML registries can be regarded as Web services and can therefore be registered in UDDIs; thus, they become available like any other Web service [MMHM01]. Further, ebXML provides a framework to define core components for e-business that can be re-used in any context, such as the naming of data types in a WSDL document. Thus, the core components help to establish naming conventions and semantics for WSDL documents and UDDI entries [OAS01b].

2.3 Operation Detection in Service-oriented Software Development

Together with Web services, an architecture trend, the “Service-oriented Architecture” (SOA) has emerged. Web services are commonly regarded as one possible implementation vehicle for such an architecture style. Just as the term Web service, the term SOA is not precisely defined. On the one hand, not every application that is implemented using Web services deserves to be described as a SOA. On the other hand, SOAs need not necessarily make use of Web services for implementation. It is even disputable if it is justified to speak of an architecture [Vos06]. For some authors, the term “Service-oriented Integration” (SOI) seems to be more appropriate to describe the changes that currently take place in the software landscape of larger enterprises [Ars05].

In this section, the needs of the software architect within a service-oriented software development project are examined to derive requirements for search support. The focus of this section is on the search support that is needed during a service-oriented development project which includes a study of the problem scope, a conceptual and logical analysis as well as a physical design of the solution.

2.3.1 SOA Overview

Generally speaking, a service-oriented architecture is an abstract concept in which every functionality is described, published, and invoked as a service via a network [WCL$^+$05, DJMZ05]. The primary goal is to support loose coupling of services to allow simple and flexible interaction between applications. This is a long known idea in the design of hardware as well as information systems and other system architectures [LD04].

The Enterprise Service Bus

The current approach for a SOA implementation are Web services. Their standards have emerged in a bottom-up fashion as shown in Figure 2.19. This representation provides a high-level view of the Web service protocol stack. A new standard has been added for every new aspect that occurs when implementing service-oriented applications. Yet, building a SOA bottom-up is neither the recommended strategy suggested by information system research [Vos06] nor the way software architects and application developers want to proceed [Noe05].

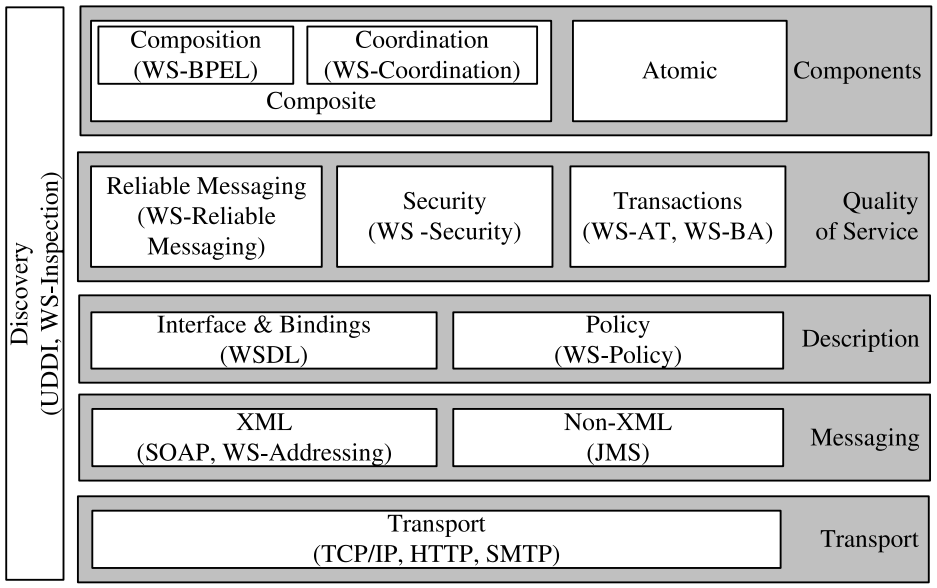

Discovery (UDDI, WS-Inspection)
Composition (WS-BPEL)
Coordination (WS-Coordination)
Composite
Atomic
Components
Reliable Messaging (WS-Reliable Messaging)
Security (WS -Security)
Transactions (WS-AT, WS-BA)
Quality of Service
Interface & Bindings (WSDL)
Policy (WS-Policy)
Description
XML (SOAP, WS-Addressing)
Non-XML (JMS)
Messaging
Transport (TCP/IP, HTTP, SMTP)
Transport

Figure 2.19: SOA protocol stack according to [WCL+05].

Transport, messaging, security, transactional guarantees, and composition are cross-cutting concerns of every SOA implementation. Therefore, these issues need to be dealt with only once. As already mentioned in Section 2.2.3, an ESB may act as “service middleware” [WCL+05, PvdH05].

Vendors such as IBM or SAP already provide ESB implementations in their SOA suites[12], but also “home-grown” decentralized implementations are reported [Bel06]. Although not every description of a service-oriented architecture adopts the service bus as a component in which these protocols reside, the service bus is prevailing in recent architecture discussions [PvdH05, WCL+05]. The ESB itself can be regarded as a service-oriented middleware layer.

Layered Software Architectures

To show how the service-oriented paradigm is logically related to layered IT architectures, the schematic representation of the IT landscape in different logical tiers as in [Bal00, ACKM04] is used. The representation chosen here is generic and distinguishes the operational, the application, the business process, and the presentation layer.

Turning such a system into a service-oriented architecture can be achieved by equipping functions with a service interface and publishing them as services as shown in Figure 2.20. This is not restricted to a certain layer of the different IT systems, but can be used across all layers [WCL+05]. For example, amazon.com[13] offers hard disk storage accessible as a web service, called “Amazon S3”. [Wes05] suggests to use visualization web services to display e-learning content. These examples show that Web services can be exposed at any layer.

Services on the data layer may be simple interfaces to an SQL database. On the application layer, a service may execute a complex computation such as the calculation of the present value of a cashflow, calling other services for input data or auxiliary functionality. On the process layer, services may be more complex than services on the data layer or on the application layer because they provide a higher level of abstraction of the business logic and may be composed of many different services disguised as sub-processes.

The IBM SOA foundation reference architecture also includes further aspects, that “envelope” a SOA: integration layer, quality of service layer, data architecture and business intelligence layer, and governance layer. This is depicted in Figure 2.21.

[12]IBM WebSphere Enterprise Service Bus:
http://www-306.ibm.com/software/integration/wsesb/
SAP NetWeaver:
http://www.sap.com/platform/netweaver/index.epx

[13]www.amazon.com

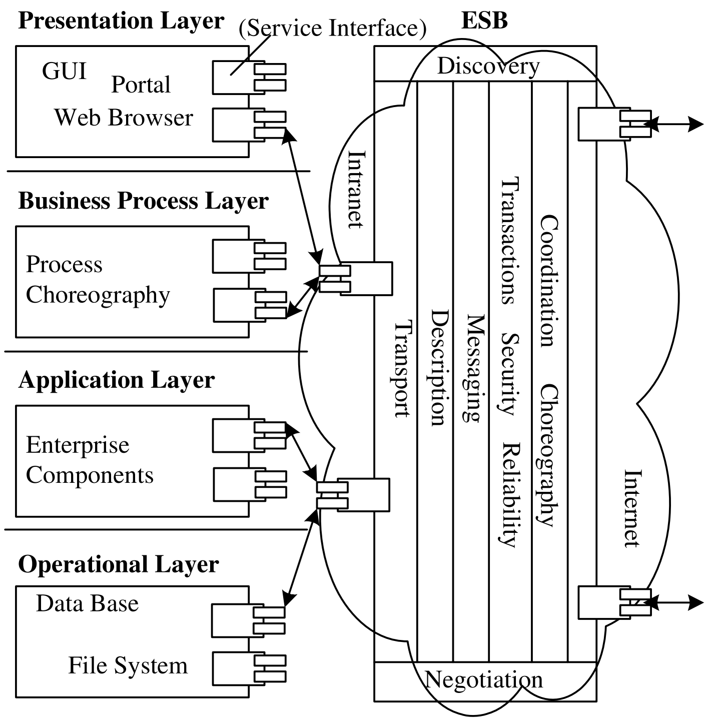


Figure 2.20: Logical tiers of IT architectures with ESB.

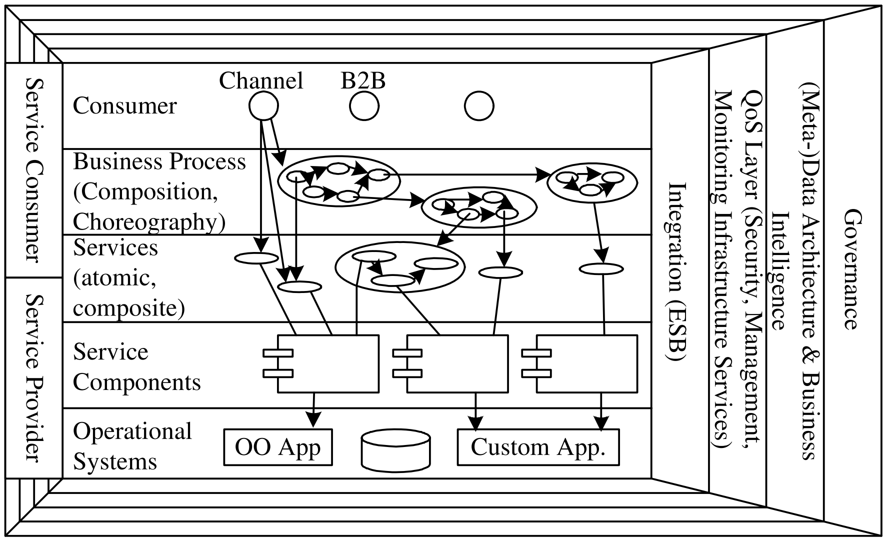


Figure 2.21: SOA foundation reference architecture according to [GAA+06].

From this, the following types of Web services within a SOA are derived:

Infrastructure Services provide intermediary, low-level, mostly atomic, data-centric services such as access to remote hard disc storage, queuing or service wrappers to legacy applications that do not posses generic Web service interfaces.

Information Services offer access to specific information sources and their inherent functionality such as databases or data warehouses with their analytical and statistical functions. Further, information services offer access to performance data including closely data-related analytical functions.

Transformation Services occur as simple conversion services, e.g., currency or time conversion, as adapters that translate between different data formats or as encryption and decryption services. They do not produce new information but simply transform information carrying data. Numerous conversion services can be found on the Internet[14] .

Business Function Services offer access to functionality that is specific for an application domain. These are usually more complex applications [Vos06, Ars04], e.g., a credit pricing tool.

Business Process Services are responsible for the coordinated and controlled execution of partial or complete business processes that are composed of business function services, transformation services, information, and infrastructure services [Vos06, Ars04], e.g., the execution of a loan application.

Visualizing Services can be used to display the results returned by an information service, a transformation service, a business function service, or a business process service to a human user. They are the only pure software-to-human services. Although they are not mentioned very often [Ars04], they are likely to gain more attention in the near future, especially in the e-learning context [Wes05].

State of SOA Adoption in Industry

The examples in the introduction (Toyota Australia, Hewlett Packard, Wachovia Bank, Qwest) show that companies from different industry branches adopt the idea of service orientation for their IT infrastructure. The road

[14]Currency conversion, e.g., http://finance.yahoo.com/currency?u and time conversion, e.g., http://www.timezoneconverter.com/.

towards a service-oriented architecture has been analyzed in different reference projects by Microsoft and IBM. The following presentation of the current state of the art in SOA implementation is therefore a synopsis of [Mic03b] and [AH05] with additional findings of the Aberdeen benchmark report [Kas06].

The introduction of a SOA often proceeds evolutionary in three stages:

Simple Web Services: In this stage, the company creates services from functionality that is available in legacy as well as newly built applications. Mostly small to midsize companies implement light-weight mission uncritical services. The company offers these services internally as well as externally, not necessarily using a UDDI, but simply relying on WS-Inspection or even on direct publication on the Intra- or Internet. In this stage, Web services are hardly more than a web-based remote procedure call.

Composite Services: In this stage, services are integrated across applications inside as well as outside the enterprise to achieve a business objective. The services are aligned with business goals. The primary focus is to abandon cumbersome point-to-point integration using the experiences made in the first stage. An ESB is used for mediation, routing, and transformations of service invocations.

Virtual Infrastructure: In this stage, the IT infrastructure of an enterprise is re-designed. A company-wide service-oriented architecture strategy is in place. The experience with services matures and services are managed throughout the complete software life-cycle.

In mid 2006, 64% of all participants of the Aberdeen benchmark report [Kas06] had ESB software already deployed in their service-oriented architecture. Still 46% used a meta-data repository or a service registry respectively. More than 90% of all questioned companies intended to reach an active level of service programming at the end of 2006. The participants came from the industry sectors manufacturing, consumer goods, services, and the public sector.

2.3.2 Software Engineering Principles for SOA

The development of a SOA follows general modeling principles such as abstraction, encapsulation, modularization, and separation of concerns [LP72]. Different design and development strategies can be chosen. To analyze Web service detection support during the different stages of a service-oriented software development project, an iterative, meet-in-the-middle strategy with four stages of the development cycle is chosen. This is summarized next to serve as a framework for the enclosed case study.

Iterative Development Models

The development of a SOA from individual service to a virtual infrastructure is a project that starts with an existing infrastructure and legacy systems in place. Following modern development models, it is recommended to produce intermediate results in development cycles working iteratively [Bal00]. Two models, the evolutionary and the incremental model, have been found in the literature [Mar06, GAA$^+$06] as recommendations for SOA development.

The incremental model starts with a thorough analysis and tries to capture all requirements as completely as possible at the beginning of the development project. When all requirements are documented, the importance of each aspect is weighed by the end user. Based on this assessment, a project plan is drawn up that determines the sequence in which the different requirements are designed in detail and implemented. The Wachovia Bank has implemented a SOA using an incremental model [Mar06].

The evolutionary model is recommended by IBM [GAA$^+$06]. In the evolutionary model the development proceeds step by step starting with a basic version of the system without having a complete picture of all requirements. The first version is revised and new features are added. This is repeated as long as the system is complete.

Both models (see Figure 2.22) aim at producing early results, encouraging end-user feed back, and reducing development risk because of short and focused development cycles. The evolutionary model has the drawback that the final picture is not known in the beginning and, thus, risking to tear down parts of earlier versions when requirements change. This is avoided in the early requirement analysis of the incremental model.

Top-Down Versus Bottom-Up Strategy

The starting point of a SOA development project is either the identification of functionality that qualifies as a service or the analysis of the existing software systems and their service capabilities. These two approaches correspond to the well-known software development approaches of *top-down* and *bottom-up* system design. A combination of both strategies is known as *fan-out* or *meet-in-the-middle* approach. The development cycles in the evolutionary or the incremental model can follow either strategy. Therefore, the three strategy variations are examined independently of the chosen development model,

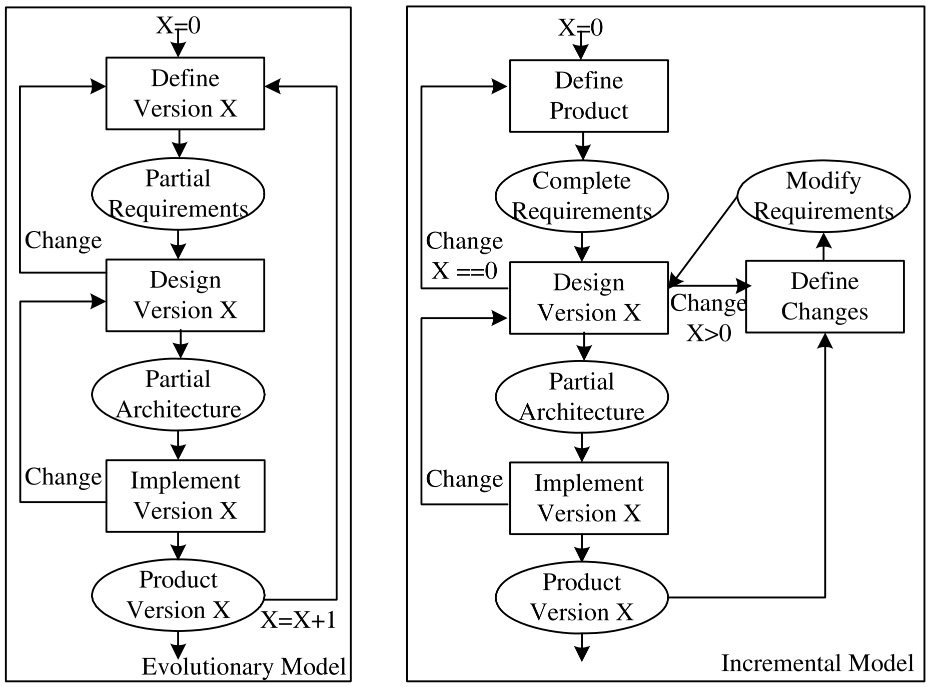


Figure 2.22: Iterative development models according to [Bal00].

looking at a simple service-oriented development scenario, the creation of a new service.

The way to identify a service and to create it is multifaceted. The top-down strategy starts with a model of the business process or the use case that is to be supported. The process is refined step by step into sub-processes and sub-use cases until individual services can be identified. This is repeated until the complete process is modeled as a service choreography. The bottom-up strategy first analyzes the existing systems and the implemented processes that run on these systems. Functionality that qualifies as a service and seems appropriate to contribute to the final software solution is identified. Later, the identified services are integrated into the solution architecture.

In the top-down approach, identified services need to be compared to existing services to find out if existing services can be reused. If this is not the case, existing components must be analyzed to check if the desired functionality is already available in existing systems. Then, this functionality can be exposed as a new service. If the desired functionality does not exist at all, a new service must be created. The top-down approach must be combined with a system analysis to avoid redundant programming of services.

In the bottom-up approach, identified components that qualify as services need to be compared to existing business tasks that need IT support. A mapping between service capabilities and business needs must be conducted to identify gaps. The bottom-up approach must be complemented with a use case analysis to evaluate the need for the service and to achieve integrated service choreographies.

Therefore, starting top-down or bottom-up results in both cases in a meet-in-the-middle design to map services and business needs for IT alignment. This always includes the search for already existing services.

Service-oriented Modeling and Design

So far, it has been established that service orientation supports general software engineering principles, that iterative development models are preferred for service-oriented development, and that the identification of services within such a development model can be pursued in the top-down or bottom-up way. The next step is to introduce a development methodology for service-oriented software engineering that takes the special characteristics of Web services into account. A service has a consumer, a provider and the consumer must be able to find the service. This linkage of perspectives must be reflected in service-oriented modeling and design.

A well established development methodology [Bal00] for top-down development undergoes the following steps:

Requirement Analysis identifies use cases and relevant user groups of the application. Information and processing requirements are collected. The result of the requirement analysis is a requirement specification for the application.

Conceptual Design produces a conceptual model of the application that is independent of logical or physical implementation considerations. It shows the concepts that the application consists of. The result is a conceptual model.

Logical Design transforms the conceptual design into a logical model that reflects the chosen model for implementation.

Physical Design defines the internal representation of the application and the system parameters that are to be observed. Owing to performance considerations, the physical design may differ from the logical design.

This development methodology is especially appealing as it makes a clear distinction between conceptual, logical and physical design. At the same time, the user is not forced to use a specific modeling language or a predetermined physical implementation. For example, this generic design approach is used for object-oriented applications [Oes01] as well as for relational databases [Vos00].

The design process from consumer and provider perspective is summarized in Figure 2.23. The service consumer defines the application scope in the requirement analysis. This serves as starting point for the identification of desired services on the consumer side and for the identification of realizing components on the provider side during the conceptual design. In the logical design step services are categorized by the service consumer and components are specified by the service provider. Both of them need to agree upon the mapping between services and components and the overall layering of the service-oriented application. Here, especially the reuse of existing services must be examined. A mapping between service specifications and existing service implementations must take place. In the physical design step the service consumer decides which services of the new application are to be exposed for future use. The service provider designs the layering and models new components physically. Implementation standards need to be adhered to by both sides.

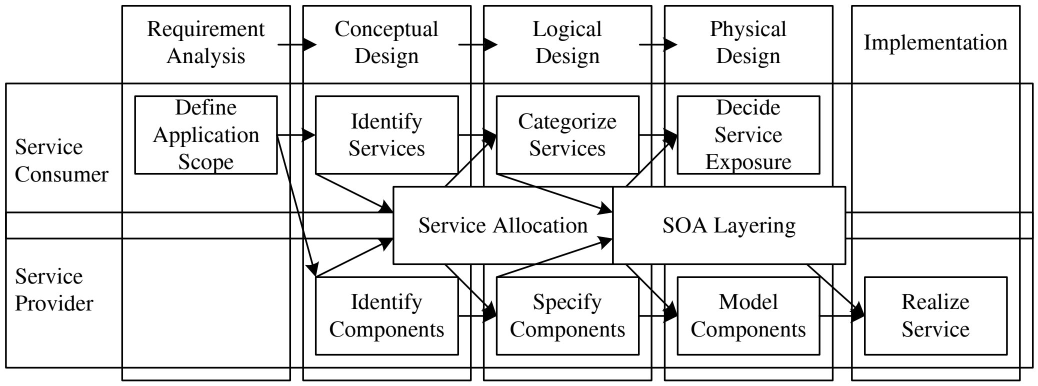


Figure 2.23: Activities of service-oriented design.

In the following, the scenario from Section 2.1 is resumed to show the stages of service-oriented software development by example. The complete use case is documented in Appendix A.2.

2.3.3 Requirement Analysis

The requirement analysis has to identify business goals that drive the creation of a new application, the stakeholders that have an interest in the new application, and the business domains that they represent. This can be derived from interviews and an analysis of the business use cases that the application is to support. The requirement analysis is conducted from the point of view of the end user.

The following business goals are identified for the loan application as shown in Table 2.2.

Table 2.2: Loan application goals.

No.	**Description**	**Type**
1	Increase revenue	general
2	Increase number of consumer loans	general
3	Provide self-service loan application capability	functional
3.1	Provide user-friendly application experience	functional
3.2	Provide customer-specific loan offers	functional
4	Monitor usage of Internet application	functional

In general, the online application process is to increase revenue by increasing the number of consumer loans, which are granted online. Functionally, the application must provide a portal for self-service loan applications. This is to be designed in a user-friendly fashion and must be able to offer customer-specific loans, e.g., by taking the customer scoring into account. Finally, the marketing department wishes to monitor the application. The first two business goals are general. The other goals are application specific functional goals that are likely to have an influence on the application realization and the services needed.

In a second step, candidate use cases are identified that are to be supported by the new application. The following use cases have been identified as described in Table 2.3.

In a third step, the business domains are identified that are involved in the use-cases and in achieving the business goals as shown in Figure 2.24.

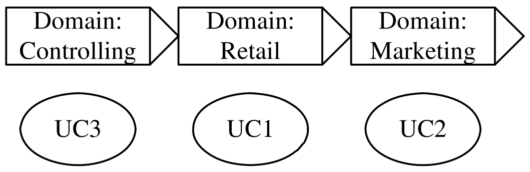


Figure 2.24: Loan application business domains.

Consumer loans belong to the group of retail banking products. Therefore, the customer front end belongs to the retail domain. Monitoring the

Table 2.3: Loan application use cases.

No.	Description	Actors
UC1	The customer applies for a loan on the Internet, fills out the forms and gets a contract by post. This is sent back to the bank and checked by a sales consultant.	Customer, Application, Sales Consultant
UC2	A controller regularly updates the loan conditions used by the application according to the changes in interest rates.	Controller, Application
UC3	A marketing specialist monitors the application.	Marketing Specialist, Application

usage of the application is a technical facility for marketing purposes. The information needed to offer a customer specific loan is provided by the controlling department. They monitor the interest rates and prices at the international banking markets and regularly compute new conditions for consumer loans that have to be used by the loan application.

In this stage, the analysis is conducted without taking existing services into account. Search support for services is not needed in the early requirement evaluation.

2.3.4 Conceptual Design

In the conceptual design step, the use cases of the requirement analysis are refined. From this model, the service provider identifies system components, models the internal behavior of each component and also the flow between components. On the service consumer side, the results of the analysis are used to identify candidate services, the communication between them and other system components (e.g., a GUI), and to map candidate services to candidate components.

For a more detailed analysis, the business activities are modeled that are executed in each use case. The activities of the loan application use case are modeled in the activity diagram shown in Figure 2.25. If an activity is executed successfully, the next activity is executed. If an error occurs, an error report is created. For a better readability the error activities are depicted at the bottom of the diagram. All arrows that do not point to any activity represent error cases and are logically connected with the error diagram.

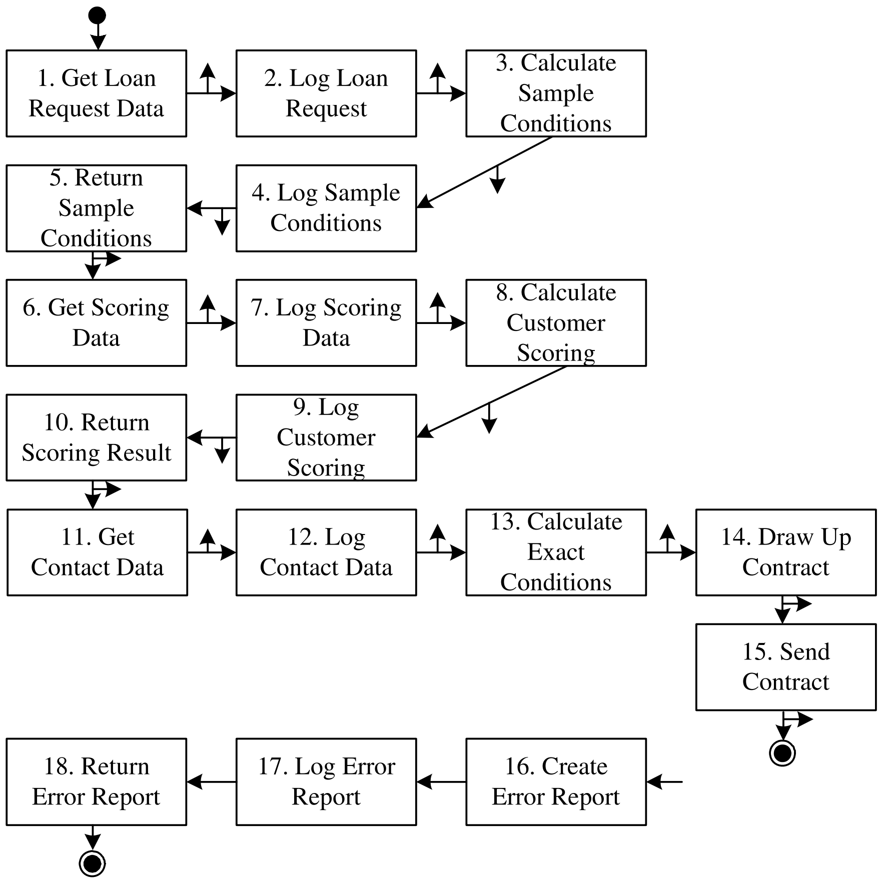


Figure 2.25: Activity diagram for loan application use case (UC1).

In the first step of the application process, the customers state details about the loan they need, the net value of the loan, and the duration. Depending on this information, the monthly installment and interest rate are calculated, while it is assumed that the customers have the best credit scoring possible. Next, the customers have to give details about their private and professional life to assess their credit worthiness using a standardized scoring method. Finally, the customers enter their address data and ask for a loan application form that is then computed for the individual customer taking the exact customer scoring into account. The application form is sent by traditional mail. All interactions between the customer and the online credit application are monitored. This is represented by the logging activities.

In the diagram, at least three components can be identified, a logging component, a component for loan condition calculation, and a component for customer scoring calculation. The creation and mailing of the contract can either be assigned to a mailing component or can be executed by a human being.

The components reflect the service provider view and the logical grouping of functionality from the perspective of an application designer. The services show the external perception of the system by the client, e.g., the loan condition component as designed contains at least two services, the condition look-up service for sample conditions and the condition calculation service for an exact calculation of loan conditions.

Table 2.4: Conceptual components and services.

Component (Service Provider View)	**Service (Service Consumer View)**
Loan Condition Component	Condition Calculation Service Condition Look-up Service
Scoring Component	Score Calculation Service
Contract Component	Contract Creation Service Mail Service
Logging Component	Logging Service Log Evaluation Service

2.3.5 Logical Design

In the logical design step the following questions must be answered for every identified service of the conceptual design step:

- On which layer of the SOA architecture does the service logically reside? Is it a functional or a non-functional service? Is it an atomic service or a composite service? Does it govern a business process?
- Is there an already existing service that can be reused for service realization? How can the desired service be mapped to existing service functionality (top-down)?
- If the service does not exist yet: is there an existing component that can be exposed as service (bottom-up)? Does the service have to be implemented from scratch?

For the subsequent analysis, these questions are structured in a service profile for each identified service as shown in Table 2.5.

Table 2.5: Service profile.

Aspect	**Description**
Functionality	Which business aspects, processes or functions does the service operation support?
Information	What data is sent to the service? Where from? What data is sent by the service? Where to?
Accessibility	Which business process uses the service? Which GUI element uses the service? Which applications access the service?
Process	What is the relationship between the events that the service reacts to and the actions that the service takes?
Interaction	How does the calling application interact with the service? How does the service interact with other services or applications?

To derive service profiles for the first use case, a detailed process diagram is drawn that shows the interaction between the customer at the end user GUI and the application for the first subprocess in the application process, the calculation of sample conditions. The interaction between GUI and Loan Condition Component is shown in Figure 2.26. For a better overview, the monitoring activities are omitted from this diagram.

To get a sample calculation, the customers must enter the net value and the duration of the loan. The net value may range from 1,000 to 60,000 Euros, the duration must be either 12, 24, 36, 48, or 60 months. If the customers enter invalid loan data, they receive an error message and can either abort or try again. If the loan data is valid, the monthly installment and interest

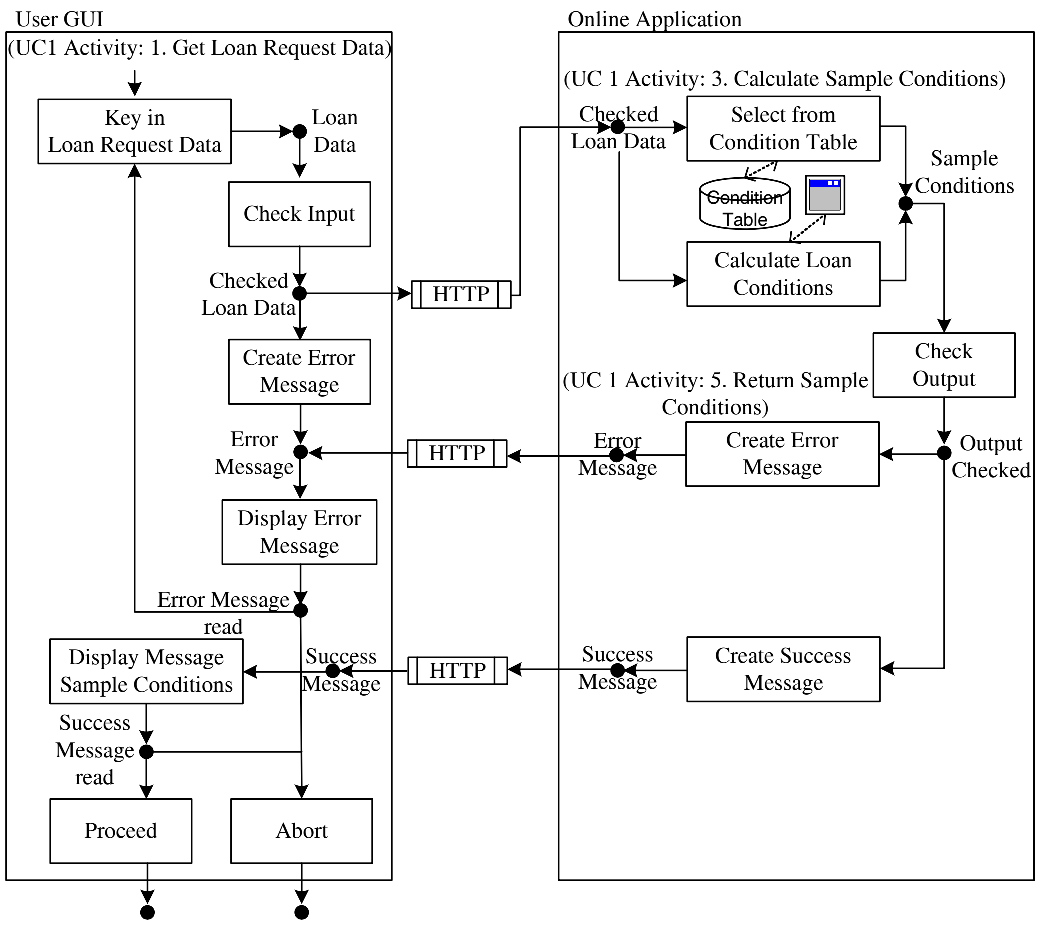


Figure 2.26: Calculation of sample conditions.

rate are looked up in a database table that contains all relevant combinations of net value, duration, and customer scoring. This data retrieval operation assumes an optimal customer scoring by default. If the information is not available in the database, it needs to be computed. If the installment is less than 25 Euros, an error message is created. Otherwise, a success message is sent, and the users are informed about the monthly installment and interest rate. Here, users may abort or proceed.

The complete logical model is documented in the appendix. From the analysis of the subprocesses, the following service profiles are derived that are to be used within the Internet loan application:

Condition Look-up Service

- Functionality: This service calculates the sample and the exact condition for the consumer loan, either by look-up or on the fly. It can be regarded as a business function service.
- Information:
  - Operation name: `get_Sample_Condition`
  - Input: net value, duration
  - Output: net value, duration, interest rate, installment
  - Operation name: `get_Exact_Condition`
  - Input: net value, duration, scoring
  - Output: net value, duration, scoring, interest rate, installment
- Accessibility: The operations are called during the loan application process by the customer for the sample calculation and for the exact calculation. Both calls are executed via the Internet front end GUI.
- Process: On reception of the input data for the operation `get_Sample_Condition`, the sample conditions are fetched from a condition table in a data base. On reception of the input data for the operation `get_Exact_Condition`, the exact conditions are calculated by a financial calculation kernel.
- Interaction: The returned data is displayed on the Internet front end GUI.

Score Calculation Service

- Functionality: This service calculates the private customer scoring based on information about private and professional circumstances. It is a business function service.
- Information:
  - Operation name: `calculate_Customer_Scoring`
  - Input: Date of birth, marital status, number of children, profession, income
  - Output: Date of birth, marital status, number of children, profession, income per month, scoring
- Process: On reception of the input data for the operation `calculate_Customer_Scoring`, the customer score is calculated by a rating application.
- Accessibility: It is called in the loan application process by the customer to prepare the exact calculation via the Internet front end GUI.
- Interaction: The returned data is displayed on the Internet front end GUI.

The service profiles resulting from the logical design step have to be matched with existing services that are already implemented to avoid redundant implementation. UDDI, the technical standard for service publication and search, has several draw-backs trying to search with the information documented in the design-time service profile.

For example, the service provider is not defined in the logical design step. From an internal perspective, a provider is hard to define. [Mic03b] suggest to define the development team that builds the service as its provider. This is only a temporary solution, because at the end of a project, the team is dissolved. The IT department that maintains the service is not a discriminating aspect either because they are responsible for all services. These examples show that the search category "provider", which is one of the main search criteria of the UDDI standard, is not applicable in the internal development context.

Furthermore, the operations `get_Sample_Conditions` and `get_Exact_Conditions` belong together, from a logical point of view, and are logically grouped into one service. However, if there are two implemented operations that fulfill the specification, but are not offered by the same Web service, they are accepted as well. Therefore, the level of a Web service is not an appropriate level for the search. The smallest unit to search for is the operation with its input and output parameters. This level is not available in a UDDI registry. A solution to fill this gap is presented in Section 2.4.

The result of the logical design step is a mapping from needed services to existing services and components and from needed services to functionality newly to be implemented. If the functionality does not exist as a service yet, it must be assigned to components that will later realize the operation in the implementation. Finally, for every identified functionality it must be decided if it is published as a new service for future reuse through other applications.

2.3.6 Physical Design

In the physical design step, it is decided if the software that realizes a new service will be reused from existing components or services, bought, custom built, or maybe even outsourced. Furthermore, technical standards for service realization are decided. If a SOA strategy is implemented, the designers often do not have a technical choice, but have to use pre-defined standards, such as WSDL and SOAP.

Finally, an architecture overview document of the application summarizes the resulting design from all four design phases. Such a document reflects how the new application fits into the existing or aspired overall SOA architecture, as depicted in Figure 2.21. It consists of the following aspects as listed in Table 2.6.

In the physical design process, the components and services that do not exist yet, but have to be created are modeled physically to prepare implementation. The existing services and components are integrated into the physical design.

Also at this stage, additional non-functional services are added that are not needed for the business logic, but must be provided by the SOA infrastructure. For example, in the case of the loan application the data between the customer and the application ought to be sent via a secure channel, e.g., using HTTPS via SSL. This is an example of a non-functional service provided by the SOA infrastructure.

Table 2.6: SOA overview document.

Layer	**Description**
Scope	Which line of business is supported by this architecture?
Operational systems	Packaged applications used by this architecture? Custom applications used by this architecture? Exposure and componentization decisions?
Enterprise components	Which business domain is supported by the component? Which process is supported by the component? Mapping of component to service?
Services	Which services are newly created and exposed? Which existing services are reused? Atomic or composite service realization?
Business process	Which business processes are represented as service choreographies? Which business process is wired into an existing application component? How do services collaborate?
Presentation	Which GUI component uses which service?
Integration	Service level agreements? Quality of service assurance? Security and authentication? Performance restrictions? Technology standards to be used? Monitoring tasks?

2.4 Operation Repository for Service-oriented Software Development

In service-oriented application development finding already implemented functionality is the central concern. The smallest unit of functionality that a Web service has to offer is an individual operation. Section 2.3.5 has revealed that this is the appropriate level of granularity for searching in a development project. Looking at the UDDI standard that has been introduced to publish services and to find them, it is obvious that UDDI does not provide any information about Web service operations as the examples in Section 2.3.5 have shown. At most, it contains a link to a site where operation descriptions are documented. For internal service-oriented development, a UDDI is not the first choice.

An additional component, a Web service operation registry is introduced in this section. The registry contains the information that WSDL files offer. It is designed to answer queries at the level of operations, input and output attributes.

In Section 2.4.1, the data that is technically available in WSDL documents and UDDI entries once a Web service has been created and published is examined and categorized.

In Section 2.4.2, a Web service operation repository will be introduced that fills the gap between the information available in a UDDI and the search capabilities needed in service-oriented application development.

2.4.1 Information Classification

The XML standards WSDL and UDDI represent descriptions of Web service interfaces and Web service registries. They are primarily intended to achieve a technological standard among all software application developers that want to use these technologies. Further, they convey information about each Web service that is implemented with these standards. For the purpose of this thesis, the information elements contained in UDDI and WSDL for Web service description are categorized in three groups:

Functionality-independent information is not related to the service operations, but exists independently, e.g., the information about the contact data of the service provider in a UDDI is functionality-independent information. Such information helps to administrate and identify services, but does not convey any particular information about the service functionality.

Functionality-related information is generated automatically when the service is created, e.g., the operation name, the input and output parameter of an operation, and their data types. This information can be used to identify a service through its operations.

Functionality-describing information contains knowledge about the semantics of the service functionality or the elements that contribute to the service functionality such as the individual service operations. This kind of information explains what the service does. It is usually not generated automatically, but must be added manually, e.g., via the documentation tag in the WSDL file.

All three kinds of information are available in UDDI entries and WSDL documents and can therefore be used for service search. To draw a better picture of the complete information that is available for service search, the most important elements of UDDI and WSDL documents are summarized in Tables 2.7 and 2.8.

WSDL documents contain mainly functionality-related information because they describe the service interface. Development environments allow to create such documents automatically, e.g., from a Java class. Therefore, they contain mostly technical information specifying how to interact with the Web service as summarized in Table 2.7. The first half of the table shows functionality-related and functionality-describing information about a service and its operations as contained in WSDL files. The second half of the table shows structural functionality related information such as the relationship between simple and complex types.

The first kind of information contained in a WSDL 1.1 document is functionality-related naming information. The names of simple and complex data types and the names of the elements of which they consist provide functionality-related information about the type system used by the service. Message names specify which messages a service sends and receives. Operation names reveal which operations a service offers; and the port type name is the name of the service.

For a better understanding, all these names can be annotated with a documentation tag that contains further functionality-describing information. If the programmer does not manually document the elements and explicitly fills the documentation tags, the documentation tags are not generated or remain empty.

Functionality-related structural information is also contained in the WSDL 1.1 document. Complex types consist of simple types; messages consist of types; operations accept an input message, return output messages, and have a fault message. Port types are a grouping of operations.

Table 2.7: WSDL 1.1 information classification.

Functionality-related Information	**Functionality-describing Information**
complex type name	(complex type) documentation
simple type name	(simple type) documentation
element name	(element) documentation
message name	(message) documentation
operation name	(operation) documentation
port type name	(port type) documentation
Functionality-related Relationship	**Functionality-describing Relationship**
complex type - simple types	
message - type	
operation - input message	
operation - output message	
operation - fault message	
port type - operation	

UDDI provides mostly functionality-independent information. The UDDI is used for service publication and search but can also be replaced by a different detection mechanism, e.g., WS-Inspection documents. UDDI entries need not be created for a service and furthermore, they are not automatically generated when a service is created. The information classification for UDDI entries is summarized in Table 2.8. Again, the first half shows the information elements, the second half the information about relationships contained in UDDI entries.

Table 2.8 contains only the non-technical data elements. Keys, URLs and also all binding template information is omitted because the focus of this analysis is on functional information as available in the logical design phase of service-oriented development rather than on technical information. The information of *tModels* is also omitted as they serve as "universal" information provider for *businessEntities*, *businessServices* and *bindingTemplates*. A *tModel* can be anything from a human-readable design document to a WSDL file.

UDDI and WSDL documents do not necessarily contain any functionality *describing* meta information if it is not added by the developer in free-text documentation tags or category bags. Without additional information as provided, e.g., by *tModels* the documentation tags, identifier and category bags are of little use because they lack semantic context information and can

Table 2.8: UDDI 3.0.2 information classification.

Functionality-independent Information	**Functionality-related Information**	**Functionality-describing Information**
businessEntity name	businessService name	businessService description
businessEntity description		businessService categories
businessEntity contacts		
businessEntity identifiers		
businessEntity categories		
Functionality-independent Relationship	**Functionality-related Relationship**	**Functionality-describing Relationship**
businessEntity - businessServices		

only be used for key word search and inspection by a human reader.

2.4.2 Derivation of a Web Service Operation Repository

The preceding analysis has revealed that a UDDI does not contain functionality-related and functionality-describing information that is needed to search for Web service operations during a service-oriented software development project. The better choice are WSDL files, but they are technical specifications and contain information structured to execute service calls. They are not structured to support operation search.

For example, to find out which input data an operation expects, its input message type needs to be determined. The message type itself contains all necessary input attributes. The data types of these attributes are defined separately. This decoupling of type information from message format and operation is intended to support modular reuse of data types, message types, and operations, but it is not suitable to support operations search based on the operation itself, its input and output attributes.

Therefore, a new component for a service-oriented architecture is introduced, a Web service operation repository that contains all information directly related to Web service operations. The following pieces of information are needed:

- the Web service name,
- the operation name,
- the parameter names,
- the distinction between input and output parameters,
- the parameter data types.

Further, the logical relationship between these elements is needed:

- the grouping of operations into Web services,
- the grouping of parameters as input of a Web service,
- the grouping of parameters as output of a Web service.

This information is stored in a table structure as depicted in Figure 2.27. The figure uses an ER-diagram as notation. This representation with all its abbreviations (PK, FK) is explained in more detail in Section 3.2.1. Each component, Web service, operation, and data type has a name, which is assumed to be unique. In practice, this uniqueness assumption is realized using identifiers, e.g., unique URLs. The relationship between Web services and operations is represented in a separate relation that consists of the Web service name and the operation name only. The relationship between parameters and operations is represented with additional information. A parameter has a type that is either "in" or "out", a name, and a data type.

The central operation is to search for Web service operations based on their name and their input and output parameters. Search operations will occur more often than insertions, updates or deletions. Therefore, the abovementioned data model is denormalized to provide an integrated view on the components of a Web service operation as shown in Figure 2.28. Artificial IDs have been omitted in this representation. In the following examples, the names are assumed to be unique. Table 2.1 in Section 2.1.1 is an example of this registry structure. In this table, repeating values have been omitted.

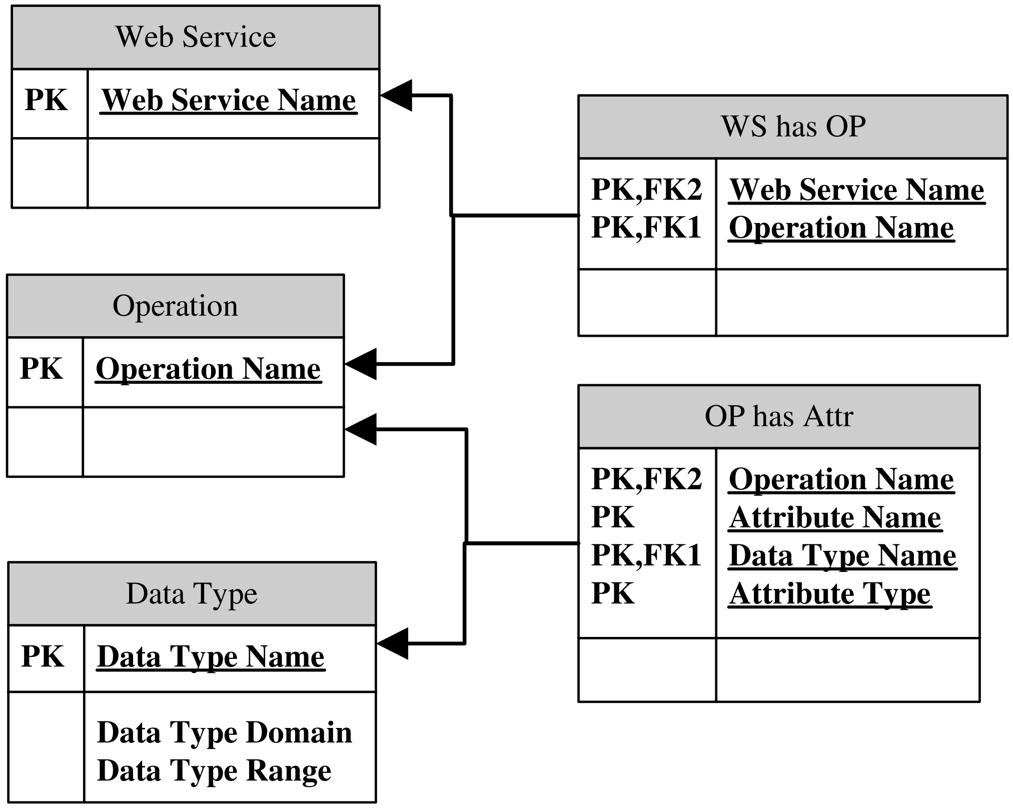


Figure 2.27: Normalized Web service registry.

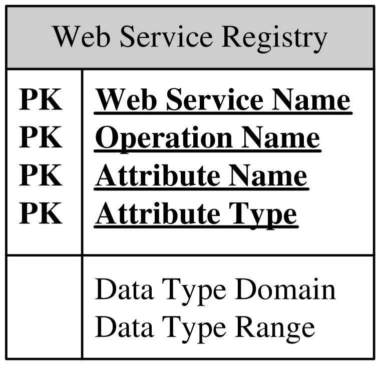


Figure 2.28: Denormalized Web service registry.

2.5 Summary

In this chapter, the process of service-oriented design has been analyzed to find out which search support for Web services is needed in an intra-enterprise development project. The example that has been chosen, is a real life example from the financial industry. It has been modeled and presented, following a generic four step development approach of requirement analysis, conceptual, logical, and physical design.

The analysis has identified the logical design step as the development step that needs crucial Web service detection support to avoid redundant development of functionality. In this design stage, the search is conducted in an Intranet setting; it is executed manually or semi-automatically to support static or semi-dynamic binding of services into the final application. The search takes place at the level of operations.

The technical standard for Web service detection is UDDI. A UDDI contains mostly functionality-independent information. An analysis of the different usage scenarios for this standard has revealed that it is technically possible to use a UDDI in an Intranet setting, but that the standard is first of all designed to support global B2B Web service detection, based on the provider and its line of business. This is not appropriate for internal service-oriented software design.

Therefore, the interface description of a service as represented by a WSDL document needs also to be taken into account for service detection. This interface description is the only document that will always be generated if Web services are used. It provides functionality related information. Functionality-describing information that contains application semantics must be attached either to the UDDI entry or to the WSDL document using additional descriptive tags. This information is either human-readable or needs an additional semantic context to be used in service detection.

As an additional component, a relational Web service repository has been introduced that gathers the information as offered in a WSDL document. The repository offers access to information about Web service operations that is adapted to the search perspective, not the implementation perspective. The relevant information for operation search as available in WSDL documents is stored and presented to fit the logical design step.

In the following chapters, different approaches to search within this registry will be introduced and analyzed. The first approach is primarily syntactic, based on the operation signature. The second approach enhances the first with additional semantic information.

Chapter 3

Syntactic Matching of Web Service Operations

In this chapter, a syntactic match approach based on information in the Web service operation repository is examined. The approach makes use of schema matching approaches that stem from schema matching of relational databases. The example from the financial industry is resumed to illustrate the approach.

Schema matching for relational databases operates on structural, externally specified information. This perspective will be adopted for Web services. To do this, a Web service specification is needed that captures the information that is available from the outside, i.e., when looking for a Web service as a consumer who wants to integrate the service into an application under his control. This is the perspective that the relational Web service repository offers in a compact form.

In Section 3.1, the example from financial industry is resumed. The relational Web service repository as introduced in Section 2.4 is refined and formally defined. Based on the introductory scenario the key research questions for this chapter are derived.

In Section 3.2, foundations of relational schema matching are introduced. They are needed in the following sections as framework and underlying basis. Different algorithms are presented that will serve as examples in the following sections.

In Section 3.3, a three-step approach to match relational Web service schemata is presented that makes use of the algorithms as presented in the section before. The approach offers a syntactic solution to operation and attribute matching.

In Section 3.4 the match results of the section before are analyzed. The notion of a *useful* match is introduced and the match is operationalized. The result is used to define a match hierarchy.

In Section 3.5, the results of this chapter are summarized while answering the second group of research questions of Section 1.2.

3.1 Motivation for Syntactic Matching

In the following sections, syntactic Web service matching is motivated by example. Section 3.1.1 resumes the running example from the financial industry. Different syntactic search queries will be presented. Section 3.1.2 refines the definition of a relational Web service operation repository over which the search queries will be evaluated. Section 3.1.3 outlines the guiding research questions for this chapter.

3.1.1 Scenario Overview

Consider, for example, the Web service that returns the loan conditions for a loan of a given amount and a given duration for a customer with a given score as introduced in Section 2.1.1. Calling the operation `get_condition` for a loan of 2,000 Euros for 12 months and a customer with a score of 2 results in sending a tuple $(2,000, 12, 2)$ to the Web service operation and receiving an output tuple, e.g., $(0.05, 175)$, 5% interest and 175 Euros installment, as result.

In this example, a Web service is regarded as a collection of operations each one with its own dedicated input and output specification as depicted in Figure 3.1. The input and output messages are represented as rows in a table. Incoming Web service calls are shown as a row in the input table. The result consists of a row in the output table. The technically necessary connection between input and output messages is omitted.

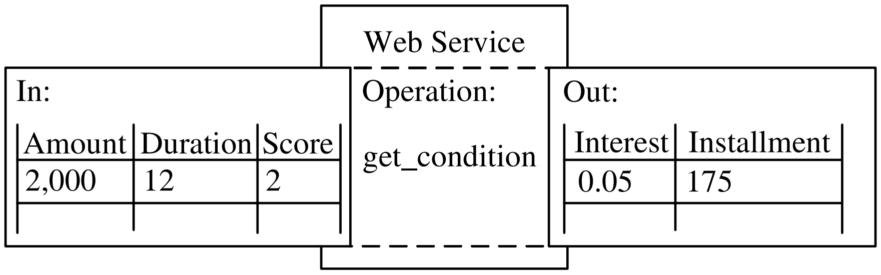


Figure 3.1: Web service specification model.

Matching a Web service schema at the depicted syntactic level results in computing three matches between the designed and the implemented Web service operation: one match on the level of operation names, one match for the input attributes of an operation, and one match for the output attributes as presented in Figure 3.2. As the input message is defined for the designed Web service operation, the match is computed from the designed to the implemented input relation. The output message is defined by the implemented Web service schema. Therefore, the match is computed from the implemented result to the intended design. Depending on the match direction the relation that is mapped is called *source* and the relation that it is mapped to is called *target*. To pursue the match approach as sketched in this section, a formal model of a Web service operation is needed that serves as foundation for a matching algorithm. The model is presented in the next Section. Examples of algorithms that are suitable for match computation are presented and analyzed in Sections 3.3 and 3.4.

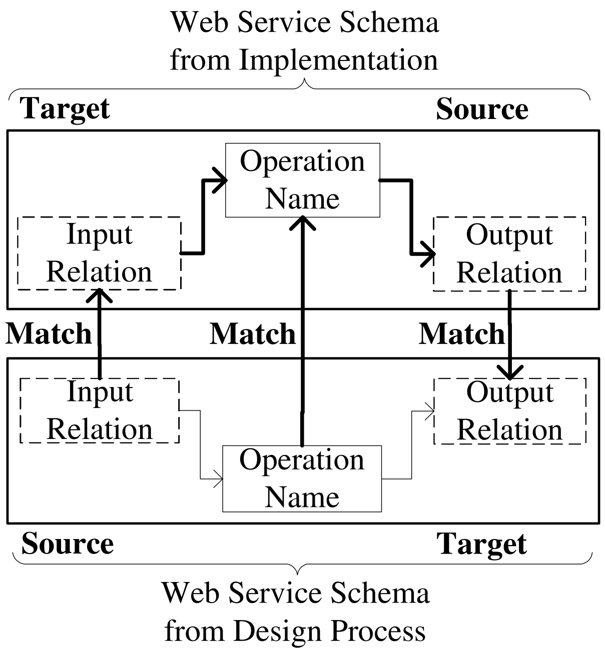


Figure 3.2: Three-step syntactic match.

3.1.2 Relational Web Service Model

A *relational Web service* consists of operations that are described by their names as well as their input and output attributes. As the sets of input and output attributes will be regarded as input and output relations, the model presented here is called a *relational* Web service model.

Relational Web Service Definition

To be able to search for a Web service, a Web service description is needed that is independent of its technical implementation and that provides enough information to decide if it matches the operation searched for. In correspondence with other approaches that model Web services as, e.g., relational transducers [AVFY00] or data-driven Web services [DSV04] (see Chapter 5), a logical model based on relational algebra is introduced here to describe a logical Web service schema that is used for match computation later.

Let $\mathcal{S}$ be an enumerable set of Web service names, $\mathcal{O}$ be an enumerable set of operation schema names, and $\mathcal{A}$ an enumerable set of attribute names. Let $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{A}$ be disjoint.

Definition 3.1. A *Web service schema* is a tuple $WS = (sname, \mathbf{O})$ consisting of a schema name $sname \in \mathcal{S}$ and a finite set $\mathbf{O} \subseteq \mathcal{O}$ of operation schema names.

An *operation schema* is a tuple $OS = (opname, \mathbf{A})$ consisting of an operation name $opname \in \mathbf{O}$ and a finite set $\mathbf{A} \subseteq \mathcal{A}$ of attribute names.

An attribute $A \in \mathbf{A}$ has a non-empty value domain $dom(A)$, a type $type(A) \in \{in, out\}$, and, optionally, an input constraint $constr_{in}(A)$ or an output constraint $constr_{out}(A)$, depending on the type of A.

It is assumed that domains are discrete, totally ordered sets of elements that belong to a defined data type. In the following analysis, the focus will be on built-in data types, e.g., as defined by the XML schema data type definition [Wor04i]. Boolean, integer, float, string, date, time, and datetime are possible domains for attributes in the following examples. These data types are either built-in primitive types such as boolean or built-in derived types such as integer. Excluding user defined types is not a severe restriction as user defined types ultimately rely upon built-in data types. Match computation can easily be extended to include user defined types as it will only operate on domain names and constraints, but not on the domain elements itself.

Every attribute domain includes the special value *null*. The type of an attribute is an element $t \in \{in, out\}$ indicating if the attribute is an input or an output attribute of an operation. If an attribute A occurs as input and output attribute, it is listed twice in the schema definition, once as attribute A of type *in* and once as attribute A of type *out*. Therefore, the attribute type $in - out$ is not needed. A constraint of an attribute A is an element c of the form $A\Theta a$, $a \in dom(A)$ and $\Theta \in \{<, \leq, >, \geq, =, \neq\}$, or a combination of such conditions combined by $\neg, \vee, \wedge$. If attribute A is of type *in*, the constraint is an input constraint. If attribute A is of type *out*, the constraint is an output constraint.

An example is given in Table 2.1 in Section 2.1. In this example, $WS = (Loan\_Pricing, \{Get\_conditions\})$ and $OS = (Get\_conditions, \{Net\_Value, Duration, Score, Interest, Installment\})$. The types of the different attributes, their domains, and their constraints are shown in Table 3.1.

Table 3.1: Attribute details for consumer loan pricing service.

Attribute Name	**Attribute Type**	**Data Type Domain**	**Data Type Range**
Net_ Value	IN	INT	not null $\wedge \geq 1,000$ $\wedge \leq 60,000$
Duration	IN	INT	not null $\wedge$ (= 12 $\vee$ = 24 $\vee$ = 36 $\vee$ = 48 $\vee$ = 60)
Score	IN	INT	not null $\wedge$ (= 1 $\vee$ = 2 $\vee$ = 3 $\vee$ = 4 $\vee$ = 5)
Interest	OUT	FLOAT	not null $\wedge \geq 0$ $\wedge \leq 1$
Installment	OUT	FLOAT	not null $\wedge \geq 25$

Relational Web Service Model Characteristics

The relational Web service model as presented above assumes a *black-box perspective*. This means, in this model, the internal logic of a Web service is not considered to be relevant to know from a consumer's point of view. The consumer calls a service to outsource the service functionality. This means he needs to know the Web service name, the operation name, and the input and output of the operation. This is structural information abstracting from any more detailed technical information such as port types.

The relational Web service model captures the information that is relevant for data exchange. Business related information such as costs or the provider's name are not included. The primary concern is to model syntactical information that is essential for the integration into a service-oriented application.

Further, Web services in this model are considered to be *stateless*. Internal information about the state of a service execution is not modeled. Therefore, the model does neither capture any constraints on the execution order of operations belonging to the same Web service nor any internal temporary attributes that might be needed for internal processing purposes, e.g., to store intermediate results.

The underlying assumption of uniqueness of names for Web services, operations, and attributes is realistic and technically realizable when using the XML namespace [Wor06] concept. This means that the underlying assumption of a unique universe of names is compatible with the current XML-based implementation standard.

Storing a relational Web service definition in a relational Web service operation repository as presented in Section 2.4.2 and as shown in Table 3.1 can be regarded as reification of the relational Web service schema.

3.1.3 Research Questions

As the example in Table 3.1 has already shown, information about Web service operations as specified in WSDL files can be represented in a relational table. This kind of schema information repository is formally examined in the next sections answering the following research questions from Section 1.2:

- Can match computation strategies from relational data bases be reused efficiently? Which existing techniques can be adapted to compute a match?

- What is the nature of a match between a service specification that results from software design and an implemented service?

- How can the match between service specification and service implementation be operationalized and assessed? How complex is the match computation?

- How can a match be categorized to express a gradually increasing degree of compliance between specification and implementation? How is the match evaluated in a qualitative way?

To answer these questions, the relational Web service operation representation as defined in Section 3.1.2 is used to adapt matching techniques from relational databases to the problem of Web service operation matching. An extension to relational algebra expressions will be defined to describe and evaluate the resulting match.

3.2 Foundations of Relational Schema Matching

In this section, fundamental concepts and the basic terminology of schema matching for relational databases are introduced. This is the foundation for the matching approach as presented in Section 3.3.

Schema matching is a research problem that is encountered in different areas of computer science such as databases systems and data warehousing [RB01], model management [Mel04], and ontology engineering for semantic Web applications [DMDH04].

The terminology used to describe the tasks involved in schema matching is not precisely defined, e.g., the terms "matching" and "mapping" are often used as synonyms. For a clear distinction of the different aspects and application contexts of schema matching the following terminology according to [RB01] will be used throughout the remainder of this thesis:

Finding and describing the correspondences between the elements of two schemata is called *schema mapping*. A mapping between two schema elements can be expressed as a triple of the form (*element schema*$_1$, *element schema*$_2$, *mapping expression*). The mapping expression describes how the elements of the two schemata are related, e.g., *element schema*$_1$ = *element schema*$_2$. Many algorithms for mapping computation also compute a similarity measure to indicate, how likely it is that the mapping expression holds. In this case, a mapping is a four-tuple (*element schema*$_1$, *element schema*$_2$, *mapping expression*, *similarity measure*). The manual, semi-automatic, or fully automatic computation of mappings between two or more schemata is called *schema matching*. A *match* between two schemata is defined as an operation that takes two schemata as input and returns a set of mappings as output.

Schema matching is often used to map independent schemata onto one global schema. This is called *schema integration*. If the global schema does not exist a priori, but is computed as a minimal superset of the individual schemata, this is also called *schema merging*.

Once a match between the schemata has been computed, either one schema must be transformed to resemble the other or the data from one schema has to be transferred into an instance of the other schema. The computation of schema transformation instructions is called *schema transformation*; the computation of instance transformation instructions is called *data integration* or *data transformation*.

Schema matching approaches in relational databases make extensive use of the characteristics of the relational data model. Therefore, Section 3.2.1 introduces basic concepts and terminology of relational database theory, first. This terminology is adhered to in the remainder of this thesis. In Section 3.2.2, an overview of relational schema matching, basic techniques, and their classification is given. Section 3.2.3 presents selected matching algorithms for relational schemata and gives examples of implemented matching systems, ranging from research prototypes to quasi-industrial strength tools.

3.2.1 Relational Database Basics

An extensive overview on relational database theory can be found in [AHV95, Vos00]. For the following summary of database fundamentals the terminology is based on [Vos00].

Definition of Databases

Attributes and *relations* form the nucleus of relational databases. Every attribute has an associated non-empty, possibly infinite value *domain*. A *relation* is a finite set of tuples over the value domains of a finite set of attributes. According to [Vos00], this is formally defined as follows:

Definition 3.2. Let $X := (A_1, \ldots, A_n)$ be a finite set[1] of attributes with non-empty domains $dom(A_i)$, $1 \leq i \leq n$ and $dom(X) := \bigcup_{A \in X} dom(A)$ be the set of all values from these domains.

A *tuple* over X is a total function $\mu : X \rightarrow dom(X)$ for which holds: $(\forall A \in X) \; : \; \mu(A) \in dom(A)$. The set of all tuples over X is denoted as $Tup(X)$.

A *relation* r over X is a finite subset of tuples over X, $r \subseteq Tup(X)$. The set of all relations over X is denoted as $Rel(X)$.

[1]Although the notation suggest a sequence, it is a set because duplicates are not permitted and the ordering is unimportant. The notation is kept because it is common.

The time-invariant structure of a relation is described by its *relation schema*, that is the set of attributes and their value domains. To restrict the valid range of tuples over the value domains of attributes, so-called *intrarelational dependencies* are introduced and included in the relation schema.

Definition 3.3. Let X be a set of attributes with associated domains as described in Definition 3.2.

An *intrarelational dependency* $\sigma : Rel(X) \rightarrow \{true, false\}$ is a partial mapping that assigns a truth value to some relations over X. The set of all dependencies over X is denoted as Σ_X. A relation $r \in Rel(X)$ fulfills Σ_X, denoted $\Sigma_X(r) = true$, if $\sigma(r) = true$ for all $\sigma \in \Sigma_X$.

A *relation schema* $R = (X, \Sigma_X)$ consists of a name R, an attribute set X and a set Σ_X of intrarelational dependencies [Vos00].

An example of an intrarelational dependency is the key dependency. A key is a subset of the attributes of a relation for which the following properties hold:

Definition 3.4. Let X be a set of attributes with associated domains as described in Definition 3.2 and $K \subseteq X$.

K is called a key for $r \in Rel(X)$ if the following holds:

1. $(\forall \mu, \nu \in r) : \mu[K] = \nu[K] \Rightarrow \mu = \nu$; $\mu[K]$ denotes the restriction of tuple μ to the attributes of K.

2. $\neg\exists K' \subset K$ so that the first condition also holds for K' [Vos00].

This means a key is a minimal subset of attributes that functionally determines all attributes in the relation. A relation can have more than one key. Therefore, all keys of a relation are called *candidate keys.* Among these keys, one key is designated as *primary key.*

A *relational database* consists of a finite set of relations over their relation schemata. This is formalized in the following definition according to [Vos00]:

Definition 3.5. Let $\mathbf{R} = (R_1, \ldots, R_m)$ be a finite set of relation schemata $R_i = (X_i, \Sigma_i), 1 \leq i \leq m$.

A *relational database* d over $\mathbf{R}$ consists of a set of relations $(r_1, \ldots, r_m)$, $r_i \in Rel(X_i)$ and $\Sigma_{X_i}(r_i) = true$, $1 \leq i \leq m$.

The set $\mathbf{R}$ is called *database schema.* Dat($\mathbf{R}$) denotes the set of all databases over $\mathbf{R}$.

The range of valid instances of a relational database schema can be restricted further by using *interrelational dependencies.* Most often such dependencies are used to express *inclusion dependencies* as formalized in the following definition according to [Vos00]:

Definition 3.6. Let $\mathbf{R}$ be a finite set of relation schemata, $R_i, R_j \in \mathbf{R}$, $R_i \neq R_j$, $R_i = (X_i, \Sigma_i)$, $R_i = (X_j, \Sigma_j)$. Let V be a sequence on n distinct attributes of X_i and W a sequence of n distinct attributes of X_j.

An inclusion dependency $R_i[V] \subseteq R_j[W]$ within a database instance $d \in \text{Dat}(\mathbf{R})$ is defined as follows:

$$(R_i[V] \subseteq R_j[W])(d) := \begin{cases} 1 & : \quad \{\mu[V] \| \mu \in r_i\} \subseteq \{\mu[W] \| \mu \in r_j\}\,, \\ 0 & : \quad \text{else.} \end{cases}$$

If W is a primary key of the relation schema R_j and $(R_i[V] \subseteq R_j[W])(d)$, then the inclusion dependency is called a *foreign key relationship*.

An example of a database schema is shown in Figure 2.27 in Section 2.4.2. In the relational representation of the different elements of a Web service operation repository, there are three tables representing Web services, operations, and data types. The data type relation has two more attributes, domain and range. It is assumed that Web services, operations and data types are uniquely identified by their names. The grouping of operations into Web services and the assignment of input and output attributes to operations is represented in the relations `WShasOP` and `OPhasAttr`. Both Web service name and operation name in the relation `WShasOp` are foreign keys. The two attributes together form the primary key of the relation `WShasOP`. An attribute is uniquely defined by its operation name, its attribute name, its data type name, and its attribute type, which is either `in` or `out`.

A concise notation, also used in the following for the graphical representation, is given below for the table `OPhasAttr`:

Code 3.1.

```
OPhasAttr((Operation Name, Attr Name,
           Data Type Name, Attr Type),
   Primary Key (Operation Name, Attr Name,
                Data Type Name, Attr Type),
   Foreign Key (Operation Name) references Operation (
      Operation Name),
   Foreign Key (Data Type Name) references Data Type (
      Data Type Name));
```

Relational Algebra Operators

For data manipulation in relational databases, the relational algebra (RA) has been defined by [Cod70]. The algebra contains the operators projection, selection, renaming, join, union, difference, and intersection. First, operations on one relation are defined according to [Vos00].

Definition 3.7. Let X be a set of attributes, $r \in Rel(X)$ and $X \supseteq Y = (Y_1, \dots Y_j)$. Let F be a condition of the form $A\Theta a$ or $A\Theta B$ with $A, B \in X$, $a \in dom(A)$ and $\Theta \in \{<, \leq, >, \geq, =, \neq\}$ or a combination of such conditions combined by $\neg, \vee, \wedge$. Let $D \notin X - \{A\}$ and $dom(A) = dom(D)$.

- $\pi_Y(r) := \{\mu[Y] | \mu \in r\}$ is called *projection* of r on Y.
- $\sigma_F(r) := \{\mu \in r | F(\mu) = true\}$ is called *selection*.
- $\rho_{D \leftarrow A} :=$
  $\{\mu \in Tup(X') | \exists \nu \in r : \mu[D] = \nu[A] \wedge \mu[X' - \{D\}] = \nu[X - \{A\}]\}$,
  $X' := (X - \{A\} \cup \{D\})$ is called *renaming* of attribute A from A to D. As short-hand notation $\rho_{A_{|D}}$ is used in the following.

Relational algebra expressions can also be expressed as SQL queries: Let $R = (A_1, \dots, A_n)$ be a relation schema. The projection of a relation r over R on, e.g., $A_1,\ A_2,\ A_3$ is expressed as:

Code 3.2.

```
select distinct A1, A2, A3 from r;
```

A selection of this schema, e.g. $A_1 = a$, a constant, and $A_2 = A_3$ is expressed as:

Code 3.3.

```
select distinct * from r
 where A1 = a and
       A2 = A3;
```

Selecting A_1 and renaming it as B is expressed as:

Code 3.4.

```
select distinct A1 as B from r;
```

Next, operations on more than one relation are defined according to [Vos00]:

Definition 3.8. Let X_1, X_2 be attribute sets, $r_1 \in Rel(X_1)$, $r_2 \in Rel(X_2)$.

$$r_1 \bowtie r_2 := \{\mu \in Tup(X_1X_2) |\ \mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2\}$$

is called *natural join* of r_1 and r_2.

Note that the definition of a natural join operator can be generalized for n relations. If X_1 and X_2 do not have any attributes in common, the natural join is in fact a Cartesian product. If X_1 and X_2 have identical attributes, the natural join becomes an intersection.

Common variation of the natural join are theta-joins as well as left and right outer joins.

Definition 3.9. Let X_1, X_2 be disjoint attribute sets, $r_1 \in Rel(X_1)$ and $r_2 \in Rel(X_2)$, $A_1 \in R_1$ and $A_2 \in R_2$, $\Theta \in \{<, \leq, >, \geq, =, \neq\}$.

$$r_1 \bowtie_{A_1 \Theta A_2} r_2 := \{\mu \in Tup(X_1X_2) |\ \mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2 \wedge \mu[A_1]\Theta\mu[A_2]\}$$

is called *theta join* of r_1 and r_2 on attributes A_1 and A_2.

$$\begin{aligned} r_1 \ltimes r_2 := \{\mu \in Tup(X_1X_2) \quad | \quad & (\mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2) \\ & \vee(\mu[X_1] \in r_1 \wedge \mu[X_2] \notin r_2 \wedge \mu[X_2] is\ null)\} \end{aligned}$$

is called *outer join* of r_1 and r_2.

A theta join is a join that relates data from two tables with a comparison operator that can be different from equality. If the comparison operator simply tests for equality, this join variation is also called *inner join.* A left or right outer join includes every record from one of the tables and only those records from the other table in which the related fields match each other exactly.

The different join operators have also a corresponding SQL query operation. As an example, an inner join is presented here. Let R, S be two relation schemata, $R = (A_1, A2)$, $S = (A_3, A4)$ and let r, s be relations over them. An inner join on attribute A_2 and A_3 is expressed as:

Code 3.5.

```
select distinct r.A1, r.A2, s.A3, s.A4
  from r, s
 where r.A2 = s.A3;
```

Definition 3.10. Let $r, s \in Rel(X)$.

- $r \cup s := \{\mu \in Tup(X) |\ \mu \in r \vee \mu \in s\}$ is called *union.*
- $r - s := \{\mu \in Tup(X) |\ \mu \in r \wedge \mu \notin s\}$ is called *difference.*
- $r \cap s := r - (r - s)$ is called *intersection.*

Note that union, difference, and intersection of relations are only defined if all relations have the same schema. If for instance $r \in Rel(AB)$ and $s \in Rel(BC)$, then $r \cup s$ is undefined, whereas $r \cup \rho_{C_{|A}}(s)$ and $\rho_{A_{|C}}(r) \cup s$ are defined if $dom(A) = dom(C)$ [Vos00].

Set operators have also corresponding operations in SQL. For example, let r_1, r_2 be relations over R. Then the union of r_1 and r_2 is expressed as:

Code 3.6.

```
select distinct * from r1
 union
select distinct * from r2;
```

As an example, the operators as defined above are applied to the database schema that represents the relational Web service operation repository, as shown in Figure 2.27. Different queries can be asked over this database schema.

Q1: Select all operations of a given Web service X:

$$\pi_{OperationName}(\sigma_{WebServiceName=X}(\\ WebService \bowtie_{WebService.WebServiceName=WShasOp.WebServiceName} WShasOp))$$

As this notation is hard to read when the relational algebra expressions become longer the SQL notation is preferred.

Q2: Select all Web services with their operations, their attributes, attribute types, data type domains, and data type ranges.

Code 3.7.

```
select WebServiceName, OperationName, AttributeName
       AttributeType, DataTypeDomain, DataTypeRange
  from WebService, WShasOP, Operation,
       OPhasAttr, DataType
 where WebService.WebServiceName =
          WShasOP.WebServiceName and
          WShasOP.OperationName  =
        Operation.OperationName  and
        Operation.OperationName  =
        OPhasAttr.OperationName  and
        OPhasAttr.DataTypeName   =
         DataType.DataTypeName;
```

The second query joins all relations of the Web service operation repository and select the attributes that are relevant for Web service search as determined in Section 2.4.2. This relation join across five relations has already been represented in Figure 2.28. In the following, this query will be named `WSOR`. Such a symbolic reference to a relational algebra expression or an SQL statement is called a *view*.

All elements that describe a relational database and its schema, e.g., relation names, attribute names, domains, primary and foreign key constraints provide information for database schema matching. When a match on the schema level has been identified, it is often necessary to generate transformation instructions that transform one database instance over the first schema into a database instance over the second schema. The operations of the relational algebra can be used to describe such data integration instructions. An overview of schema-matching techniques and their classification is given next.

3.2.2 Classification of Schema-Matching Approaches

Schema-matching approaches have been compared and analyzed in a survey by Rahm and Bernstein [RB01]. Their classification has been refined by [SE05] in a more recent approach that does not only apply to relational schemata but also to ontologies. This section introduces the classification by [RB01] and gives a brief overview of [SE05].

Classification of Rahm and Bernstein

A concise survey of schema-matching algorithms can be found in Rahm and Bernstein [RB01]. They distinguish match algorithms according to the following criteria as summarized in Figure 3.3:

Individual vs hybrid matching: Individual match algorithms employ exactly one match strategy based on a specific criterion of the input schemata. Hybrid matchers either consider several match criteria at a time or are able to combine the results of several individual matching algorithms.

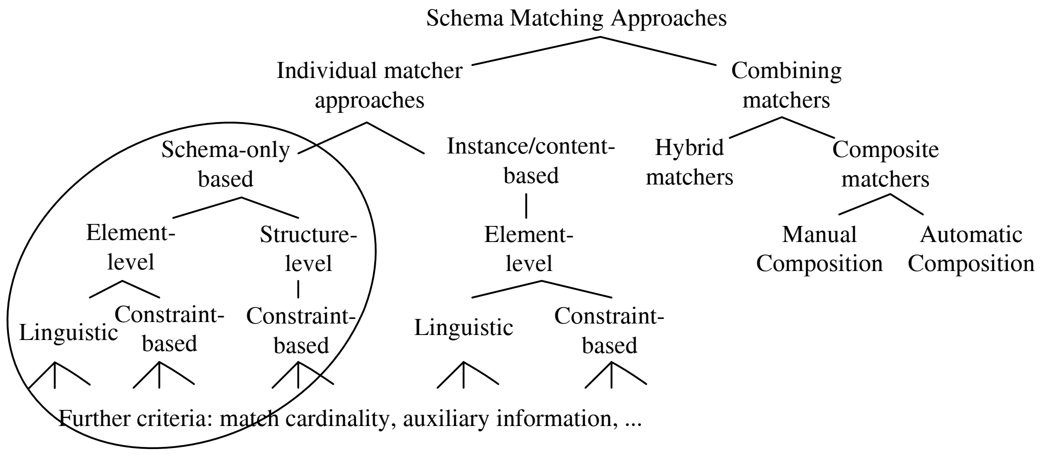


Figure 3.3: Classification of schema-matching algorithms according to [RB01].

Instance vs schema matching: The algorithm works either with the pure schema definition, e.g., the relational schema definition or with the instance data, e.g., a relation instance. The actual content of a relation can enhance the match that has been computed using schema information. Such a combination is an example of a hybrid matching approach. Instance matching can also be used on its own, e.g., if sufficient meta-information about a schema is not available.

Element vs structure matching: The match algorithm uses either single elements of an input schema, e.g., the names of each attribute within a relational schema or works on a more complex structure, e.g., the complete relational schema definition taking all attributes into account.

Language vs constraint matching: The algorithm uses names or textual descriptions within the schema, e.g., attribute names or comments, or is based on constraints, e.g., primary keys or foreign key relationships.

Matching cardinality: The mapping that the algorithm returns can relate one or more elements of the first schema to one or more elements of the second schema, thus returning matches of cardinality *1:1*, *1:n*, *n:1*, or *n:m*.

Auxiliary information: Match algorithms may make use of further information drawn from dictionaries, user input, ontologies, or global schema definitions. For instance, name-based linguistic matching algorithms rely on thesauri or sometimes even multi-language dictionaries.

Classification of Shvaiko and Euzenat

The individual, schema-based matchers, as indicated with a circle in Figure 3.3, have also been studied by Shvaiko and Euzenat [SE05]. They propose a different classification of this subbranch of algorithms that extends relational schema matching algorithms towards ontology matching.

First, the three aspects of schema matching, *input*, *matching process* and *output*, are explained because they form the framework for the classification:

Input: Matching algorithms work with different data representations or conceptual models, e.g., relational models, ER models, OO models, or XML. Further, they rely on individual *elements* of the models or on the input as a complete *structure*. The input can be interpreted as a string (*terminological* input), as a structure (*structural* input), or as a model (*semantic* input). Even if two algorithms work on the same input representation, they might still use different parts of it in a different way.

Matching Process: Every matching algorithm can either compute exact or approximate results. *Syntactic* algorithms take the input and process its structure without using any further information. *External* algorithms use knowledge as additional input that is provided by an external source such as user input or a dictionary. *Semantic* techniques rely on formal semantics to process the input and to verify their result. Terminological matchers can work *string based* or regard the input as a *linguistic* object. Structural matchers work either on the *internal structure* of the input (e.g., an attribute and its data type) or on the structure of *external relationships* (e.g., a relation and its foreign keys). Semantic matchers work with formal *models* or formal *ontologies*.

Output: The output can be of different cardinality. The computed mapping may be weighted and the relationship between mapped elements can be expressed in different formalism, e.g., as a mathematical function.

The different dimensions for input and matching process are used by [SE05] to derive two classifications as shown in Figure 3.4. The drawing can be read in both directions either focusing on how the input is interpreted (top-down) or on the kind of objects that are processed (bottom-up). The basic techniques shown in the middle are elementary building blocks for many different match algorithms and their implementation. They are encountered throughout the next sections and are therefore explained next.

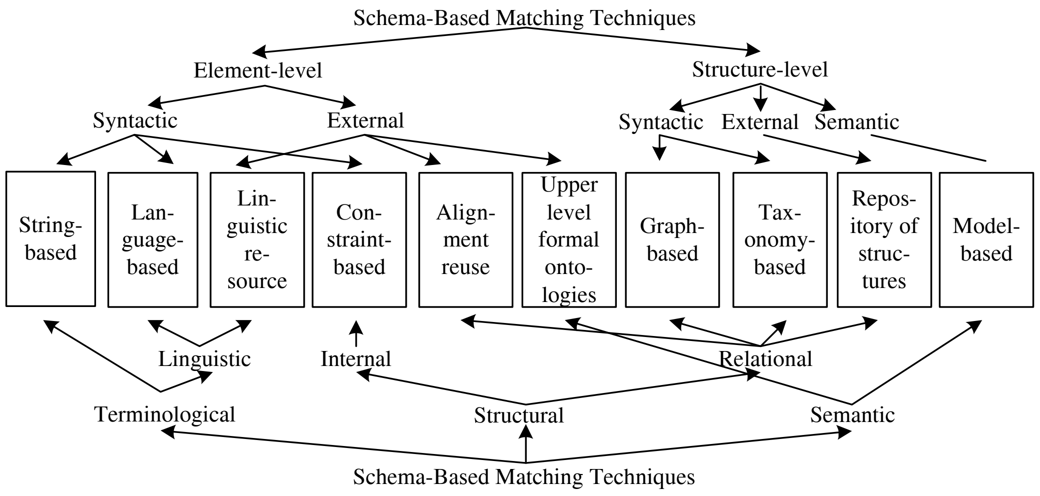


Figure 3.4: Classification of schema-matching algorithms according to [SE05].

String-based: String-based techniques compute the similarity of schema elements as the similarity of their names represented as strings. This is based on the intuition that two elements are likely to represent the same concept if their names consist of similar strings. Examples of string-based techniques are prefix, suffix, edit distance, and n-gram [DR02, GSY05].

Language-based: Language-based techniques regard the names of schema elements as words in a natural language to find morphological similarities. These techniques include tokenization, stemming and elimination of stop words [MBR01, GSY04]. They are often used as preparatory step before other techniques are applied.

Linguistic: Linguistic techniques rely on common knowledge or domain specific thesauri. These thesauri contain synonyms, hypernyms and homonyms. The hierarchies established are used to compute the similarity of two schema elements [Res95, MBR01].

Constraint-based: Constraint-based techniques make use of the different constraints that define schema elements such as the data type of an attribute, its cardinality, and its primary or foreign key relationships [RB01].

Alignment reuse: Alignment reuse subsumes all techniques that exploit previous schema matching results to deduce new matches from the given alignments. If two elements e and e' are matched and e' and e'' are also matched, then alignment reuse tries to compute a match between e and e'' as well [DR02].

Upper level formal ontologies: Upper level formal ontologies provide access to external common knowledge encoded in logic-based systems, e.g., the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [GGMO03]. Such ontologies can be used by matching algorithms as external information source.

Graph-based: Graph-based techniques transform a schema into a graph representation to apply graph matching techniques. As graph matching is a computationally expensive combinatorial problem [GJ79], most algorithms work with approximations such as two inner nodes are considered to be similar if the set of their immediate children is similar or if the set of their leave nodes is similar [DR02, MBR01].

Taxonomy-based: Taxonomy-based techniques are special graph-based techniques. A taxonomy contains information about "is-a" relationships between concepts that can be represented as a graph (see Section 4.2).

Repository of structures: In a repository of structures similarities between schema fragments are stored that have already been matched successfully. If a new schema fragment is to be matched the repository is queried to find elements that are sufficiently similar to the new fragment. This is done to reduce the search space before a thorough match computation with a more detailed algorithm is executed [RHDM04].

Model-based: Model-based match algorithms are deductive algorithms that work, e.g., with description logics reasoning techniques [GSY04, GSY05].

In the next section, different algorithms and system implementations are discussed that make use of combinations of these basic matching techniques.

3.2.3 Algorithms and System Implementations for Relational Schema Matching

When looking at the problem of match computation from a conceptually higher level, it reveals similarities to relational join processing in databases.

Both operations, match and natural join, compute pairs of corresponding elements between their two input sets. The type of input, the cardinality of the result and the semantics of the comparison expression are different. The match operator operates on meta-data, whereas the join operator processes the relation instance data. The match operator can relate several elements of one schema to several elements of the other schema, while the natural join operator combines one element of one table with one element of the other table. The meaning of match is less precise than the meaning of the natural join operator that works with a strict equality condition [BR00].

SQL-Based Match Implementation

Based on this observation, the match operator could be implemented using a dictionary that contains the transitive closure of similar terms and a similarity measure. If the schema elements and the dictionary are stored in relations, the match can be computed like a two-way join as explained in [RB01] and the result can be ordered by decreasing similarity:

Code 3.8.

```
//One entry per element of Schema1
Schema1 (Name)
//One entry per element of Schema2
Schema2 (Name)
//Dictionary with similarity score
D (Name1, Name2, Similarity)

  SELECT Schema1.Name,
         Schema2.Name,
         D.Similarity
    FROM Schema1, Schema2, D
   WHERE Schema1.Name = D.Name1
     AND D.Name2      = Schema2.Name
ORDER BY D.Similarity DESC
```

According to the classification of [RB01], this is a schema-based, element-level linguistic approach. The classifications of [SE05] regard this approach as schema-based, element-level, external linguistic approach that works on strings as linguistic objects. To reduce the number of matches, a user-defined similarity threshold can be used as additional selection condition to reduce the number of returned results. Then, matching elements are only included into the result set if their similarity exceeds the given threshold.

Similarity Flooding Match Implementation

A more sophisticated algorithm is the so-called similarity-flooding (SF) algorithm [Mel04] that has been developed as a generic graph based algorithm for matching different kinds of schema representations such as ER-diagrams, relational table definitions, or XML schema definitions. It is chosen here as an example of a more complex matching algorithm.

Let S_1 and S_2 be two relational table definitions. The similarity flooding algorithm consists of the following steps [Mel04] :

Code 3.9.

```
1. G1 = SQLDDL2Graph(S1); G2 = SQLDDL2Graph(S2);
2. initialMap = StringMatch(G1, G2);
3. product = SFJoin(G1, G2, initialMap);
4. result = SelectThreshold(product);
```

In the first step (line 1), the table definitions S_1 and S_2 are turned into graph structures G_1 and G_2 that are used as internal data model applying the function `SQLDDL2Graph`. In the second step (line 2), an initial mapping `initialMap` between the two graphs is computed. This is done using simple prefix and postfix string comparison algorithms for `StringMatch`. In the third step (line 3), an iterative fix point computation `SFJoin` is executed until the algorithm terminates. Based in the first mapping `initialMap`, the similarity between the two graphs G_1 and G_2 is computed. In the case of non-convergence, the algorithm terminates after a given number of iterations. Otherwise, it terminates when the difference between the results of two iteration steps reaches a defined ϵ. In the last step (line 4), all similarities below a certain threshold are discarded with `SelectThreshold`. This version is the core of the algorithm. Different variations are described in [Mel04].

The fix point iteration is explained in more detail, using an example from [MGMR02]. The internal data model for schema representation consists of directed labeled graphs which are defined as follows:

Definition 3.11. A directed labeled graph $G = (V, L, E)$ is a triple consisting of:

- a set V of vertices,
- a set L of labels and
- a set $E \subseteq V \times L \times V$ of labeled edges.

Consider the two directed labeled graphs $A = (V_A, L_A, E_A)$ and $B = (V_B, L_B, E_B)$ in Figure 3.5 on the left-hand side. An edge in these graphs is denoted as triple of $(source, label, target)$. A *pairwise connectivity graph* $PCG(A, B) = (V_{PCG(A,B)}, L_{PCG(A,B)}, E_{PCG(A,B)})$ is computed as preparation to fix point computation:

$$((x, y), l, (x', y')) \in E_{PCG(A,B)} \Leftrightarrow (x, l, x') \in E_A \wedge (y, l, y') \in E_B.$$

The PCG of the graphs A and B is shown in Figure 3.5 in the middle. Each node in this graph represents a *map pair*. Looking at the neighbors (a, b) and (a_1, b_1) the graph visualizes if a is similar to b then a_1 is also similar to b_1.

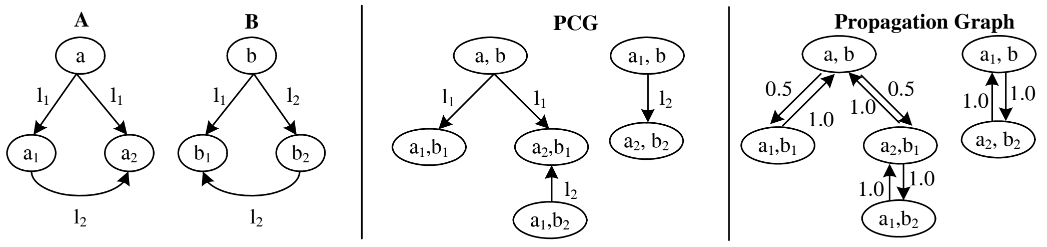


Figure 3.5: SF algorithm example according to [MGMR02].

This graph is turned into a *propagation graph* that shows how the similarity of a map pair contributes to the similarity of its neighbors. For every edge in the PCG an edge in the inverse direction is added. The weights at the edges are called *propagation coefficients* and can be computed in different ways as explained in [Mel04]. In the example given, each edge of type l_1 or l_2 makes a contribution of 1. If more than one edge of a type is present, the weight w is distributed equally, e.g., $w((a, b), (a_1, b_1)) = 0.5$ and $w((a, b), (a_2, b_1)) = 0.5$. This is shown in Figure 3.5 on the right-hand side.

Let $x \in V_A$, $y \in V_B$, then $\sigma(x, y) \geq 0$ denotes the similarity between the nodes x and y. The similarity is computed iteratively according to the following formula [Mel04]:

$$\begin{aligned}\sigma^{i+1}(x, y) \quad = \quad & \sigma^i(x, y) \\ & + \Sigma_{(a_u,l,x) \in E_A, (b_u,l,y) \in E_B} \sigma^i(a_u, b_u) w((a_u, b_u), (x, y)) \\ & + \Sigma_{(x,l,a_v) \in E_A, (y,l,b_v) \in E_B} \sigma^i(a_v, b_v) w((a_v, b_v), (x, y))\end{aligned}$$

In every step the computed measures are finally normalized, using a factor **Max**. The iteration is repeated until the Euclidean length of the vector difference $||\vec{\sigma}^n - \vec{\sigma}^{n-1}||$ converges.

The initial similarity σ^0 is computed using string comparison. For the example in Figure 3.5 this is omitted; instead, the similarity values are all initialized with 1.0. The first five iteration steps, the normalizing factor **Max** and the length of the vector difference $||\vec{\sigma}^n - \vec{\sigma}^{n-1}||$ are shown in Table 3.2. For example,

$$\sigma^1(a,b) \quad = \quad \sigma^0(a,b) + \sigma^0(a_1,b_1) + \sigma^0(a_2,b_1) = 3$$

As 3 is the maximum value in the step σ^1, 1/3 is used as normalizing factor. As can be seen in the example, the algorithm converges quickly. Intuitively, this approach computes the similarity between nodes in a graph taking their neighboring nodes into account.

Table 3.2: Example SF algorithm iteration steps.

(**AxB**)	$\sigma^\mathbf{0}$	$\sigma^\mathbf{1}$	$\sigma^\mathbf{2}$	$\sigma^\mathbf{3}$	$\sigma^\mathbf{4}$	$\sigma^\mathbf{5}$
$(\mathbf{a},\mathbf{b})$	1.00	1.00	1.00	1.00	1.00	**1.00**
(a,b_1)	1.00	0.33	0.14	0.06	0.03	0.01
(a,b_2)	1.00	0.33	0.14	0.06	0.03	0.01
(a_1,b)	1.00	0.67	0.57	0.50	0.44	0.38
(a_1,b_1)	1.00	0.50	0.43	0.41	0.40	0.39
$(\mathbf{a_1},\mathbf{b_2})$	1.00	0.67	0.64	0.66	0.67	**0.68**
(a_2,b)	1.00	0.33	0.14	0.06	0.03	0.01
$(\mathbf{a_2},\mathbf{b_1})$	1.00	0.83	0.86	0.87	0.89	**0.90**
(a_2,b_2)	1.00	0.67	0.57	0.50	0.44	0.38
Max		3.00	2.33	2.29	2.28	2.29
$||\tilde{\sigma}^\mathbf{n} - \tilde{\sigma}^\mathbf{n-1}||$		1.39	0.36	0.18	0.11	0.08

Generic Schema Matching Systems

As observed by [RB01] the representation of the schema, e.g., as ER-diagram, UML diagram, graph, or XML file does not have any influence on the classification of schema matching algorithms. This is due to the fact that most matching algorithms work on an internal schema representation suitable for their purpose. Therefore, most system implementation for automatic match computation strive to be representation independent. This is achieved by using an internal schema representation that is suitable for all implemented match algorithms within the system. Further, the implementation of the match algorithm usually consists of several different algorithms that are combined either by the system according to some heuristic or according to the users' choice. This makes the matching systems more flexible and extensible. A high-level process view for such a generic schema matching system implementation is given in Figure 3.6.

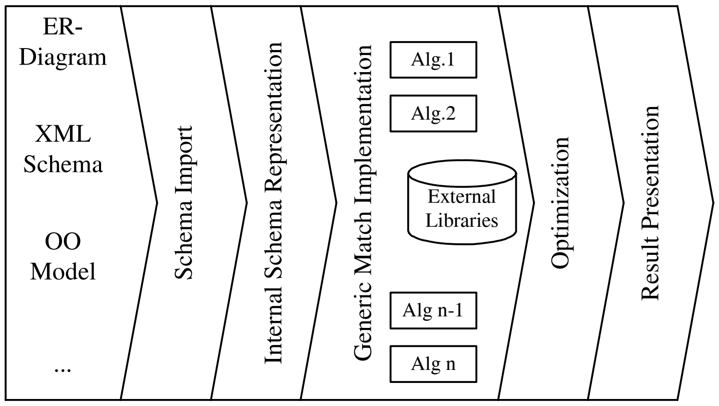


Figure 3.6: High-level process view for generic schema matching systems.

A generic matching system supports different schema representations that can be imported with adapted import routines. The result of the import step is a generic internal representation of the imported schema. The match operation itself is implemented as a collection of schema matching algorithms that are either applied in a sequence implemented in the system itself or can be chosen by the user. Such matchers make use of external information resources for matching and may also ask for user input. The result of the matching can further undergo optional optimization procedures that prune the best matches from a set of alternatives. The result is usually represented graphically to the user.

Numerous system implementations for schema matching have been reported and compared, e.g., in [RB01]. For example, the generic schema matching approach is pursued by the systems Rondo, Cupid, Clio, and COMA. This makes these systems especially valuable when studying the analogies between relational schema matching and Web service matching.

Rondo [Mel04] is a prototypical implementation of a tool for model management. A model is a representation of structured meta-information such as a relational table DLL or an XML schema document. In Rondo a number of operators for model manipulation are implemented, such as uniting two models or matching two models. The match operator in Rondo is implemented based on the SF algorithm.

Cupid [MBR01] combines name matching with a structural match approach. The schema definition is internally transformed into a tree. The similarity of elements is based on the similarity of their components at the lowest level of granularity, e.g., the similarity between two attributes in a relational table definition is computed using the similarity of the attribute names and the data type of the attribute.

Clio [HHH$^+$05] is a research prototype from IBM that claims to have reached industrial strength. The system computes schema mappings semi-automatically and, in addition, also produces queries that automate data transformation from one schema into the other. The mappings are either computed using an internal schema mapping algorithm or are provided by the user via a GUI. These mappings are then transformed into internal logical mappings expressed as declarative assertions which, in turn, are used to generated physical transformation function as SQL statement, XQuery, or XSLT script.

COMA [DR02] consists of a library of different matchers. They can be used in isolation or in combination and take domain specific synonyms and abbreviations into account that have to be defined once in advance. All algorithms are schema-based element-level matchers and work on an internal, rooted, directed acyclic graph representation of the schema. Recently, this system has also been extended to be used for ontology matching [ADMR05].

The systems introduced so far return $1:1$ matches. They do not make any suggestions on how the mapped elements are related, i.e., they do not return a mapping expression suggesting, e.g., that an instance in the first schema element can be transformed into an instance of the second schema element by a simple type cast function.

3.3 Schema Matching for Relational Web Services

The previous section has shown that relational database theory is able to give structure to persistent data and to describe the relationships between data instances. Further, based on this model, a wide variety of algorithmic approaches has evolved to compute the similarity between database schemata. The different approaches have also been implemented successfully.

Experiences with these implementations have shown that a generic schema matching process is beneficial. The match computation ought to have the following characteristics:

- It should be independent of the representation of the input data and use a uniform internal data model for computation.
- It should allow the combination of different matching algorithms to be adaptable and extendable.

This is a promising foundation for the matchmaking problem of Web services. In this section, it will be shown how relational database theory in general and relational schema matching theory in particular contributes to the problem of Web service operation matching.

Match computation for Web services using the relational operation representation establishes a match at the syntactic level in terms of operation names, attribute names, domains, and domain ranges. Following the match classification of [RB01] as presented in Section 3.2.2 these schema elements allow for schema-based element-level matching. In the following, the emphasis will be on element-level matching at the level of operation names and at the attribute level.

Motivated by the example in Section 3.1.1 a three-step approach is proposed here to execute match computation at a syntactic level: given a finite number of match candidate Web services and a Web service schema description that is searched for. First, the number of candidate services is reduced by matching the operation names. This relies on the assumption that the operation name adequately describes what the web service does when the operation is executed. This is presented in Section 3.3.1. Second, for the remaining Web services, input and output relations are to be matched as will be explained in Section 3.3.2. Both steps adapt the matching algorithms as presented in Section 3.2.3. These algorithms serve as examples to show that this approach is feasible, but may be replaced with other algorithms. Third, a manual revision of results must take place to operationalize the match as Section 3.3.3 will show. For simplicity, it is assumed that the desired Web service consists of one operation only. If more than one operation needs to be matched the three steps must be repeated per operation.

3.3.1 Matching of the Operation Name

In a preparatory step, the operation name must undergo simple language-based transformations such as stemming. For the following analysis, it is assumed that this preparation has already been executed. The schema specifications of all available implemented Web services are stored in a repository relation `WSOR = (WebServiceName, OperationName, AttributeName, AttributeType, DataTypeDomain, DataTypeRange)`, as introduced in Section

3.2.1. To compute a match at the level of operation names, the operation name of the desired operation *opname* is compared to every operation name in `WSOR.OperationName`.

The complexity of this comparison depends on the complexity of the search algorithm used. The search could be conducted, e.g., with simple string comparison algorithms or with the help of a dictionary `D = (Name1, Name2, Similarity)` containing the transitive closure over similar names and similarity measures between all terms listed in it. As the last approach also takes synonyms and homonyms into account, a simple example is given how such a comparison could be achieved as explained in [RB01] and presented in Section 3.2.2.

Code 3.10.

```
select distinct
       WSOR.WebServiceName,
       WSOR.OperationName,
       D.Similarity
 from  WSOR, D
where  D.Name1 = <OPNAME>
  and  D.Name2 = WSOR.OperationName
order by D.Similarity;
```

Proposition 3.1. Given a dictionary relation D of size d and a Web service operation repository $WSOR$ of size w. The overall costs of the operation name comparison given by the algorithmic approach above are in $O(d * w)$.

Proof. The complexity is determined by the size d of the dictionary and the size of the repository w. Executing the selection on `D` needs $O(d)$ comparisons and returns an intermediate result set of size $d' \leq d$. The projection on `WSOR` is executed in $O(w \log w)$ and returns an intermediate result set of size $w' \leq w$. If the join is executed as naive nested loop join which compares every entry in the first table with every entry in the second relation the costs are in $O(d' * w')$. □

As the intermediate result sets are usually much smaller than the original relations this worst case complexity will not occur in practice. Further, better join strategies than a naive nested loop join can be applied [SB02]. In a more restricted setting, it could even be assumed that equivalent operations are equally named. Examples such as ebXML [OAS01a] indicate that this assumption is realistic. In this case, the dictionary is not needed.

3.3.2 Attribute Matching

The first step has produced a set of operation names that are similar to the operation name searched for. This has already reduced the number of match candidates.

Next, the input and output attributes of the remaining match candidate operations are compared to the desired operation schema attributes. For this comparison, attribute names and attribute domains need to be taken into account. The complexity of this comparison depends on the number n of match candidate operations and the chosen comparison algorithm. For example, the similarity flooding algorithm (SF) of [Mel04] could be used as explained in Section 3.2.3.

Preparation

Among other input formats, the algorithm can take two relational schemata as input and compare (among other criteria) attribute names and domains. The relational Web service schema representation can easily be turned into two SQL table DDLs, one for the input relation r_0^{in} and one for the output relation r_0^{out}. The same must be done for the input relations $r_1^{in}, \dots, r_n^{in}$ and output relations $r_1^{out}, \dots, r_n^{out}$ of the n match candidate operations. The length of the code per operation depends on the number of attributes of the input relation r_i^{in} and the output relation r_i^{out} for $i \in \{0, \dots, n\}$, denoted by $|r_i^{in}| + |r_i^{out}|$. Let $|r^{in}| := \max_i(|r_i^{in}|)$ be the maximum number of attributes over all input relations and $|r^{out}| := \max_i(|r_i^{out}|)$ the maximum number of attributes over all output relations. Although the number of input or output attributes of an operation is theoretically unbounded, in practice the number will always be bounded by a constant. Therefore, the complexity of the schema transformation step is dominated by the number n of match candidate operations. Further, this transformation steps needs to be executed only once for every operation in the operation repository. When the transformation result is stored, the transformation costs may be neglected.

Lemma 3.1. The overall schema transformation preparation step for n match candidate operations is in $O(n)$.

Here, an example is given for the input and output relations of the Loan Pricing Service that has been defined in Table 3.1.

Code 3.11.

```
LoanPricing_GetConditions_In (
       net_value    FLOAT,
       duration     INT,
       score        INT
)
LoanPricing_GetConditions_Out (
       installment  INT,
       interest     FLOAT
)
```

Similarity Flooding

The SF algorithm then turns the two DDLs that are to be matched into two graphs, G_1 and G_2, with one node for every attribute and one node for every domain. Figure 3.7 shows the graph for the *LoanPricing_GetConditions_In* and the *LoanPricing_GetConditions_Out* table DDL. The representation shown here is based on [Mel04], which is an adaptation of the Open Information Model (OIM) specification of [BBC+99]. The same is done for the input and output relations of all match candidate operations.

Initially, the similarity between elements of the source relation graph G_1 and the target relation graph G_2 is computed using simple string comparison. Then similarity is computed using fix-point iteration. Similar nodes propagate their similarity to their neighbors. The SF algorithm returns a ranked list of 1 : 1 matches between elements of the two graphs and takes a predefined similarity threshold into consideration. Further filtering techniques could additionally be used on the results [Mel04].

According to [Mel04], the SF algorithm has a worst-case complexity of $O(|G_1| * |G_2|)$ per iteration, with $|G_1|$, $|G_2|$ being the number of edges in the two relation graphs respectively.

A relation r with $|r|$ attributes as used for the description of input and output in the relational Web service model produces a graph with $4|r| + 5$ nodes if each attribute has a different domain. The maximum number of edges in a graph is reached in a fully interconnected graph. For a graph representation of an SQL table DDL as used for Web service input and output, this will not be the case because the node for the table name is connected to the nodes for the columns and the nodes for the columns are

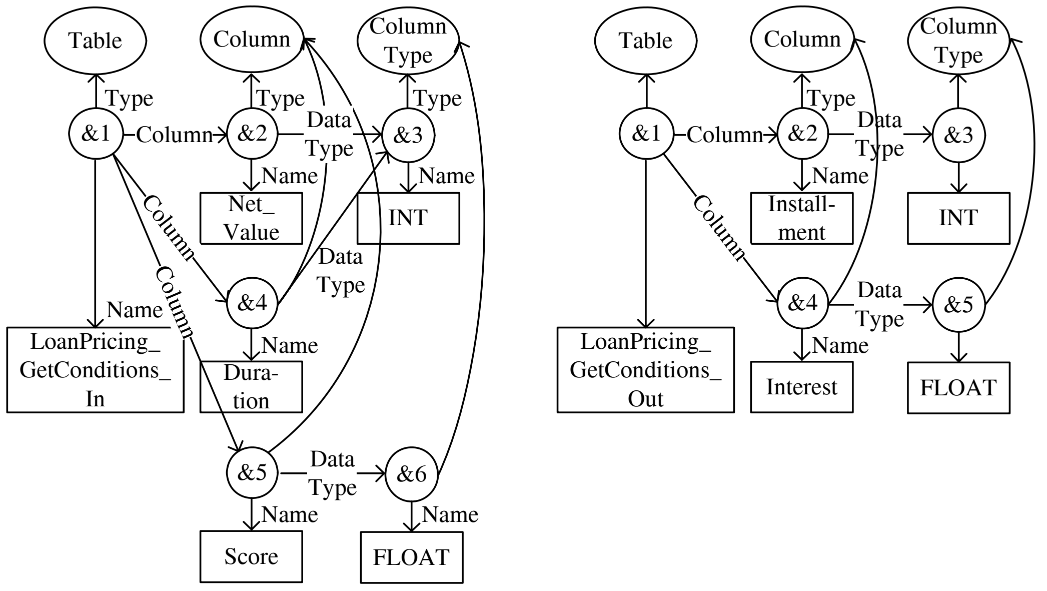


Figure 3.7: Input graphs for SF algorithm.

connected to their domains, but there is not any direct connection between the different columns, the different domains, or between the domains and the table name. In this model, it can be shown that a graph has at most $6|r| + 2$ edges. Therefore, the worst-case complexity for one iteration of the algorithm is $O(|r| * |s|)$ with $|r|$ and $|s|$ being the number of attributes of the two relations to be matched. According to [Mel04], the algorithm terminates after 5 to 30 iterations on the average. As above, the numbers $|r|$ and $|s|$ of attributes are theoretically unbounded, but in practical implementations the number of input and output attributes will always be bounded.

Proposition 3.2. Let r be the input or output relation schema searched for and $s_1, \ldots, s_n$ the implemented input or output relations of relational Web service operations in an operation repository. The overall complexity for one single match computation between r and s_i using the similarity flooding algorithm is $O(|r| * |s_i|)$ for the core of the SF algorithm. As the size $|r|$, $|s_i|$ is finite, the overall complexity of the search is in $O(n)$.

Proof. See [Mel04]. □

Therefore, reducing the number n of match candidates in the Web service operation repository, e.g., as shown by the dictionary approach in Section 3.3.1 enhances the performance of this second match computation.

Let $|r^{in}|$ and $|r^{out}|$ be defined as above. Let r_0^{in} and r_0^{out} be the designated input and output relation of the Web service schema searched for. Let $r_0 :=$

$\max(|r_0^{in}|, |r_0^{out}|)$ be the maximum number of attributes of these two relations and let $m := \max(|r^{in}|, |r^{out}|)$. Then the complexity of the operation name comparison and the input and output comparison can be summarized as follows:

Theorem 3.1. *The complexity of the two step search strategy using a dictionary of size d for operation name matching and a Web Service repository with n operations is in $O(d * n)$.*

Proof. This follows directly from Proposition 3.1 and Proposition 3.2. □

In general, the complexity depends on the number of match candidates and the complexity of the chosen matching algorithm for the two comparison steps.

3.3.3 Refinement

If the match computation has found more than one match, the match with the highest result is chosen or the list of matches can be revised manually. This is necessary because there might be several target operations with suitable input and output relations and nearly the same similarity measure.

In general, the users might want to add or remove a mapping between attributes due to their expert domain knowledge. Furthermore, if the domains of the two attributes mapped are not identical, the user needs to add transformation functions. This aspect of manual match refinement and its impact on the computed match are analyzed in detail in Section 3.4. Here, a short example is given:

A bank wants to buy a service that calculates a private customer scoring. All information available about private customers in the databases of the bank consists of name, gender, marital status, town, date of birth, yearly income, and number of children. As scoring a value between “not trustworthy” (0) and “very reliable” (1) is expected to be returned. The schema of the desired Web service is shown in Table 3.3. The name of the Web service and the operation are not repeated in this representation to enhance readability.

A match between operation names has revealed that there is a scoring service that offers private customer scoring functionality. This service expects the attributes first name, last name, gender, town, date of birth, monthly income, and number of family members as input and returns a value between 0 and 1. The schema of this implemented Web service is shown in Table 3.4. Again, the name of the Web service and the operation are not repeated.

A match has been computed between the attributes with similar names. Manually, the information has been added that number of family members

Table 3.3: Desired scoring Web service schema.

Web Service Name	**Operation Name**	**Attribute Name**	**Attribute Type**	**Data Type Domain**	**Data Type Range**
Loan_Pricing	Get_Scoring	Name	IN	STRING	not null
		Gender	IN	CHAR	not null $\wedge (=' F' \vee =' M')$
		Marital_status	IN	INT	not null $\wedge (= 0 \vee = 1)$
		Town	IN	STRING	not null
		Birth_date	IN	DATE	not null
		Income_year	IN	FLOAT	not null $\wedge \geq 12,000$
		Children	IN	INT	not null $\wedge \geq 0$
		Score	OUT	FLOAT	not null $\wedge \geq 0 \wedge \leq 1$

Table 3.4: Implemented scoring Web service schema.

Web Service Name	**Operation Name**	**Attribute Name**	**Attribute Type**	**Data Type Domain**	**Data Type Range**
Private_Customers	Get_Scoring	First_name	IN	STRING	not null
		Last_name	IN	STRING	not null
		Gender	IN	INT	not null $\wedge (= 1 \vee = 2)$
		Town	IN	STRING	not null
		Birth_date	IN	STRING	not null
		Income_month	IN	FLOAT	not null
		Family_size	IN	INT	not null
		Score	OUT	FLOAT	not null $\wedge \geq 0 \wedge \leq 1$

can be approximated from the information about marital status and number of children. To call the implemented Web service the bank must transform their own information into the format accepted by the Web service operation. To this end, transformation functions are needed. They are added manually in the revision step of the matching process. The result is shown in Table 3.5. The attribute *Name* is decomposed into *First_name* and *Last_name* using extraction functions. A mapping is needed that translates the *Gender* attribute. Further, a conversion function *date2string* is needed and the number of family members must be calculated. If a special transformation function is not needed, the identity function f_{id} is used to indicate this.

Table 3.5: Match and transformation.

Direction	**Source Attribute**	**Target Attribute**	**Instance Transformation**
IN	Name_s	First_name_t	$extract_{fn}(\text{Name}_s) = \text{First\_name}_t$
IN	Name_s	Last_name_t	$extract_{ln}(\text{Name}_s) = \text{Last\_name}_t$
IN	Gender_s	Gender_t	$f_{map}(\text{Gender}_s) = \text{Gender}_t$
IN	Town_s	Town_t	$f_{id}(\text{Town}_s) = \text{Town}_t$
IN	Birth_date_s	Birth_date_t	$date2string(\text{Birth\_date}_s) = \text{Birth\_date}_t$
IN	Income_year_s	Income_month_t	$f_{1/12}(\text{Income\_year}_s) = \text{Income\_month}_t$
IN	Children_s, Marital_status_s	Family_size_t	$f_{c+ms+1}(\text{Children}_s, \text{Marital\_status}_s) = \text{Family\_size}_t$
OUT	Score_s	Score_t	$f_{id}(\text{Score}_s) = \text{Score}_t$

3.4 Web Service Match Evaluation

Once a match between two relational Web service schemata has been computed, the next step is to define a transformation that turns the schema match into an executable transformation at the instance level of input and output messages. With such an executable transformation, it becomes possible to plug an implemented Web service into a pre-defined series of tasks without changing the underlying data flow.

This section shows how to derive such a uniform transformation expression at the implementation-independent level of extended relational algebra expressions. These extended relational match expressions can be used to establish a hierarchy of matches that indicates how close an implemented Web service meets the original specification.

A set of all transformation functions can be used to describe how to derive an executable match. At this point, the question arises if this always yields useful results. To answer this question, the notion of usefulness in this context must be defined.

Intuitively, a match between a desired Web service schema and the schema of an implemented Web service is *useful* if the resulting transformation instructions for input and output messages at the instance level yield the same results as would have been achieved with a new implementation of the desired service. This is given if all values that are possible inputs for the desired Web service schema are also accepted by the implemented service and if all outputs of the implemented Web service are also outputs within the desired Web service schema. This commutative relationship is shown in Figure 3.8.

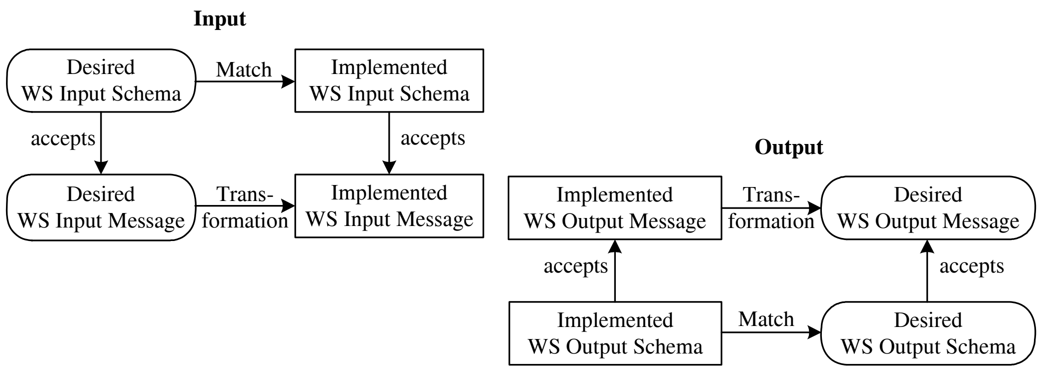


Figure 3.8: Information preserving match and transformation.

How schema mappings and transformation functions influence this goal is analyzed in detail in Section 3.4.1. In Section 3.4.2, it is shown how an implementation-independent transformation expression can be derived for useful matches. Section 3.4.3 uses this transformation expression to establish a match hierarchy.

3.4.1 Identifying Useful Matches

As already motivated in the example in Table 3.5 and Figure 3.8, an implemented Web service can be used in place of a specified service if mappings and transformation functions for all attributes of input and output can be defined. To analyze the usefulness of such a combination of mappings and transformations, the notion of information capacity as studied in [MIR93] is adopted.

Definition 3.12. The *input information capacity* of a Web service instance $I(WS)$ over a given Web service schema WS is defined as the set of all valid input tuples that this Web service instance accepts. The *output information capacity* of $I(WS)$ is defined analogously as the set of all valid output tuples that the Web service instance returns.

A Web service instance $I_1(WS_1)$ has more input information capacity (output information capacity) than a Web service instance $I_2(WS_2)$ if every valid input (output) tuple of $I_1(WS_1)$ can be transformed into a valid input (output) tuple of $I_2(WS_2)$. This is formalized in the following definitions in analogy to [MIR93].

Definition 3.13. An *information preserving mapping* between the valid input (output) tuples of two Web service instances $I_1(WS_1)$ and $I_2(WS_2)$ is a total injective function f that maps every valid input (output) tuple of $I_1(WS_1)$ to a valid input (output) tuple of $I_2(WS_2)$. If f is a total onto function that is not injective, f is called *information reducing mapping*. If f is information-preserving and a bijection, the mapping is called *equivalence preserving*. If f is total, but neither injective nor an onto function, f is called *information changing*.

Definition 3.14. A Web service instance $I_2(WS_2)$ is called a *substitution* for a Web service instance $I_1(WS_1)$ ($I_1(WS_1) \preceq I_2(WS_2)$) if there exists an information preserving mapping f_I from the set of valid input tuples of $I_1(WS_1)$ to the set of valid input tuples of $I_2(WS_2)$ and an information preserving mapping f_O from the set of valid output tuples of $I_2(WS_2)$ to the set of valid output tuples of $I_1(WS_1)$. If either f_I or f_O is information-reducing, $I_2(WS_2)$ is called a *replacement* for $I_1(WS_1)$ ($I_1(WS_1) \succ I_2(WS_2)$). If f_I and f_O are both equivalence-preserving, $I_1(WS_1)$ and $I_2(WS_2)$ are called *equivalent* ($I_1(WS_1) \equiv I_2(WS_2)$).

Lemma 3.2. Let the mapping between two Web services $I_1(WS_1)$ and $I_2(WS_2)$ consist of n transformation functions on the input side and m transformation functions on the output side.

- The two Web services are equivalent if all $n + m$ transformation functions are bijections.
- If Web service $I_2(WS_2)$ is not equivalent to $I_1(WS_1)$, but at least one transformation function is a total injective function and the remaining $n + m - 1$ transformation functions are either total and injective or bijective, then $I_2(WS_2)$ is a substitution for $I_1(WS_1)$.

- If Web service $I_2(WS_2)$ is neither equivalent nor a substitution for $I_1(WS_1)$, but at least one transformation function is a total onto function and the remaining $n+m-1$ transformation functions are either total and onto, total and injective, or bijective, then Web service $I_2(WS_2)$ is a replacement for $I_1(WS_1)$.

Looking at the customer score calculation example in Table 3.5, the output transformation consists of just one function. Only the attribute $Score$ needs to be mapped. As the domains on both sides are identical, the transformation function is the identity function f_{id}. Therefore, the mapping on the output side is equivalence preserving.

For the input mapping, the transformation function consists of all individual transformation functions at the attribute level as defined in Table 3.5. The two operations $extract_{fn}$ and $extract_{ln}$ that extract the first and the last name are total onto functions. The logically inverse function is the function $combine(\text{First\_name}_t, \text{Last\_name}_t) = \text{Name}_s$. The mapping f_{map} from the set $\{'F','M'\}$ to the set $\{1, 2\}$ is a bijection. The identity mapping for the attribute $town$ and the $date2string$ operation are bijections as well. The computation of the monthly income as the twelfth part of the yearly income is total and injective, but the inverse function is not total because the range for monthly income is not restricted, whereas the range of the yearly income must be at least 12,000. The computation of the family size is not injective but an onto function as, e.g., an unmarried parent with two children has a family of size three, which is also the size of a family consisting of a couple with one child. Therefore, the input mapping is information reducing and the implemented Web service is a replacement for the Web service searched for. From the perspective of the bank, the implemented Web service offers the desired functionality based on slightly less detailed information than the bank has stored.

Only if the computed match can be turned into an operational transformation of input and output tuples, it is possible to employ the Web service already implemented in the intended context specified by the Web service schema sought. This is examined in more detail in the following for the schema level, the match cardinality, and the instance level of input and output tuples.

Schema Level: The match algorithms as described in Section 3.3 compute a mapping between operation names, attribute names and domain names. If the operation name and the attribute names match, an additional domain match increases the degree of compliance and indicates that a domain transformation function owing to different data types is not needed on the instance level. If the domains do not match, a transformation function is needed. If the attribute names do not match, but the domains match, this information needs further investigation because it is very likely that two completely different attributes are of the same data type without having anything in common. Therefore, the detailedness or *granularity* of the match on the schema level influences the necessary transformations on the instance level.

Cardinality: A similar observation can be made for the aspect of match cardinality. A $1:1$ mapping of attributes with matching domains is described, using the identity function f_{id}. It is assumed that the two attributes have the same meaning and a domain transformation is not necessary. If an $n:1$ mapping is discovered, a transformation function is always needed to combine n attributes. Therefore, $n:1$ mappings always induce a transformation function. For $1:n$ mappings, two cases can be distinguished. Such a transformation is either a *replication* because the same attribute in its entirety is used several times or it is a *decomposition* because different parts of the attribute are used to compute new attributes. In the example above, the attribute $Name$ is used twice in a decomposition. Therefore, $1:n$ mappings also induce the use of a decomposing or replicating transformation function. Mappings of cardinality $n:m$ are excluded from this analysis because they do not contribute enough information or can be reduced to one of the other three types at a different level of detail as explained in [RB01].

Instance Level: On the instance level, the nature of all transformation functions finally determines the information capacity of the executable mapping instruction. This includes the domain constraints. If $f: A \rightarrow B$ is a total injective mapping, it is defined on every element of A, $f(A) \subseteq B$, and $a_1, a_2 \in A, a_1 = a_2$ implies $f(a_1) = f(a_2)$. Furthermore, restricting B to the image of A under f yields a bijection. If A gets restricted by a constraint c_A, this does not change the total and injective character of f. If B gets restricted by a constraint c_B, this might have the effect that f is not total any more, but gets undefined for some $a \in A$. This results in a partial mapping that is neither information-preserving not information-reducing any more.

Consider the following example: the mapping from attribute *income_year* to attribute *income_month* assumes an equal distribution of the yearly income across 12 months, $\textit{income\_month} = \textit{income\_year}/12$. This transformation function is total and injective. The inverse function exists and is total. Therefore, the transformation function is a bijection, $\textit{income\_year} = \textit{income\_month} * 12$ is the inverse function. Adding constraints on both sides, e.g., $18,000 \leq \textit{income\_year}$ and $1,200 \leq \textit{income\_month}$ still yields a useful match as $18,000/12 \leq \mathit{income\_year}/12$ implies that $1,200 \leq \mathit{income\_month}$ is also satisfied. Adding a different constraint on the target side, e.g., $2,000 \leq \mathit{income\_month}$ results in a conflict as a yearly income between $18,000$ and less than $24,000$ fulfills the source constraint, but violates the target constraint.

$$\begin{aligned}
& & & \mathit{income\_month} = \mathit{income\_year}/12 \\
& & \Leftrightarrow\ & \mathit{income\_year} = \mathit{income\_month} * 12 \\
\\
& & & 1,200 \leq \mathit{income\_month} \\
& & \Leftrightarrow\ & 14,400 \leq \mathit{income\_year} \\
18,000 \leq \mathit{income\_year} & & \Rightarrow\ & 14,400 \leq \mathit{income\_year} \text{ constraint satisfied} \\
\\
& & & 2,000 \leq \mathit{income\_month} \\
& & \Leftrightarrow\ & 24,0000 \leq \mathit{income\_year} \\
18,000 \leq \mathit{income\_year} & & \not\Rightarrow\ & 24,000 \leq \mathit{income\_year} \text{ constraint not satisfied}
\end{aligned}$$

Useful Matches:

Definition 3.15. *Useful matches* have matching operations names (renaming is allowed), matching attribute names (renaming is allowed), and domains between which transformation functions $f_1, \ldots, f_n$ exist:

- If $f_1, \ldots, f_n$ are total bijections, the match is called *equivalence match.*
- If $f_1, \ldots, f_n$ are total injective functions and there exists at least one f_i that is not onto, the match is called *substitution match.*
- If $f_1, \ldots, f_n$ are total onto functions and there exists at least one f_i that is not injective, the match is called *replacement match.*
- If one transformation function f_i is not total, but the image of the source domain (dom_s) under source constraints (c_s) has a non-empty overlap with the target domain (dom_t) observing the target constraints (c_t), the match is *restricted.*

If the overlap is empty, the match is *not useful.* If the operation name does not match, the attributes are not considered further. If the attribute names

Table 3.6: Hierarchy of useful matches.

Operation	**Attribute**	**Domain Transformation**	**Constraint**	**Match Type**
Match	Match	total, bijective	$f(c_s(dom_s)) \subseteq c_t(dom_t)$	equivalence
Match	Match	total, injective, not onto	$f(c_s(dom_s)) \subseteq c_t(dom_t)$	substitution
Match	Match	total, onto, not injective	$f(c_s(dom_s)) \subseteq c_t(dom_t)$	replacement
Match	Match	total but neither injective nor onto	$f(c_s(dom_s)) \nsubseteq c_t(dom_t) \wedge f(c_s(dom_s)) \cap c_t(dom_t) \neq \emptyset$	restrictedl
Match	Match	not total	$f(c_s(dom_s)) \cap c_t(dom_t) = \emptyset$	not useful
Match	Match	No Match		not useful
Match	No Match			not useful
No Match				not useful

do not match, they are not considered either because matching domains alone do not justify a match. These definitions are summarized in Table 3.6.

For the mapping, to be at least partially useful, the domain constraint on the source and on the target must be fulfilled for a non-empty set of elements of the source domain. It is desirable to know in advance, before implementing the mapping, if the source constraint also implicitly fulfills the target constraint. In a strict sense, this is only possible if the transformation function on the domains disregarding the constraints is a bijection. The yearly and monthly income is an example as shown above. There, restrictions on the monthly income can be translated into restrictions on the yearly income using the inverse function f^{-1}.

For $n:1$ matches, it is harder to decide if input that is accepted by the source service is also accepted by the target service. Consider the following example: The mapping $Family\_size = 1 + Marital\_status + Children$ is given. The mapping is total but not injective, e.g., the family size of an unmarried parent with two children is the same as of a couple with one child. Restricting the $Family\_size$ to four people can be translated into constraints for the source side allowing the following combinations of $(Marital\_status, Children)$:

$$\{(0,0),(0,1),(0,2),(0,3),(1,0),(1,1),(1,2)\}.$$

This can only be deduced by evaluation of the transformation function. The restriction on the target side can be expressed in terms of the source attributes as a rule like $(Marital\_status = 0 \Rightarrow Children \leq 3) \vee (Marital\_status = 1 \Rightarrow Children \leq 2)$. This shows that the restriction on the target domain imposes a restriction on two or more independent values of the source domain, indicating a dependency that does not exist in the source domains.

With this classification of useful matches and their characteristics the following section describes how to derive a self-contained operational match.

3.4.2 Relational Match Expressions

To make the match result operational, the individual match expressions must be turned into an integrated transformation instruction. In the following, the necessary transformation from source to target relation are described as relational algebra expressions using the operations *selection* σ, *renaming* ρ and *join* $\bowtie$ as defined in Section 3.2.1. The definition of *projection* π is extended, allowing *extended projections* based on the definition of extended projection in [SKS05] with additional extensions to multi-column input.

Definition 3.16. Let X be a relation schema and r a relation over X. Let $\{A_1, \ldots A_n\} \subseteq X$ and $B \notin X - \{A_1, \ldots A_n\}$. Note that the A_i need not be pairwise disjoint, but duplicates are permitted. Let $f : dom(A_1) \times \ldots \times dom(A_n) \rightarrow dom(B)$ be an information equivalent, an information preserving, or an information reducing function. Then the *extended projection operation* is defined as follows:

$$\begin{gathered}\rho_{B \leftarrow f(A_1,\ldots,A_n)}(\pi_{f(A_1,\ldots,A_n)}(r)) \\ := \{\mu \text{ over } B |\ \mu[B] = f(\nu[A_1, \ldots A_n]) \text{ for } \nu \in r\}\end{gathered}$$

For short, this operation is written as:

$$\pi_{f(A_1,\ldots,A_n)|_B}(r)$$

The extended projection translates to SQL as:

Code 3.12.

```
Select f(A1, ..., An) as B from r
```

In a generalized projection operation, any number of transformation functions is allowed. The ordinary projection operation as known from relational algebra can be expressed using the identity function f_{id} and keeping the attribute name. The ordinary renaming operation for an attribute as known from relational algebra can also be expressed using the identity function for projection and renaming the result.

The useful matches between two Web service schemata as listed in Table 3.6 can be described with these operations. As an example, the match for the customer scoring Web service as given in Table 3.5 is used. The matching expression is built up step by step examining the different cases as identified in Table 3.6.

The 1:1 matches for the input attributes `Gender`, `Town`, `Birth_date`, and `Income_year` can be expressed as extended projections on the source input relation (`In`) as follows:

Code 3.13.

```
select
       fmap(Gender)            as Gender,
       fid(Town)               as Town,
       date2string(Birth_date) as Birth_date,
       f1/12(Income_year)      as Income_month
  from In
```

Here, attribute names are equal or get renamed, domain names match and bijective transformation functions between attribute domains are applied. As there do not exist any domain constraints on the target side, the matching expression does not need any selection condition. A constraint on the source side needs not to be translated into a selection condition.

Assuming a constraint on the target side, e.g., `Income_month` $\geq 1,200$ results in a selection condition that is always fulfilled because it is less restrictive than the constraint `Income_year` $\geq 18,000$ that has already been defined on the source side. In this case, the matching expression is not changed. If the constraint on the target side is more restrictive, this needs to be expressed in a selection condition. Assuming that `Income_month` $\geq 2,000$ results in a matching expression of the following kind:

Code 3.14.

```
select
       f_map(Gender)           as Gender,
       f_id(Town)              as Town,
       date2string(Birth_date) as Birth_date,
       f1/12(Income_year)      as Income_month
  from In
 where Income_year >= 24,000
```

The $1:n$ match between `Name` and `First_name`, `Last_name` is also integrated using an extended projection:

Code 3.15.

```
select
       extract_fn(Name)        as First_name,
       extract_ln(Name)        as Last_name,
       f_map(Gender)           as Gender,
       f_id(Town)              as Town,
       date2string(Birth_date) as Birth_date,
       f1/12(Income_year)      as Income_month
  from In
 where Income_year >= 24,000
```

The $n:1$ match between `Children`, `Marital_status` and `Family_size` is also integrated:

Code 3.16.

```
select
      extract_fn(Name)                   as First_name,
      extract_ln(Name)                   as Last_name,
      f_map(Gender)                      as Gender,
      f_id(Town)                         as Town,
      date2string(Birth_date)            as Birth_date,
      f1/12(Income_year)                 as Income_month,
      fc+ms+1(Children, Marital_status) as Familiy_size
 from In
where Income_year >= 24,000
```

If a constraint is added on the target side, restricting $family\_size \leq 4$, this cannot be directly expressed as a constraint on the source side as the transformation is an onto function. The constraint can only be integrated after the transformation has been computed. This results in a matching expression of the following form:

Code 3.17.

```
select * from
(select
      extract_fn(Name)                   as First_name,
      extract_ln(Name)                   as Last_name,
      f_map(Gender)                      as Gender,
      f_id(Town)                         as Town,
      date2string(Birth_date)            as Birth_date,
      f1/12(Income_year)                 as Income_month,
      fc+ms+1(Children, Marital_status) as Familiy_size
 from In
where Income_year >= 24,000) as Intermediate
where Familiy_size <=4
```

For the output relation (`Out`), a second matching expression is needed. In the example given, this is: `select f_id(Score)as Score from Out`.

The two matching expressions for the input and output side can be regarded as two operations of a new, relational Web service with the following schema specification as shown in Table 3.7. The transformation Web service offers an input transformation operation and an output transformation operation between the two customer score operations.

Table 3.7: Transformation Web service schema.

Web Service Name	**Operation Name**	**Attribute Name**	**Attri-Bute Type**	**Data Type Domain**	**Data Type Range**
Transformation	In_transformation	Name	IN	STRING	not null
		Gender	IN	CHAR	not null $\wedge (=' F' \vee =' M')$
		Marital_status	IN	INT	not null $\wedge (= 0 \vee = 1)$
		Town	IN	STRING	not null
		Birth_date	IN	DATE	not null
		Income_year	IN	FLOAT	not null $\wedge \geq 24,000$
		Children	IN	INT	not null $\wedge \geq 0$
		First_Name	OUT	STRING	not null
		Last_Name	OUT	STRING	not null
		Gender	OUT	CHAR	not null $\wedge (=' F' \vee =' M')$
		Town	OUT	STRING	not null
		Birth_date	OUT	STRING	not null
		Income_month	OUT	FLOAT	not null $\wedge \geq 2,000$
		Family_size	OUT	INT	not null $\wedge \leq 4$
	Out_transformation	Score	IN	FLOAT	not null $\wedge \geq 0 \wedge \leq 1$
		Score	OUT	FLOAT	not null $\wedge \geq 0 \wedge \leq 1$

This means that a match between two Web service specifications ultimately produces a Web service specification as result. The internal behavior can be described entirely as an extended relational algebra expression. It describes in a declarative way the transformation that an implementation of a transformation Web service needs to conduct. In the next section, it will be examined if such an extended relational algebra expression can always be derived from a computed mapping.

3.4.3 Match Hierarchy

The characteristics of the relational matching expression are to be examined next. First of all, it needs to be observed that this description is independent of any programming language, but not unambiguous due to the redundancy inherent in relational algebra. For example, the selection expression $\sigma_{Family\_size \leq 4}$ in the example in Section 3.4.2 can be expressed as a join with a table containing just one column `Family_size` and four values $\{1, 2, 3, 4\}$. Therefore, the relational algebra expression does not give a direct hint as to how "good" the match is. To amend this, a normal form is introduced.

Definition 3.17. Let R be a relational Web service schema with n operation names $Op_1, \ldots, Op_n$. For every operation name Op_i two relation schemata R_i^{in} and R_i^{out} for input and output are defined. r_i^{in} and r_i^{out} denote relations over these schemata containing all valid input and output tuples. The set of matching expressions M is recursively defined as follows:

- The relations r_i^{in} and r_i^{out} are matching expressions.
- If E is a matching expression and c is a selection condition on attribute $A \in E$ of the form $A\Theta a$, $a \in dom(A)$ and $\Theta \in \{<, \leq, >, \geq, =, \neq\}$, or a combination of such conditions combined by $\neg, \vee, \wedge$, then $\sigma_c(E)$ is a matching expression.
- If E is a matching expression, $X \subseteq E$ set of attributes of E, $B \notin E$ not an attribute of E and $f : X \to B$ either a bijective, or total and injective, or total and onto transformation function, then $\pi_{f(X)|_B}(E)$ is a matching expression.
- All matching expressions are achieved by applying the above rules a finite number of times.

The following equivalences apply:

Theorem 3.2. *Given a relational Web service schema and $r \in \{r^{in}, r^{out}\}$ input or output relation over the relation schema $R \in \{R^{in}, R^{out}\}$ respectively.*

1. *Let* $X := \{A_1, \ldots A_n\} \subseteq R,\ B \notin R - X,\ C \notin R - X,$
   $f : dom(A_1) \times \cdots \times dom(A_n) \to dom(B),$
   $g : dom(B) \to dom(C)$. *Then*
   $\Rightarrow \exists h : dom(A_1) \times \cdots \times dom(A_n) \to dom(C) :$

$$\pi_{g(B)|_C}(\pi_{f(X)|_B}(r)) = \pi_{h(X)|_C}(r)$$

2. $\sigma_{c_1}(\sigma_{c_2}(r)) = \sigma_{c_1 \wedge c_2}(r) = \sigma_{c_2 \wedge c_1}(r) = \sigma_{c_2}(\sigma_{c_1}(r))$

3. *Let* $X_1 := \{A_{11}, \ldots A_{1n}\} \subseteq R,\ X_2 := \{A_{21}, \ldots A_{2m}\} \subseteq R,\ X_1 \cap X_2 = \emptyset,$
   $B_1 \notin R - X_1,\ B_2 \notin R - X_2,\ B_1 \neq B_2,$
   $f : dom(A_{11}) \times \cdots \times dom(A_{1n}) \to dom(B_1),$
   $g : dom(A_{21}) \times \cdots \times dom(A_{2m}) \to dom(B_2)$. *Then*

$$\begin{aligned} & \pi_{B_1, g(X_2)|_{B_1,B_2}}(\pi_{f(X_1), f_{id}(X_2)|_{B_1,X_2}}(r)) \\ = \ & \pi_{f(X_1), g(X_2)|_{B_1,B_2}}(r) \\ = \ & \pi_{g(X_2), f(X_1)|_{B_2,B_1}}(r) \\ = \ & \pi_{B_2, f(X_1)|_{B_2,B_1}}(\pi_{g(X_2), f_{id}(X_1)|_{B_2,X_1}}(r)) \end{aligned}$$

The first rule states that consecutive transformation and renaming on the same attribute can be merged into one single transformation step. This is easily shown defining $h := g(f)$. The second rule states that selection conditions can be swapped and follows directly. The third rule is obvious as the two attribute sets X_1 and X_2 are disjoint.

In the ordinary relational algebra, projection and selection operation can be exchanged if all attributes that are referenced in the selection condition also occur in the projection list. This rule is only valid in a restricted case for generalized projection operations:

Proposition 3.3. Let $X := \{A_1, \ldots A_n\} \subseteq R, B \notin R - X,$
$x := (a_1, \ldots a_n) \in dom(A_1) \times \cdots \times dom(A_n),$
$X \Theta x = (A_1 \Theta a_1 \wedge \ldots \wedge A_n \Theta a_n),$ selection condition,
$f : dom(A_1) \times \cdots \times dom(A_n) \to dom(B),$ monotone bijection. Then

$$\pi_{f(X)|_B}(\sigma_{X\Theta x}(r)) = \sigma_{B\tilde{\Theta} f(x)}(\pi_{f(X)|_B}(r))$$

In general, if f is not a bijection or not monotone, swapping selection and projection is not possible. The comparison operator Θ must be adapted to $\tilde{\Theta}$ using the inverse function f^{-1}.

Definition 3.18. A match between the input or output attributes of two Web service schemata is in *relational match normal form* (RMNF) if it has a matching expression of the following form

$$\sigma_d(\rho_g(\pi_{f_1(A_{11},\ldots,A_{1k}),\ldots,f_m(A_m,\ldots A_{ml})}(\sigma_c(r))))$$

where $0 \leq m \leq n$, r is the source relation over the relation schema $R = (A_1, \ldots, A_n)$. c is a selection condition of the form $A_i \Theta a$, $A_i \in R$, $a \in dom(A_i)$ and $\Theta \in \{<, \leq, >, \geq, =, \neq\}$ or a combination of such conditions combined by $\neg, \vee, \wedge$. ρ_g is a renaming operator that renames all $f(A_{ij})$ to B_i and d is a selection condition of the form $B_i \Theta b$, $b \in dom(B_i)$, $\Theta \in \{<, \leq, >, \geq, =, \neq\}$ or a combination of such conditions combined by $\neg, \vee, \wedge$.

The result of the example in Section 3.4.2 is a matching expression in RMNF.

Theorem 3.3. *Every match result between two Web service input or output schemata source S and target T of a schema matching algorithm given in the form $(\{S.A_1, \ldots, S.A_n\}, \{T.B_1, \ldots, T.B_m\}, exp)$ with $n = 1, m \geq 1$ or $n \geq 1, m = 1$ can be operationalized using an extended relational algebra expression in RMNF.*

Proof. 1. The transformation function is a monotone bijection and there is a domain constraint on the target side:

(a) The mapping is $1:1$: The mapping has the form
$(\{S.A\}, \{T.B\}, f(A) = B)$.
On B there is a domain constraint $B\Theta x$ defined. As f is a monotone bijection the target domain constraint can be expressed in terms of the source domain as $A\tilde{\Theta}f^{-1}(x)$. $\tilde{\Theta}$ denotes the adaptation of Θ under f^{-1}.

(b) The mapping is $1:n$: The mapping has the form
$(\{S.A\}, \{T.B_1, \ldots, T.B_m\}, f_1(A) = B_1, \ldots, f_n(A) = b_n)$.
On some B_i there are domain constraints $B_i \Theta_i x_i$ defined. As all f_i are monotone bijections the target domain constraints can be expressed in terms of the source domain as $A\tilde{\Theta}f^{-1}(\vec{x})$.

(c) The mapping is $n:1$: The mapping has the form
$(\{S.A_1, \ldots, A_n\}, \{T.B\}, f(A_1, \ldots, A_n) = B)$.
On B there is a domain constraint $B\Theta x$ defined. As f is a monotone bijection the target domain constraint can be expressed in terms of the source domains as $A_1\tilde{\Theta}f^{-1}(x)_1 \wedge \ldots \wedge A_1\tilde{\Theta}f^{-1}(x)_n$. $f^{-1}(x)_i$ denotes the *ith* component of the vector $f^{-1}(x)$.

All selection conditions from this case are written into the inner selection condition, followed by projection and renaming. The extended relational algebra expression has the form:

$$\rho_g(\pi_{f_1(A_{11},\ldots,A_{1k}),\ldots,f_m(A_{m1},\ldots A_{ml})}(\sigma_c(r))).$$

2. There is no domain constraint on the target side: An inner selection condition is not needed. Extended projection followed by renaming of the attributes is used. The extended relational algebra expression has the form:

$$\rho_g(\pi_{f_1(A_{11},\ldots,A_{1k}),\ldots,f_m(A_m,\ldots A_{ml})}(r)).$$

3. The transformation function is not a monotone bijection and there is a domain constraint on the target side: In this case, first the attributes are selected using extended projection followed by renaming. Then an outer selection condition is applied on the renamed attributes. The extended relational algebra expression has the form:

$$\sigma_d(\rho_g(\pi_{f_1(A_{11},\ldots,A_{1k}),\ldots,f_m(A_m,\ldots A_{ml})}(r))).$$

The above three cases are disjoint and cover all relevant variations. Therefore the three steps can be applied one after the other to yield the desired RMNF representation. Further steps are not necessary because a mapping between two individual relations is considered. □

Notice that the selection conditions that are constructed above are not necessarily satisfiable. Consider, for example, the following transformation that restricts the number of children in the inner selection conditions and the family size in the outer selection condition such that the two conditions together are not fulfilled at the same time.

Code 3.18.

```
Select * from
(select *
      Children                          as Children
      fc+ms+1(Children, Marital_status) as Family_size
  from In
 where Children <=2) as Intermediate
where Family_size >=5
```

Based on the above defined normal form, seven different match types can be distinguished. The following match hierarchy can be derived:

Perfect Match: All attributes on source and target side are matched 1:1. The match expression consists of generalized projection using the identity function only.

Equivalence Match: All attributes on source and target side are matched 1:1. The match expression consists of generalized projection using monotone bijective transformation functions and renaming operations only.

Substitution Match: All attributes on source and target side are matched. The match expression consists of generalized projection using total, injective transformation functions and renaming operations only.

Replacement Match: All attributes on source and target side are matched. The match expression consists of generalized projection using total onto transformation functions and renaming operations only.

Restricted Match: All attributes on source and target side are matched. One or more domains cannot be mapped completely. The match expression consists of generalized projection, renaming, and an *inner* selection condition which is satisfiable.

Partial Match: At least one attribute on source or target side remains unmatched or the inner selection conditions of the restricted match are not satisfiable.

Uncertain Match: All attributes on source and target side are matched. One or more domains cannot be mapped completely. The match expression consists of generalized projection, renaming, and an *outer* selection condition.

A perfect match and an equivalence match can be used straightforwardly without any changes to the logical model that was the starting point for the search. Transformation functions have to be implemented, but they are all bijections. A substitution and a replacement match require the programming of transformation functions to adapt the input parameters to the Web service or to adapt the result parameters. At least one of this transformation functions is injective or an onto function. A restricted match indicates that the available Web service possesses the functionality searched for and is compatible at the schema level but is not able to cover the complete domain of interest at the instance level.

3.5 Summary

In this chapter, a relational model that captures the syntactic characteristics of Web service operations has been defined. It consists of information about Web services, operations, data types, the grouping of operations into services, and the input and output attributes of operations. The relational model adopts a black-box approach. The operation is described by its name, its input, and output parameters. The internal implementation is hidden. The model is suitable to administrate schema information using relational algebra operators.

It has been shown that existing techniques from relational schema matching can be adapted to this relational model. As an example for such an adaptation, a dictionary-based approach in combination with similarity flooding has been examined. The match is computed on the level of operation names, input and output schema, and is finally refined manually adding transformation functions. For this, a notation for matching expressions has been defined. It has been shown that this approach can be realized adapting existing algorithms without increasing their algorithmic complexity. The complexity of the complete computation is dominated by the number of potential match candidates, which is decreased in the first step.

Based on the transformation functions, the match is operationalized using an extended relational algebra expression. Thus, a match between two Web service operation schemata becomes a Web service transformation operation itself, returning a Web service as result. The transformation functions are used to define a class of useful matches. Useful matches are expressed in a relational match normal form. This normal form is used to define a match hierarchy and to differentiate between match results.

The approach presented in this chapter has the advantage that it is completely based on the information available in WSDL files. The relational representation is suitable to use the tool-box of relational matching algorithms. As a drawback, the approach assumes a likeness in operation names, attribute names, and domain names. Therefore, the relational syntactic approach is enhanced with semantic support in the following chapter.

Chapter 4

Operation Matching with Semantic Annotations

For support of Web service detection in an intra-enterprise setting, it is desirable to use domain specific terminology that is typical of the business domain. The definition of this terminology is often one of the first steps of software development methodologies, e.g., object-oriented analysis and design [Oes01] or the entity-relationship approach to database design [Vos00]. The results of the initial domain analysis are often still visible in the final software product, e.g., as hierarchical dimensions in a data warehouse, as categories of the standard reporting, or as logical groups of input fields on a GUI. The business analyst and the programmer are both familiar with these terms. Therefore, these terms are suitable to describe a Web service operation.

When existing software is equipped with a Web service interface, it is natural to reuse the existing business terminology to describe the operations that a Web service interface offers. Intuitively, it is desirable to "connect" a Web service operation, its input and output parameters with terms of the application domain. This chapter presents an intuitive, yet efficient way to achieve this: *semantic annotations*. The idea of semantic annotations is based on the assumption that the technical Web service interface is generated from existing, implemented software and that the description is added after this automatic generation. The terminology for the semantic annotation exists independently of the Web service and stems from the business domain.

In the following, the idea and the use of semantic annotations are introduced. Section 4.1 presents a simplified usage scenario for semantic annotations in software development and maintenance. In Section 4.2, the underlying foundations of semantic Web technology and description logic are briefly summarized. They form the basis for semantic annotations for Web services. Section 4.3 introduces semantic annotations. An algorithm is presented that retrieves Web service operations, which match a given query, based on semantic annotations. It is shown that the algorithm is correct and efficient, yet not complete given an open-world assumption. Section 4.4 presents extensions and variations of the retrieval algorithm. Section 4.5 summarizes the results of this chapter while answering the remaining research questions from Section 1.2.

4.1 Motivation for Semantic Annotations

This section resumes the Internet loan application scenario and broadens the application scope so that it encompasses the functionality of a financial calculation kernel. The functionality of this software is a typical reuse candidate to be offered as a collection of Web service operations, which can be described by using a highly domain-specific terminology. Therefore, it is chosen as a running example to motivate the usage of semantic annotations in Web service operation search.

Section 4.1.1 gives a brief overview of the scenario. The complete scenario description can be found in the appendix. The methodology used to model the vocabulary for semantic annotations follows the methodology as introduced by [Hüs05].

In Section 4.1.2 typical search queries are categorized to introduce simple semantic queries. These queries use semantic annotations of Web service operations for search and retrieval.

Section 4.1.3 summarizes the research questions that arise when using semantic annotations for Web service operation detection. These questions will be answered in the remainder of this chapter.

4.1.1 Scenario Overview

The operations of a financial calculation kernel are used in different application contexts, e.g., the marketing department uses the algorithms to adjust the interest rates for loans and savings when the market conditions change. The calculated condition tables are then used in the Internet loan application as introduced in Section 2.1. A corporate customer consultant must be able to make an individual loan offer for important customers, yet be able to calculate the profit margin with the same methods as for a private customer's standard loan. In these two application scenarios, the price for loans is calculated ex-ante to make an offer to the customer. Later, when the loan is granted and the data is entered into the operational database, the same algorithms are used to calculate the profit margin which is then imported into the data warehouse and used for reporting purposes. This ex-post calculation ought to be executed with exactly the same algorithms as the ex-ante calculation to ensure consistency between ex-ante and ex-post results.

These examples show that the functionality of a financial calculation kernel is destined to be reused, not only for technical, but also for conceptual consistency. To this end, an easy and standardized access as granted by Web service interfaces is desirable. Neither is it important how the operations are grouped into individual services, nor is it important which department provides the service. Therefore, search functionality as offered by a UDDI, categorized by provider and service, is not sufficient, but it is also necessary to search on the level of operations and to refine this search based on input and output parameters. For example, an operation that calculates the present value of a loan can take the expected cashflow as input or take the loan account as input to calculate the cashflow on the fly.

From the application designer's point of view an integrated approach towards Web service administration and operation discovery is desirable. When the interface description of a Web service is published, it must be possible to annotate this description and attach domain specific information to the operation name as well as the input and output attributes. For example, a Web service operation that calculates a gross initial present value for a deferred payment loan is annotated as follows:

- The operation name is annotated with “Measure Calculation”.
- The input parameter is annotated with “Deferred Payment Loan”.
- The output parameter is annotated with “Gross Initial Present Value”.

Using annotations for Web service operation detection needs two prerequisites:

1. When searching for a Web service operation, the designer refers to the vocabulary that has been used for semantic annotations. For example, the designer might search for all Web service operations that output a gross initial present value. This presumes that the vocabulary for semantic annotations is controlled and designed in a domain-specific way. A model is needed to formalize this vocabulary and to use it in simple semantic queries.
2. Further, it is desirable to relate the terms of the vocabulary to each other. A search for all Web service operations that return a present

value is expected to include also all Web service operations that return a gross initial present value because it is a specialization of a present value. To achieve such an interpretation of the query the relationships between the terms of the vocabulary must be modeled explicitly and the search must be semantically enhanced.

To achieve the first aim, the relational model as used in Chapter 3 is extended to include domain-specific annotations as shown in Table 4.1. The programmer has called the operation "Compute_PV" and named the input parameter "Account" and the output parameter "PV". The semantic annotations sa_O and sa_A explain in domain-specific terms the meaning of these names. The table shows only the newly annotated part of the schema definition. The domain and constraint columns have been omitted for better readability. They do not contribute to the following explanations.

Table 4.1: Semantically annotated operation schema.

Web Service Name	**Operation Name**	sa_O	**Attribute Name**	sa_A	**Attribute Type**
Calculation_Service	Compute_PV	Measure Calculation	Account	Deferred Payment Loan	IN
Calculation_Service	Compute_PV	Measure Calculation	PV	Gross Initial Present Value	OUT

To achieve the second aim, taxonomies are employed. A taxonomy contains the domain-specific vocabulary and relates these terms to each other by three basic relationships: "broader-than", "narrower-than" and "same-as". Terms of a taxonomy are also called concepts. International standards for structured vocabularies are provided by, e.g., the British Standards Institution [Bri05a, Bri05b].

A concept is visualized as rectangle, a "narrower-than" relationship is depicted as an arrow from the specialized concept to the more general concept. The "broader-than" relationship is the inverse of the "narrower-than" relationship. The "same-as" relationship between two concepts states that the first is narrower or broader than the second and vice versa. Therefore, they are synonyms. In the following examples, the "narrower-than" relationship is preferred for visual notation.

The vocabulary to describe the financial calculation kernel consists of five domain-specific taxonomies for financial products, financial measures, customers, cashflows, and calculations. An overview is given in Figure 4.1. The complete model is documented in Appendix A.3.

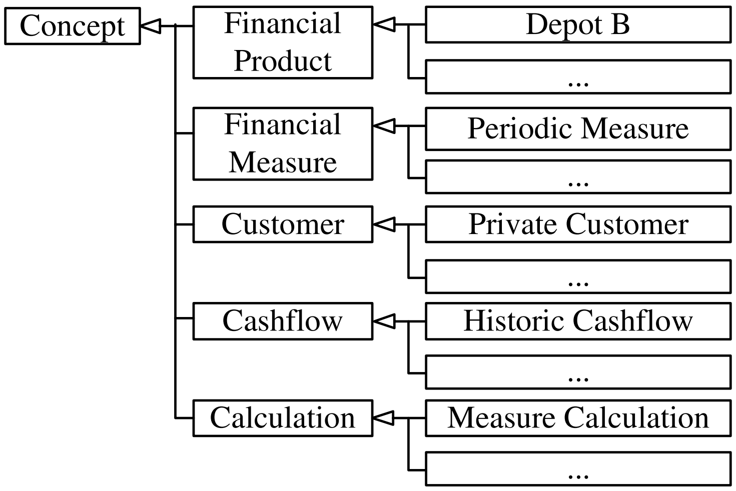


Figure 4.1: Domain-specific semantic submodels.

Financial measures are an example of a concept hierarchy of superconcepts and subconcepts as shown in Figure 4.2. The concept hierarchy as shown here is typical of financial reporting, e.g., for internal controlling.

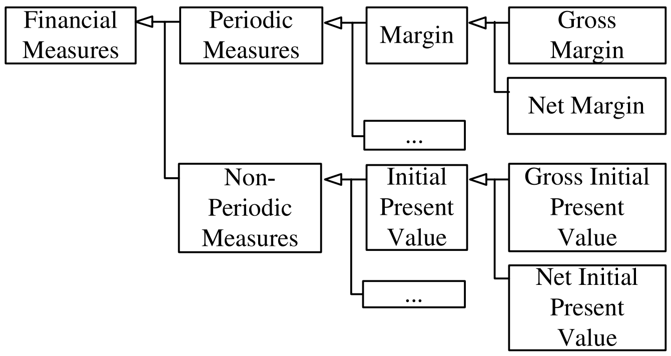


Figure 4.2: Measure concept hierarchy.

The financial products are also modeled as superconcepts and subconcepts as depicted in Figure 4.3. The online loan that has been the example of Section 2.1 is a subclass of a consumer loan which is a traditional loan product with fixed conditions and deferred payment.

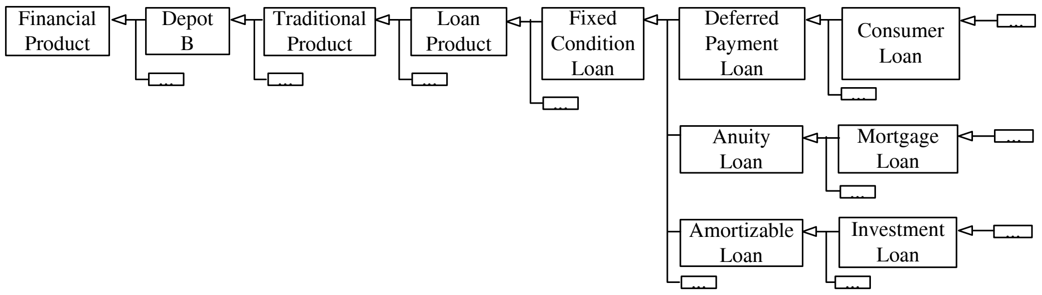


Figure 4.3: Financial products concept hierarchy.

Operations are also modeled as a concept hierarchy of superconcepts and subconcepts. The hierarchy shown in Figure 4.4 distinguishes three semantically different operation concepts: cashflow calculations, measure calculations, and rating calculations. To keep the example simple, only these three operations concepts are modeled in the beginning. The example will be extended in later sections.

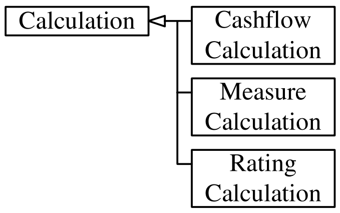


Figure 4.4: Calculation concept hierarchy.

An application that makes use of these annotations and supports a developer in adding semantic annotations as well as using them for semantic search could consist of the following components as shown in Figure 4.5:

- All interface descriptions with semantic annotations are available to the application (1).
- The taxonomy provides an acknowledged vocabulary for annotations and for searching (2).
- A GUI permits to explore the taxonomy graphically to choose concepts for annotations and for searching (3).
- The development environment permits to annotate operations, input, and output with concepts of the taxonomy (4).

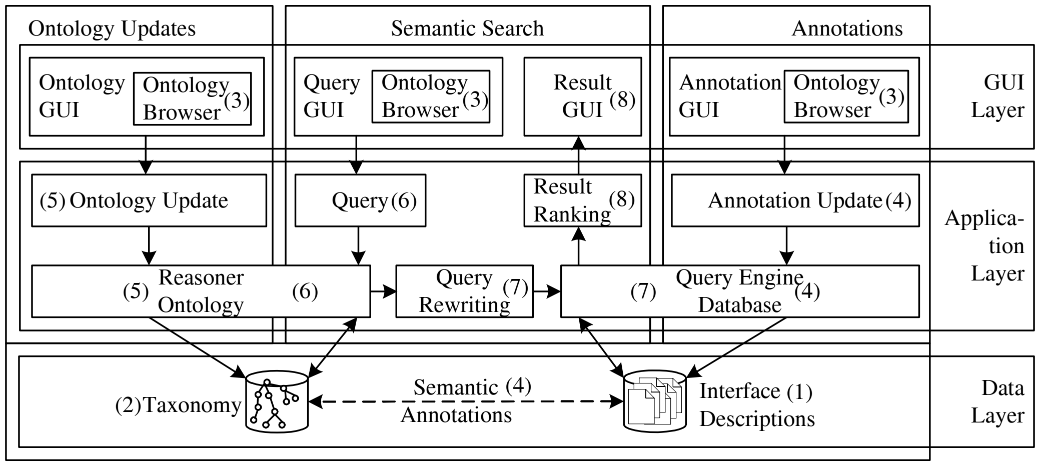


Figure 4.5: Ontology-based search.

- The development environment permits to extend and update the taxonomy (5).
- The development environment permits to search for available operations using concepts of the taxonomy (6).
- The search uses the taxonomy for inferences (7).
- The returned results are ranked according to their similarity with the query and displayed (8).

For this application, to work effectively, the developer must meet the following prerequisites in using the taxonomy:

- The developer uses a given application taxonomy to annotate the operation as well as its input and output parameters. Free text descriptions instead of annotations are not allowed, only in addition to annotations.
- The developer chooses the correct and most specific concept for annotation. The more general the chosen concept is the less it is useful for distinction, e.g., if all calculation operations are annotated as "calculations", the information provided is too general for useful search support.
- The developer extends the ontology if a concept is missing rather than using a concept too general for annotation. The taxonomy represents the vocabulary allowed for annotations, but this vocabulary is not exhaustive. It needs constant maintenance and extensions.

Techniques for visualization and graphical user support are offered, e.g., by Touchgraph interfaces[1]. Implementations that provide the necessary support for taxonomy creation and maintenance, such as OntoEdit[2] or Protege[3] exist. Reasoners for inferencing such as RACER [HM03] are often already integrated.

Based on this vision of semantic annotations for Web services and semantic search for operations the following section shows how this approach can be used for semantic match computation. The focus is on steps (6) and (7) of Figure 4.5.

4.1.2 Simple Semantic Queries

During software design and implementation reuse aspects need to be taken into account. Therefore, software designers as well as programmers must be able to find implemented functionality. The following questions represent typical searches for Web service operations based on operations, input, and output that are all expressible with the help of semantic annotations.

1. Select all operations that implement a specific operation concept.
2. Select all operations that have a specific concept as input resp. output.
3. Select all operations that implement a specific operation concept and have a specific concept as input resp. output.
4. Select all operations that have concept A or concept B as input resp. output.
5. Select all operations that implement a specific operation concept and have either concept A as input or concept B as output.
6. Select all operations that have concept A and concept B as input resp. output.
7. Select all operations that implement a specific operation concept and have concept A as input and concept B as output.

Further, more complex relationships between the concepts used for searching and the results need to be expressed. For example, the programmer searches for an operation that takes an annuity loan as input and returns the

[1]http://touchgraph.sourceforge.net/
[2]http://www.ontoknowledge.org/tools/ontoedit.shtml
[3]http://protege.stanford.edu/

gross present value of the account but this operation is not available. Only two operations are available: one that takes a fixed condition loan as input and returns its initial cashflow as output and a second operation that takes a cashflow as input and returns its present value. In this situation different searches must be expressible.

8. Select all operations that implement a specific operation concept and have a specific concept as input resp. output. Accept also operations with more specialized operation concepts, input and output concepts. This kind of search is called *subsumption search*.

9. Select all operations that implement a specific operation concept and have a specific concept as input resp. output. Accept also operations with a more generalized input concepts and more specialized operation and output concepts. This kind of search is called *plug-in search* because the results can be "plugged" into the place of operations that are exactly annotated with the search concepts.

10. Given an operation with a specific input concept, select all operations that generate this concept or more specialized concepts as output. This kind of search is called a *sequence search* because it allows to construct sequences of operations.

11. Given an operation with a specific output concept, select all operations that accept this concept or more generalized concepts as input. This is also a *sequence search*.

12. Select all operations that have a concept as input that has a given property. This is a *property-based search* that relies on the property of concepts when the exact concept is unknown to the user.

Examples for the first group of search queries are categorized next. The more complex queries are analyzed later in Section 4.3 and Section 4.4. Table 4.2 contains two Web service operation schemata that are used as running examples in the remainder of this chapter. The query examples shown make use of the SQL-notation for conjunctive queries instead of relational algebra expressions to keep the code readable.

Examples for the first two types of questions are shown in the following queries Q1 and Q2. Q1 returns a result bag containing the tuples (Calculation Service, Compute_MGL) and (Calculation Service, Compute_PVL) twice. Q2 returns the same results as Q2 without duplicates. The SQL implementation does not eliminate duplicates from the result set. Duplicate elimination is

Table 4.2: Web service operation example.

Web Service Name	**Operation Name**	sa_O	**Attribute Name**	sa_A	**Attribute Type**
Calculation Service	Compute_MGL	Measure Calculation	Account	Loan Product	IN
Calculation Service	Compute_MGL	Measure Calculation	MG	Margin	OUT
Calculation Service	Compute_PVL	Measure Calculation	Account	Loan Product	IN
Calculation Service	Compute_PVL	Measure Calculation	PV	Initial Present Value	OUT

achieved by applying the `distinct` operator to the final result bag. In the following examples, this duplicate elimination is omitted.

Code 4.1.

```
Q1: Select (WSNAME, OPNAME) from WSSCHEMA
     where SAO = ‘‘Measure Calculation’’;

Q2: Select (WSNAME, OPNAME) from WSSCHEMA
     where TYPE  = ‘‘In’’
       and SAA   = ‘‘Loan Product’’;
```

An example for the third type of query is given in the next code example. Q3 returns the set {(Calculation Service, Compute_MGL)}.

Code 4.2.

```
Q3: Select (WSNAME, OPNAME) from WSSCHEMA
     where SAO   = ‘‘Measure Calculation’’
       and TYPE  = ‘‘Out’’
       and SAA   = ‘‘Margin’’;
```

Queries like Q1, Q2, and Q3 are selection-projection queries and will be called *SP queries* in the following.

Examples for the fourth and fifth query are given in the following. Q4 and Q5 both return the result set {(Calculation Service, Compute_MGL), (Calculation Service, Compute_PVL)}.

Code 4.3.

```
Q4: Select (WSNAME, OPNAME) from WSSCHEMA
     where TYPE    = ``Out''
       and (   SAA = ``Margin''
            or SAA = ``Initial Present Value'');

Q5: Select (WSNAME, OPNAME) from WSSCHEMA
     where SAO          = ``Measure Calculation''
       and (    (TYPE   = ``In''
                 and SAA = ``Loan Product'')
             or (TYPE   = ``Out''
                 and SAA = ``Margin''));
```

Queries like Q4 and Q5 are selection-projection-union queries because the disjunction in the `where`-clause is equivalent to a union of SP queries. This type of query will be called *SPU query* in the following.

In the last two queries, a join is used to express the conjunction of search criteria on input and output attributes. Therefore, this type of query will be called *SPJ query*. The first query returns an empty set, the second query returns the result set {(Calculation Service, Compute_PVL)}.

Code 4.4.

```
Q6: Select * from
    (Select WSNAME, OPNAME from WSSCHEMA
      where TYPE    = ``Out''
        and     SAA = ``Margin'')
           MARGIN_VW,
    (Select WSNAME, OPNAME from WSSCHEMA
      where TYPE    = ``Out''
        and     SAA = ``Initial Present Value'') IPV_VW
    where MARGIN_VW.WSNAME = IPV_VW.WSNAME
      and MARGIN_VW.OPNAME = IPV_VW.OPNAME;

Q7: Select * from
    (Select WSNAME, OPNAME from WSSCHEMA
      where SAO   = ``Measure Calculation''
        and TYPE  = ``In''
        and SAA   = ``Loan Product'')            IN_VW,
    (Select WSNAME, OPNAME from WSSCHEMA
      where SAO   = ``Measure Calculation''
```

```
        and TYPE  = ''Out''
        and SAA   = ''Initial Present Value'') OUT_VW
     where IN_VW.WSNAME = OUT_VW.WSNAME
       and IN_VW.OPNAME = OUT_VW.OPNAME;
```

If the queries are evaluated, they return all operation names that match the query, not the complete Web service schema. If the complete Web service schema is wanted the information about Web service name and operation name as contained in the result set can be used to select the complete schema.

All queries that are discussed here are typical questions a software designer might ask to determine if a specific operation implementation exists. They are a special form of (unions of) conjunctive queries [AHV95], which use semantic annotations in the `where`-clause and are therefore called *simple semantic query* in the following. Conjunctive queries consist of selection, projection, and join operations, allowing positive conjunctions of selection conditions.

Definition 4.1. Given a set of concepts $\mathcal{C}$ and a set of Web service operations whose operation name, input, and output parameters are described by concepts of $\mathcal{C}$ as shown in Table 4.2. Then *simple semantic queries* (SSQ) are constructed applying the following construction rules:

1. A query which projects the Web service name and the operation name by using at most two concepts of $\mathcal{C}$ in the `where`-clause combined by a positive conjunction is a simple semantic query, called semantic selection-projection query (SP-query).

2. Unions of SP-queries are also simple semantic queries, called semantic selection-projection-union queries (SPU-queries).

3. Joins of SP-queries or SPU-queries are also simple semantic queries, called semantic selection-projection-join queries (SPJ(U)-queries).

4. All SSQs are constructed applying the above rules a finite number of times.

Simple semantic queries are SPJU-queries of relational algebra with the additional constraint that at least one selection condition always refers to a semantic annotation. Thus, complexity results for the evaluation of SPJU-queries are applicable to SSQs, which will be used in Sections 4.3 and 4.4. The examples of the previous section show that simple semantic queries are sufficient for meaningful searches and represent the main objects of investigation for the remainder of this chapter.

4.1.3 Research Questions

Section 4.1.1 has introduced the general setting for semantic annotations of Web services. Section 4.1.2 has shown by example that simple semantic queries can be expressed as SP, SPU, and SPJ queries with semantic annotations in the `where`-clause. This approach is used to answer the remaining questions of Section 1.2:

- How can additional semantics be attached to a service so that this additional documentation is generated at design time?
- How complex is semantic searching based on semantically enhanced service specifications?
- In which way do semantic extensions to existing standards enhance the quality of service description?
- What is the contribution to the software-development process?

The answers to these questions are given in three steps:

- A formalized model for simple knowledge representation systems such as taxonomies is presented, which is suitable for semantic annotations of Web service operations.
- The relational Web service repository is extended with semantic annotations and the meaning of a semantic annotation in the context of the formalized knowledge representation system is defined.
- An algorithm is presented that efficiently computes the answer to the semantic queries as motivated in the above examples. The correctness of the algorithm is shown and its complexity is analyzed.

Along this line, the impact of semantic annotations for service-oriented software development is examined. The search algorithm is based on description logic and embedded into the larger research scope of semantic Web services. Therefore, the following section first outlines the foundations of semantic annotations to prepare for the aspired solution.

4.2 Foundations of Semantic Annotations

The semantic Web and Web services represent complementary techniques to make information on the Web machine-processable and to make services on the Web machine-consumable. A combination of both techniques has resulted in *semantic Web services*. The semantic Web vision and the challenges of a combination of semantic Web and Web service techniques is briefly summarized in Section 4.2.1.

One of the most important formalisms to describe semantics is description logic (DL). DL is a first order calculus that allows to model knowledge bases and that has also been used to propose various models for semantic Web services, e.g., [MBH$^+$06, dBBD$^+$05]. The basics of DL that are used in the main part of this chapter are succinctly introduced in Section 4.2.2.

4.2.1 Introduction to Semantic Web Ideas

The semantic Web vision as described by Tim Berners-Lee [BLHL01] is an extension of the known WWW that enables machines to derive the meaning of the displayed information and to process it automatically.

For this vision to become true, Web pages must contain semantic meta-information about the meaning of the displayed information. This context must be represented in a structured way to be machine-processable and it must be possible to express knowledge in terms of concepts and rules that hold between concepts. For example, if a Web page contains a person's address, this information must be marked unambiguously as either the place where the person lives or works or as the text of a speech that the person delivers,. This "terminological control" [Gau05] is established through so-called ontologies. The resulting "semiotic triangle" [OR23] is shown in Figure 4.6.

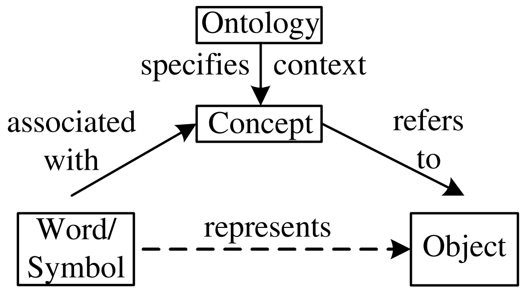


Figure 4.6: Semiotic triangle with ontological context control according to [OR23].

Originally, “ontology” is a technical term of Aristotelean philosophy and denotes the science (Greek “logos”) of being (Greek “onta”). Its aim was to create a classification of all existing things to be able to talk about them and to identify their order and their nature. In philosophy the ontology tries to capture the essence of the material world [Sow00] in one classification schema.

In computer science the term ontology is often defined as “a formal explicit specification of a shared conceptualization” [Gru95]. A conceptualization is a model of concepts. Concepts consist of objects that usually have several attributes. Concepts are related to each other either by horizontal “synonym-of”- or vertical “specialization-of”-relationships . An ontology models concepts, attributes and relationships explicitly and formally. This makes ontologies suitable for automatic processing. Usually, an ontology is restricted to a specific domain. Within this domain, the ontology represents a shared model of concepts which means that there must be a mutual consensus among the majority of users within the domain about the relevant domain specific concepts. Therefore, many ontologies exist in computer science, e.g., [Mil95]. They are language-dependent and none of them raises the claim to be complete.

If terminological control does not exist, the communication process may result in a misunderstanding because a symbol is ambiguous. Figure 4.7 shows this situation with an informal and a formal interpretation of the term “address”.

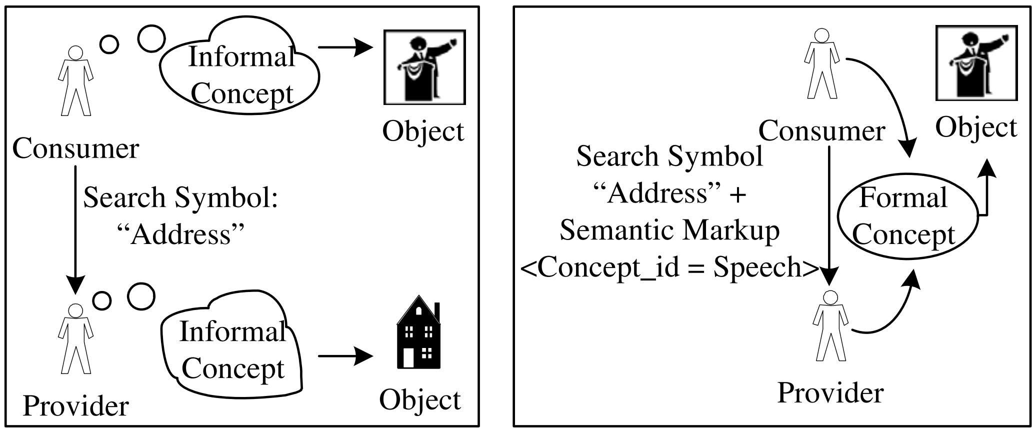


Figure 4.7: Communication with informal and formal interpretation of concepts.

If both communication partners agree upon a common formalization of context this can be used as additional explicit and formal semantic information in the communication process to avoid misunderstandings. The semantic markup must be added by the information provider and used by the consumer.

Different types of ontologies are distinguished according to [Gua98] as depicted in Figure 4.8. So-called *top-level ontologies* are used to describe general concepts that do not belong to a specific domain. Such ontologies represent a common consensus on a very high level of abstraction and do not go into detail. They specify general concepts like space, time, or object. Top-level ontologies are refined by *domain ontologies* and *task ontologies*. Domain ontologies refine concepts of the top-level ontologies related to a domain, e.g., multi-media. Task ontologies describe concepts that are related to specific activities within this domain, e.g., selling. An example for a domain ontology is the multimedia reference ontology as defined in [Hüs05]. A combined specialization of task and domain ontologies is used to define an *application ontology*.

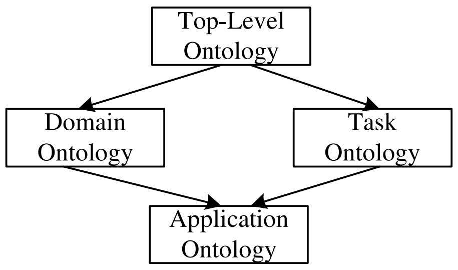


Figure 4.8: Ontology types according to [Gua98].

The combination of Semantic Web and Web service techniques has let to *semantic Web services*. As Figure 4.9 shows, there are at least two possible ways to combine the two technologies: on the one hand, Web services can be used to give access to semantic Web technology (arrows marked with A in Figure 4.9), e.g., as a Web service interface to a knowledge representation base; on the other hand, semantic Web technology can be used to describe the functionality of Web services, to find them, and to compose them (arrows marked with B in Figure 4.9). The latter is the more common understanding of the term semantic Web service [MCSZ01, QW04]. This notion will also be used in the remainder of this work and described in more detail in the following sections.

The syntactic representation of a Web service, the WSDL file, is technically machine-processable, but the meaning of the different syntactic elements is not machine-processable. To achieve automatic processing of the semantics of a description element, a uniform and unambiguous interpretation of

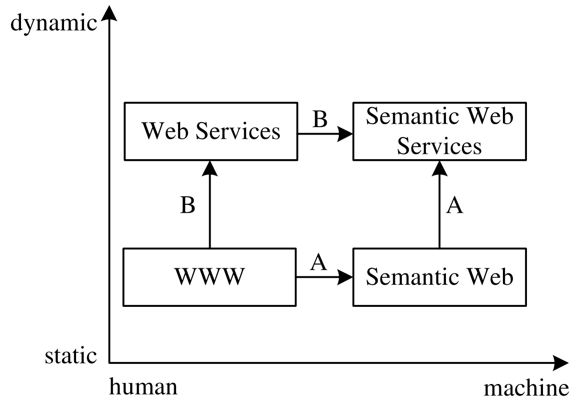


Figure 4.9: Technology evolution on the Web.

the elements must be provided for. Web applications based on machine processable semantics rely on simplifying assumptions [Usc03]:

1. The representation language that the application understands is widely used on the Web. If different representation languages are used on the Web, an application must be able to process all of them.

2. The assumptions about the concepts of a representation language must be compatible, e.g., the concept of time must either be represented as a time interval or as time points.

3. The concepts modeled in an ontology must be publicly declared. This is to prevent two independent ontology designers from inventing different ontologies for the same domain.

On the one hand, there are a number of publicly available categorization schemes and high level ontologies, such as the lexical reference system for the English language WordNet[4], the NISO-Standard Dublin Core[5] for generalized cross-domain resource descriptions, or the "Suggested Upper Merged Ontology" (SUMO[6]), which is intended as a categorization scheme for lower level ontologies. These ontologies are general enough for a broad consent. On the other hand, there are many, very domain specific ontologies, mainly concerned with a discipline of natural science, that already posseses a very standardized vocabulary, e.g., the Gene Ontology[7], the Protein Ontology[8],

[4]http://wordnet.princeton.edu/
[5]http://dublincore.org/
[6]http://www.ontologyportal.org/
[7]http://www.geneontology.org/
[8]http://proteinontology.info/

or the Plant Ontology[9]. These examples show that there are at least two ways to achieve consensus about the concepts of an ontology: Either it is so general that it fits to many domains of discourse, or it is the formalization of an already established vocabulary that is only disputable in details by experts. In both cases, the assumptions on machine processable semantics as stated above, are fulfilled.

The application scenario of this thesis, company internal service-oriented software engineering, is situated in between a generalized high-level and a domain specific low-level ontology because it is assumed that the ontology evolves with the company needs. The developers do not only use a fixed set of terms but also develop the ontology. In the internal setting, it is possible to choose one representation language for semantic Web service description and to assume compatibility of conceptualizations and public consensus about their meaning. Public, in this case, means "among the employees of the enterprise". In a more general setting encompassing several enterprises of the same industry or even different industries, this consensus is harder to achieve. For the financial industry, different international standards exist, e.g., XBRL[10] as mentioned before.

Using machine-processable semantics always requires first, to identify the meaning of a Web service and then, to add an ontological reference. This is an additional step after the creation of the Web service. In general, there are different approaches how to derive the meaning of machine-processable information:

Creating sophisticated search engines: The analysis of Web pages is based on information retrieval techniques and artificial intelligence to deduce meaning from word frequencies and additional information such as the links between pages which Google uses in the PageRank algorithm [BP98]. The Web service search engine Woogle [DHM+04] has also used information retrieval techniques to detect Web service operations. This approach is only applicable in an environment with a considerable number of services that can be clustered. In an enterprise-internal setting redundant service development is to be avoided. Therefore, an information retrieval approach is not applicable.

Involving the user in the evaluation of results: User feedback is encouraged in the Search Wikia[11] project. This is the attempt to develop an open source search engine with user editable results to improve the

[9]http://www.plantontology.org/

[10]http://www.xbrl.org/

[11]http://search.wikia.com/

quality of search results. Yahoo!Mindset[12] sorts results based on user feedback. Each result has a score between -2 and 2 indicating whether content of the page is commercial (-2) or informational (2). Semantic Wikipedia is combination of semantic Web and Web 2.0 [VH07] techniques. Instead of employing semantic annotations by experts the content of Web pages is semantically enhanced by a community of users replacing expert judgment with collective judgment. These techniques could be employed to improve the quality of a Web service search engine as well, but they are not geared towards Web service discovery in particular.

Engaging the provider: Searching the Web becomes less difficult if providers annotate their content in a machine processable standardized way to encode the meaning of a text fragment explicitly into the document. This can be done by using standardized XML tags. As already mentioned, standards such as XBRL[13] for financial reporting define an XML vocabulary for a specific application domain. In such an XML page, the tag `<revenue>` has a defined meaning; and a search engine that is programmed to "understand" the specialized XML vocabulary can exploit this information. As a standardized vocabulary is a basic result of software development activities, this approach is also applicable for Web services as software artifacts.

The provider-centric solution is the approach of the semantic Web and Web services. In an internal enterprise setting, a combination of user and provider involvement is also a possible solution scenario. As the vocabulary for annotation evolves with the software needs, constant maintenance by the community of users is necessary. The vocabulary of the ontology can be formalized using description logic which will be introduced in the following section.

4.2.2 Introduction to Description Logic

Description Logic (DL) offers a family of knowledge representation formalisms based on first-order predicate calculus to build *knowledge bases* (KB). In DL, an application domain or so-called *world of discourse* is modeled by, first, defining the different *concepts* and *roles* within the domain and, second, by specifying *properties of individuals* within the domain with the help of concepts and roles. Concepts can be thought of as unary predicates,

[12]http://mindset.research.yahoo.com/
[13]www.xbrl.de

roles as binary predicates, which represent relationships between concepts, and individuals as constants.

A KB consists of two parts: the *TBox*, which defines concepts and roles, provides the terminology to model the intensional knowledge about the application domain. The *ABox* consists of assertions about the individuals of the application domain and contains extensional knowledge. KBs vary in the constructs that they allow to specify new concepts and roles from already existing ones. The expressive power of a DL is determined by these constructors [BN03].

Definition 4.2. Given a set $\mathcal{C}$ of concepts. Terminological axioms between concepts $A, B \in \mathcal{C}$ are either *inclusions* $A \sqsubseteq B$ or *equalities* $A \equiv B$. A concept is called *atomic* if it occurs only on the left-hand side of an equality. An equality with an atomic concept on the left-hand side is called a *definition*. A set of axioms is called a *terminology* or *TBox*, if each atomic concept is defined only once.

This definition can be extended to include roles and role constructors. As an example, DL-Lite [CGL$^+$04] is presented, which is a simple yet expressive DL language. In DL-Lite the following concept constructors are allowed:

$$\begin{aligned} B &:= A|\exists R|\exists R^- \\ C &:= B|\neg B|C_1 \sqcap C_2 \end{aligned}$$

As a notational convention, A denotes an *atomic concepts*, R an *atomic role*, and R^- the *inverse* of an atomic role. Atomic concepts and roles are not defined with the help of other concepts and roles. B denotes *basic concepts*. C or also D are used for non-basic and non-atomic concepts.

The TBox allows inclusion and functionality assertions: $B \sqsubseteq C$ and $(func\ R), (func\ R^-)$. An inclusion assertion states the a basic concept is subsumed by a more general concepts. A functionality assertion of a role states that the relationship is functional.

The ABox allows membership assertions of the form: $B(a), R(a, b)$ stating that a constant a belongs to a concept B or that the pair (a, b) belongs to role R.

The interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of an infinite domain Δ and an interpretation function $\cdot^{\mathcal{I}}$ with the following semantics:

$$\begin{array}{rcl}
A^{\mathcal{I}} & \subseteq & \Delta \\
(\neg B)^{\mathcal{I}} & = & \Delta \setminus B^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
R^{\mathcal{I}} & \subseteq & \Delta \times \Delta \\
(\exists R)^{\mathcal{I}} & = & \left\{c | \exists c' : (c, c') \in R^{\mathcal{I}}\right\} \\
(\exists R^{-})^{\mathcal{I}} & = & \left\{c | \exists c' : (c', c) \in R^{\mathcal{I}}\right\}
\end{array}$$

In addition, $\uparrow$ denotes the universal concept and $\downarrow$ represents the empty concept.

$$\begin{array}{rcl}
\uparrow^{\mathcal{I}} & = & \Delta \\
\downarrow^{\mathcal{I}} & = & \emptyset
\end{array}$$

The main reasoning tasks for TBoxes in general are checking *satisfiability* and *subsumption* of concepts [BN03].

Definition 4.3. Given a TBox T.

- A concept C is *satisfiable* with respect to a TBox T if there exists a model $\mathcal{I}$ of T so that $C^{\mathcal{I}} \neq \emptyset$. $\mathcal{I}$ is called a *model* of C.
- A concept C is *subsumed* by a concept D with respect to a TBox T if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of T.

The reasoning tasks of satisfiability checking and subsumption checking can be mapped onto each other [BN03]:

Lemma 4.1. Given a TBox T.

- A concept C is unsatisfiable $\Leftrightarrow C \sqsubseteq \downarrow$.
- C is subsumed by $D \Leftrightarrow C \sqcap \neg D$ is unsatisfiable.

Inclusion axioms express that a concept C is contained in a concept A, usually because there is a specializing characteristic that all members of C possess, but that is not common in all members of A. Therefore, $C \sqsubseteq A$ is a short-hand notation for $C := \overline{C} \sqcap A$, where $\overline{C}$ represents the concept that is characteristic of all members of C. For example, the concept *woman* is contained in the concept *person*. The specializing characteristic concept is *female* [BN03]. Therefore:

$$woman \sqsubseteq person \Leftrightarrow woman := female \sqcap person.$$

In this example, *woman* is a defined concept or *name* that uses the concepts *female* and *person* for definition.

Definition 4.4. A concept A *directly uses* a concept B if B appears on the right-hand side of the definition of A. The relation *uses* is the transitive closure of *directly uses*. A TBox is *cyclic* if there exists an atomic concept that uses itself. Otherwise, the TBox is *acyclic* [BN03].

In an acyclic terminology T, all defined concepts are determined by atomic concepts. This can be shown through expansion of T. The expansion of T, T', is derived by replacing each occurrence of a name for a defined concept by its right-hand side definition. It can be shown that T and T' have the same models [BN03]. This means that T is satisfiable if T' is satisfiable.

An ABox contains membership assertions like $C(c)$ or $R(a, b)$ stating that an individual c belongs to concept C or that two individuals a, b belong to a role R. The semantics of an ABox is defined by extending the interpretation $\mathcal{I}$ to individual names so that for each individual name a, $a^{\mathcal{I}}$ is an element of Δ. Such an interpretation usually relies on the unique name assumption. This means if $a \neq b$, then also $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. $\mathcal{I}$ satisfies the membership assertions of the ABox if for each $C(a)$ and $R(a, b)$ the following holds: $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ [BN03].

An interpretation $\mathcal{I}$ is a *model for a knowledge base* KB=(ABox, TBox) if it satisfies the TBox and respects the membership assertions of the ABox at the same time. An ABox is *consistent* with respect to the TBox if there exists an interpretation that is a model for both, the TBox and the ABox.

The main reasoning tasks for ABoxes are consistency checking and instance testing [BN03]. The later means to test if a given individual belongs to a given concept or, as generalization, to find all instances of a given concept.

DL relies on an open-world assumption. This means that ABoxes are not assumed to be complete. There may exist individuals x that belong to a concept C for which a membership assertion $C(x)$ does not exist. Selecting all members of a concept, for which a membership assertion exists, does not ensure that this set is complete.

In the following section, DL formalisms will be employed to improve Web service operation search with semantics. The concept hierarchy as modeled for the financial domain is translated into DL, e.g., the concept `Financial Measure` (FM) subsumes the concept `Periodic Measure` (PM), which is translated into $PM \sqsubseteq FM$. Next, the semantic annotations are interpreted as membership assertions, e.g., the operation `Compute_PV` (*cpv*) is a member of the concept `Measure Calculation` (MC), which is expressed as $MC(cpv)$. Then, this formalization is used to extend the SSQs taking the formalized knowledge of the ontology into account. A search for an operation that returns an FM is extended to search for an operation that returns an FM or a PM based on the knowledge that $PM \sqsubseteq FM$.

4.3 Searching Operations with Semantic Annotations

Searching Web service operations with semantic annotations means to find all matches for a given SSQ. For this approach, the following is needed:

- a terminology that provides a vocabulary for annotations,
- an extension to the relational model that permits semantic annotations using the terminology, as already used in Table 4.1,
- an algorithm that uses the terminology and the semantic annotations to find Web service operations that match given search criteria expressed as SSQ.

Section 4.3.1 presents the terminology and the semantic extension of the relational Web service repository. Section 4.3.2 is dedicated to the analysis of the search algorithm. Section 4.3.3 shows variations of the search algorithm to achieve different match semantics.

4.3.1 Semantic Extension of the Relational Web Service Repository

A *terminology* (TBox) contains all terms and terminological axioms used to describe an area of knowledge, e.g., a software system. In the following, parts of DL-Lite [CGL+04] are used to define a simple terminology that consists of atomic and basic concepts as well as acyclic inclusion axioms only.

Definition 4.5. A *simple terminology* $\mathcal{M} = (\mathcal{B}, \mathcal{T})$ consists of a set of basic concepts $\mathcal{B}$ and a set $\mathcal{T}$ of acyclic inclusion axioms of the form $C \sqsubseteq D$ with $C, D \in \mathcal{B}$. Basic concepts $B \in \mathcal{B}$ are defined as follows: $B ::= A | \uparrow | \downarrow$. A denotes an atomic concept. $\mathcal{B}$ is not empty. It contains at least the universal concept $\uparrow$ and the bottom concept $\downarrow$. $\mathcal{T}$ is not empty. For all concepts $B \in \mathcal{B}$ the following two inclusion axioms hold: $\downarrow \sqsubseteq B$ and $B \sqsubseteq \uparrow$.

The formal semantics of this terminology is defined as an interpretation $\mathcal{I}$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ as interpretation domain and an interpretation function $\cdot^{\mathcal{I}}$ [BN03].

Definition 4.6. The *formal semantics* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of the simple terminology $\mathcal{M} = (\mathcal{B}, \mathcal{T})$ consists of a non-empty set $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ with:

$$\begin{array}{rcl} \uparrow^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\ \downarrow^{\mathcal{I}} & = & \emptyset \\ C^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\ (C \sqsubseteq D)^{\mathcal{I}} & \Leftrightarrow & C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \end{array}$$

for $C, D \in \mathcal{B}$.

The simple terminology $\mathcal{M}$ represents intentional knowledge about the world of discourse. It is used as vocabulary to extend the relational Web service model with *semantic annotations* at the level of operations as well as input and output attributes. The semantic annotations provide extensional knowledge about individual instances in the form of operations and attributes which exist in the interpretation domain.

The Web service as a whole is not annotated as the UDDI standard already permits descriptive attributes and references to external classification systems. Semantic annotations are modeled as extensions of the relational Web service registry schema. They provide additional semantics, stating that a Web service operation or an attribute is a member of a semantic concept of $\mathcal{M}$. In this sense, a semantic annotation represents a membership assertion.

Let $\mathcal{M} = (\mathcal{B}, \mathcal{T})$ be a simple terminology, $\mathcal{S}$ an enumerable set of Web service names, $\mathcal{O}$ be an enumerable set of operation schema names, and $\mathcal{A}$ an enumerable set of attribute names. Let $\mathcal{B}$, $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{A}$ be pairwise disjoint.

Definition 4.7. *Semantic annotations for operation names* in $\mathcal{O}$ given a simple terminology $\mathcal{M}$ are defined as a function sa_O from $\mathbf{O_S} \subseteq \mathcal{O}$ to $\mathcal{B}$, $sa_O : \mathbf{O_S} \rightarrow \mathcal{B}$. $\mathbf{O_S}$ is the subset of operation names in $\mathcal{O}$ for which semantic annotations exist.

Semantic annotations for attribute names in $\mathcal{A}$, given a simple terminology $\mathcal{M}$, are defined as a function sa_A from $\mathbf{A_S} \subseteq \mathcal{A}$ to $\mathcal{B}$, $sa_A : \mathbf{A_S} \rightarrow \mathcal{B}$. $\mathbf{A_S}$ is the subset of attribute names in $\mathcal{A}$ for which semantic annotations exist.

In this definition, operation names in $\mathcal{O}$ and attribute names in $\mathcal{A}$ need not have a semantic annotation. Further, the semantic annotation is a function but it is neither injective nor onto. Several operation names or attribute names may be annotated with the same semantic element but every element from $\mathbf{O_S}$ and $\mathbf{A_S}$ is annotated with at most one concept from $\mathcal{M}$.

Definition 4.8. A *Web service schema with semantic annotations* is a pair $WS = (sname, \mathbf{O})$ consisting of a schema name $sname \in \mathcal{S}$ and a finite set $\mathbf{O}$ of operation schemata with semantic annotations.

An *operation schema with semantic annotation* is a tuple $OS = (opname,$ $\mathbf{A}$, $sa_O(opname))$ consisting of an operation name $opname \in \mathbf{O}$, a finite set $\mathbf{A} \subseteq \mathcal{A}$ of semantically annotated attributes, and a semantic annotation $sa_O(opname)$.

A *semantically annotated attribute* consists of an attribute name $name \in \mathbf{A}$, a semantic annotation $sa_A(name)$, and a type $type(A) \in \{in, out\}$.

An example for a semantically annotated Web service operation has already been presented in Section 4.1.1 in Table 4.1.

In terms of descriptions logic, the simple terminology represents the *TBox*, and the semantic annotations represent the *ABox*. Both in combination form a knowledge base $\mathcal{K} = (TBox, ABox)$.

Lemma 4.2. The knowledge base $\mathcal{K} = (TBox, ABox)$ represented by the simple terminology $\mathcal{M} = (\mathcal{B}, \mathcal{T})$ and the semantic annotation functions sa_A and sa_O is satisfiable.

Proof. A knowledge base is satisfiable if there exists a model for it that fulfills the inclusion axioms of the $TBox$ and the membership assertions of the $ABox$. This model is given by the interpretation $\mathcal{I}$ that is based on the semantic annotation functions. The construction proceeds in four steps:

- The domain Δ consists of all operation and attribute names that are annotated.

$$\Delta^I := \dot{\bigcup}_{C \in \mathcal{B}} \left(\{x | sa_A(x) = C\} \,\dot{\cup}\, \{x | sa_O(x) = C\} \right).$$

- For all atomic concepts $A \in \mathcal{B}$ that appear only on the left-hand side of inclusion axioms or that are not used in inclusion axioms at all, the interpretation consists of the operation and attributes names that are annotated with A.

$$A^{\mathcal{I}} := \{x | sa_A(x) = A\} \,\dot{\cup}\, \{x | sa_O(x) = A\}.$$

- For each inclusion axiom $A \sqsubseteq B$, the interpretation of B consists of all operation and attribute names that are annotated with B and all operation and attribute names that are annotated with A.

$$B^{\mathcal{I}} := A^{\mathcal{I}} \dot{\cup} \{x | sa_A(x) = B\} \,\dot{\cup}\, \{x | sa_O(x) = B\}.$$

- The last step is repeated until no further changes occur in the constructed sets.

By definition, the interpretation $\mathcal{I}$ fulfills the membership assertions of the ABox as given by the semantic annotations. The interpretation also fulfills the inclusion assertions of the TBox by construction [BN03]. $\square$

The satisfiability of the knowledge base, given by semantically annotated Web service operations, is the prerequisite which is needed to use semantic annotations for the semantic search of Web service operations. In the following section, an algorithm will be presented that is based on a satisfiable knowledge base as shown in Lemma 4.2. This section has served as a preparation to present and examine this algorithm.

The approach assumes that each element of the relational Web service model is annotated with at most one identifier from the semantic model. If more than one semantic model for the same world of discourse is needed, e.g., ontologies in different technical formats, this can be integrated into the model in two ways:

1. There exists a mapping between the two ontologies. Then, it is sufficient to annotate the elements from the relational Web service model with one semantic element from one of the two ontologies as defined above.

2. If the two ontologies are not mapped explicitly, then the semantic relational Web service model can be extended to allow more than one semantic annotation per element defining a second semantic annotation function for operations and attributes.

From a modeling perspective, the first solution is better because the mapping between the two ontologies is documented unambiguously, whereas in the second solution the mapping is only indirectly expressed through the semantic annotation of an interface definition. This might cause inconsistencies and is therefore not considered further.

4.3.2 Matching of Semantic Operations as Query Answering

All queries considered in Section 4.1.2 use the concepts of the semantic model as a defined vocabulary for selection. The selection does not take inclusion assertions into account. If concept C is subsumed by concept D, $C \sqsubseteq D$, all operation or attribute names annotated with C are implicitly also annotated with D. Therefore, the semantic search for all elements annotated with D is expected to return all elements annotated with C or D. The simple semantic queries as shown above do not use inclusion assertions, yet. This draw-back is amended in this section. The inclusion axioms of the semantic model are applied to the queries as rewriting rules as illustrated in the following example.

If a user searches for operations that return an initial present value, operations that return a gross initial present value or a net initial present value are also elements of the query result because the two concepts are specializations of an initial present value (see Figure 4.2). Therefore, the simple semantic query Q8, interpreted as a *semantic query*, can be rewritten as shown in Code 4.5. Applying the inclusion axioms "Gross Initial Present Value" $\sqsubseteq$ "Initial Present Value" and "Net Initial Present Value" $\sqsubseteq$ "Initial Present Value" results in a union of conjunctive queries with semantic annotations, thus, in the *simple semantic query* Q+8.

Code 4.5.

```
Q8:  Select WSNAME, OPNAME from WSSCHEMA
       where TYPE  = ``Out''
         and SAA   = ``Initial Present Value'';

Q+8: Select WSNAME, OPNAME from WSSCHEMA
      where TYPE       = ``Out''
        and    (   SAA = ``Initial Present Value''
                or SAA = ``Gross Initial Present Value''
                or SAA = ``Net Initial Present Value'');
```

This intuitive understanding of the meaning of a semantic query is formalized in the following definition.

Definition 4.9. Given a simple terminology $\mathcal{M} = (\mathcal{B}, \mathcal{T})$ with interpretation $\mathcal{I}$ as defined in Lemma 4.2 and a relational Web service registry $R = (WSNAME, OPNAME, SAO, NAME, SAA, TYPE)$ with semantically annotated Web service schemata.

Let q be a simple semantic SP-query over R with concepts C and D as selection criteria on SAO and SAA resp. Then tuple $t \in R$ is in the answer set *ans* of the *semantic interpretation* of q, $q^{\mathcal{I}}$, if $t[SAO] \in C^{\mathcal{I}}$ and $t[SAA] \in D^{\mathcal{I}}$ resp. $q^{\mathcal{I}}$ is called a *semantic* SP-query over R and $\mathcal{M}$.

Let $q = q_1 \cup \ldots \cup q_n$ be a simple semantic SPU-query over R consisting of unions of n SP-queries. Then $q^{\mathcal{I}} := q_1^{\mathcal{I}} \cup \ldots \cup q_n^{\mathcal{I}}$.

Let $q = q_1 \bowtie \ldots \bowtie q_n$ be a simple semantic SPJ query over R consisting of joins of SP- and SPU-queries. Then $q^{\mathcal{I}} := q_1^{\mathcal{I}} \bowtie \ldots \bowtie q_n^{\mathcal{I}}$.

The following algorithm applies the same rewriting technique as shown in Code 4.5 to compute the semantic interpretation of an SP-query. It rewrites the semantic SP-query $q^{\mathcal{I}}$ and returns a union of simple semantic queries. It will be shown that the answer set to the semantic query $q^{\mathcal{I}}$ is equal to the answer set of the union of simple semantic queries as returned by the algorithm. The query expansion is an adaptation of the algorithm suggested by [CGL+04]. An inclusion $A \sqsubseteq B$ is represented as a pair (A, B).

Code 4.6.

```
Query Expansion Algorithm
input:
 semantic query q of type SP,
 inclusion assertions T
output:
 union of simple semantic queries P

% Preparation step %
T+ := transitive-closure(T);

% Expansion of first concept %
P := {q};
let C be the first semantic concept in q
  for each inclusion I = (A,B) in T+ do
    % A subsumed by B %
    if C = B
    then
      % Replace C with A %
      P := P union {q[C|A]};

% Expansion of second concept %
for each query q in P
  let C be the second semantic concept in q
  if C exists
  then
    for each inclusion I = (A,B) in T+ do
      % A subsumed by B %
      if C = B
      then
        % Replace C with A %
        P := P union {q[C|A]};
return P;
```

In the first step the transitive closure of the inclusion assertions is computed. A semantic query is a conjunctive query over a relational Web service operation repository R. Such a query has at most two semantic concepts in the `where`-clause. The first concept is expanded in the first step. If a second concept exists, it is expanded in the second step.

Informally speaking, the suggested method of query rewriting compiles the knowledge of the semantic model into the query so that the query can then be answered like any other union of conjunctive queries over a relational database.

Lemma 4.3. Given a semantic query $q^{\mathcal{I}}$. The query expansion algorithm returns a simple semantic query q'.

Proof. Disregarding the interpretation $\mathcal{I}$, a semantic SP-query $q^{\mathcal{I}}$ is also a simple semantic SP-query q. The query rewriting algorithm produces a union of SP-queries. Thus, according to Definition 4.1 the algorithm returns a simple semantic query. $\square$

With these preparations, correctness of the re-writing algorithm can be shown. The algorithm is correct iff the original semantic query and the rewritten simple semantic query return the same result set.

Theorem 4.1. *Given a satisfiable simple semantic terminology $\mathcal{M} = (\mathcal{C}, \mathcal{T})$, a schema repository for semantically annotated Web services R and the annotation functions sa_O and sa_A. Let $q^{\mathcal{I}}$ be a semantic SP-query over R and q' the simple semantic query returned by the query expansion algorithm. Then $t \in ans(q^{\mathcal{I}})$ iff $t \in ans(q')$.*

Proof. The following proof runs along the proof sketch given in [CGL$^+$04].

If $t \in ans(q')$ then $t \in ans(q^{\mathcal{I}})$. This is obvious as q' is obtained using the inclusion assertions of the simple semantic model.

To proof the other direction, a chase-like technique is applied to the schema repository R. The chase of R, $chase(R)$, is obtained from R by applying the following chase rule: if concept B is contained in concept D, $B \sqsubseteq D$, and B is used to annotate a Web service schema WS in R then copy the entries for WS and insert them into $chase(R)$, replacing the semantic annotation B with D. This is repeated until the set $chase(R)$ does not change any more. The chase algorithm terminates and produces a unique result because the inclusion assertions are acyclic [AHV95]. Further, if $\mathcal{M}$

is satisfiable,, then $chase(R)$ is a representative of all models that satisfy $\mathcal{M}$ [CGL+04]. Thus, if t is an element of $chase(R)$ and t is in the result set of the semantic query $q^{\mathcal{I}}$ over R then t is also in the result set of the simple semantic query q' over $chase(R)$ and vice versa. □

Theorem 4.2. *The query expansion algorithm runs in time polynomial in the number m of inclusion assertion of the simple semantic terminology. It returns at most $O(m^2)$ unions of conjunctive queries for a semantic SP-query as defined in Section 4.1.2.*

Proof. If a semantic terminology has m inclusion assertions, then at most $m + 1$ *different* concepts are used to express the inclusion assertions.

The first step of the algorithm computes the transitive closure of the inclusion assertions. This computation is polynomial in m, e.g., $O(m^3)$ with Warshall's algorithm.

As the inclusion assertions are acyclic, a concept C contains at most a number of sub-concepts linear in m after the transitive closure is computed.

The second step, the expansion of the first concept, therefore produces a number of conjunctive queries which is linear in m.

The third step, the expansion of the second concept, is executed a number of times linear in the number of conjunctive queries as returned by step two. The inner loop returns a number of conjunctive queries which is linear in m. Therefore, the third step of the algorithm is in $O(m^2)$ and returns $O(m^2)$ conjunctive queries.

Summing up, the algorithm is polynomial in m for a semantic SP-query. □

Computing the transitive closure is the most "expensive" part of the algorithm. However, the transitive closure may be computed only once so that the result can be stored and reused. As long as the terminology does not change, a new computation of the transitive closure is not necessary.

The simple terminology may degenerate in two ways: Either it is flat and consists of concepts, which are all derived from the general top-level concept, or it is a deep linear list. These two extreme variations are depicted in Figure 4.10.

Assuming that the semantic SP-query references the top-level concept, the algorithm returns $O(m^2)$ conjunctive queries. As soon as the SP-query does not reference the top-level concept, but the more specific concept A_i, the algorithm will return only one conjunctive query in the first case and $O(m - i)^2$ queries in the second case. In practice, the construction of a terminology tries to avoid these two degenerate cases. Further, it is desirable to use

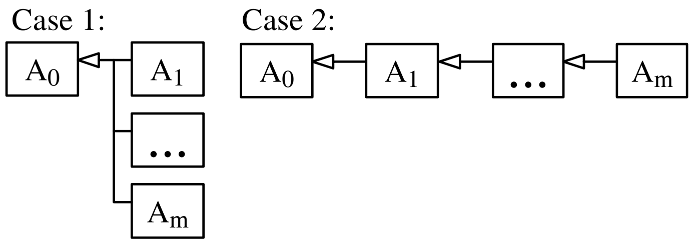


Figure 4.10: Degenerated terminologies.

more specific concepts for querying than the top-level concept. Therefore, the worst case behavior is unlikely.

Next, the computational complexity of query answering is analyzed. For that purpose, three parameters that influence complexity are distinguished:

1. The size n of the repository R determines the data complexity.
2. The number m of inclusion axioms in the simple semantic terminology determines the size of the TBox.
3. The length $|q|$ of the query q determines the expression complexity.

In general, the length of a query q can be measured in terms of

- the number of variables in the `select`-clause,
- the number of relations and their arity in the `from`-clause, or
- the length of the query string itself.

For a semantic query q, the number of variables in the `select`-clause is fixed as the examples in Section 4.1.2 show. The number of relations and their arity is also fixed because all queries are posed against the Web service operation repository R. The only applicable measure of expression complexity is the length of the query string that is returned by the rewriting algorithm. This is determined by the number of inclusion assertions in the terminology. For each inclusion assertion a new union is added. Therefore, the length $|q|$ is dependent on m, the number of inclusion axioms. As a consequence, the combined complexity of answering a semantic query is defined by two independent parameters:

Lemma 4.4. The combined complexity of answering a semantic query is determined by the data complexity of the repository and the number of inclusion axioms in the simple semantic terminology.

Theorem 4.3. *Given a service operation repository R with n rows, a simple terminology with m acyclic inclusion assertions and a semantic query q. Answering a semantic query q is in $O(f(m) * n^c)$ with f polynomial and c constant.*

Proof. The complexity result is established for each of the three query types identified in Section 4.1.2, SP, SPU, and SPJ.

Queries of type SP, regarded as simple semantic queries, are selections, thus, each such query is evaluated in $O(n)$. A simple semantic query has at most two semantic annotations in the `where`-clause for the operation itself and either one in- or one out-attribute. Therefore there are at most $f(m) = i * m^2$ with i constant simple semantic queries after the execution of the query expansion algorithm, as already argued in the proof of Theorem 4.3.

Queries of type SPU consist of j queries which are each of type SP. Therefore, $f(m) = i * j * m^2$. Although, in theory, the number of unions in a SPU is unbounded the disjunction is used to execute a selection according to different concepts of the terminology. Therefore, it is safe to assume that $j \leq m^2$.

Queries of type SPJ consist of c acyclic joins of queries of type SP or SPU. Therefore, the result set can be computed in $O(f(m) * n^c)$ with $f(m) = i * j * m^{2c}$, i, j as above. Although, in theory, the number of joins in a SPU is unbounded, the join is used to select operations that fulfill several selection criteria at once. Therefore, c can be bounded by the maximal number of input (in) and output (out) attributes that a Web service operation in R has, $c \leq 1 + in + out$. □

Although answering arbitrary conjunctive queries over an arbitrary database schema is NP-complete in combined complexity [AHV95], this is not the case for semantic queries as defined in Section 4.1.2 over a relational Web service operation repository. The size of the terminology and the number of Web service operations result in polynomial time complexity.

For the finance example, the application of this search strategy becomes feasible if the size of the terminology does not outsize the number of available Web service operations. If the terminology is too detailed so that each bottom-level concept references exactly one operation, many queries are generated that return an empty result set or just one element. The distinction between gross initial present value and net initial present value is an example for a level of detail that might not be needed more than once. This trade-off between the detailedness of the taxonomy and the number of services needs to be observed.

4.3.3 Variations of Semantic Operation Matching

Query rewriting can be applied to find more specialized and more general results to a given query. This technique is applied to different types of operation searches. In preparation, a hierarchy of matches based on the inclusion axioms is defined:

Match Hierarchy In a terminology as defined in Definition 4.5, concepts may overlap. Based on the interpretation of the semantic model as defined in Section 4.3.1, the following match hierarchy is defined:

1. If $A \equiv B$, then concept B is an *exact match* for concept A.
2. If $B \sqsubseteq A$, then concept B is a *specialized match* for concept A.
3. If $A \sqsubseteq B$, then concept B is a *generalized match* for concept A.
4. If A overlaps B, then concept B is a *partial match*, i.e., $\exists a \in A^I : a \notin B^I \wedge \exists b \in B^I : b \notin A^I$.

First, queries that return exact and specialized matches are examined.

Subsumption Matches A subsumption match to a given query is described as:

- The input concepts of the returned results are exact or specialized matches of the original input concepts of the query.
- The operation concepts of the returned results are exact or specialized matches of the original operation concept of the query.
- The output concepts of the returned results are exact or specialized matches of the original output concepts of the query.

This means query rewriting is applied to input, operation, and output concepts alike, traversing the inclusion axioms from left to right.

A search like the following query for a measure calculation operation that is annotated with, “Initial Present Value”is then extended by taking all subconcepts of “Initial Present Value” into account.

Code 4.7.

```
Q9: Select subsuming WSNAME, OPNAME
      from WSSCHEMA
     where TYPE = ‘‘Out’’
       and SAA  = ‘‘Initial Present Value’’;
```

The direction of semantic expansion is indicated by the keyword “subsuming” in the SQL pseudo-query Q9. The translation into an expanded SQL query is shown in Code 4.5.

The returned results can be ranked qualitatively according to the hierarchy of matches as established above. Research results from other areas, such as Information Retrieval, can be used to compute a quantitative ranking, e.g., computing a weight for each annotation and computing a rank from the weights [Fer03].

Plug-In Matches So far, exact and subsumption matches have been considered. As the search is a search for operations, also plug-in matches are desirable. A plug-in match is an operation into which the sought operation can be “plugged-in”. This means the input concepts of the operation sought must be subsumed by the operation found. The output concepts of the operation found must be subsumed by the operation sought. The operation concept found is a subconcept or contained in the operation concept sought to avoid too general matches. This is the type of match that has already been considered in Section 3.1.1. In the relational model with semantic annotations, a plug-in match is defined as follows:

- The input concepts of the returned results are exact or *generalized* matches of the original input concepts of the query.
- The operation concepts of the returned results are exact or specialized matches of the original operation concepts of the query.
- The output concepts of the returned results are exact or specialized matches of the original output concepts of the query.

In the following example a plug-in match for a measure calculation operation is wanted that calculates the initial present value for consumer loans.

Code 4.8.

```
Q10: Select plugIn * from
        (Select WSNAME, OPNAME from WSSCHEMA
          where SAO  = ``Measure Calculation''
            and TYPE = ``In''
            and SAA  = ``Consumer Loan'')
               IN_VW,
        (Select WSNAME, OPNAME from WSSCHEMA
          where SAO  = ``Measure Calculation''
            and TYPE = ``Out''
            and SAA  = ``Initial Present Value'') OUT_VW
         where IN_VW.WSNAME = OUT_VW.WSNAME
           and IN_VW.OPNAME = OUT_VW.OPNAME;

Q+10: Select * from
          (Select WSNAME, OPNAME from WSSCHEMA
            where SAO    = ``Measure Calculation''
              and TYPE   = ``In''
              and (  SAA = ``Consumer Loan''
                  or SAA = ``Deferred Payment Loan''
                  or SAA = ``Fixed Condition Loan''
                  or SAA = ``Loan Product''
                  or SAA = ``Traditional Product''
                  or SAA = ``Depot B''
                  or SAA = ``Financial Product''))
                           IN_VW,
          (Select WSNAME, OPNAME from WSSCHEMA
            where SAO    = ``Measure Calculation''
              and TYPE   = ``Out''
              and (  SAA = ``Initial Present Value''
                  or SAA = ``Gross Initial Present Value
                     ''
                  or SAA = ``Net Initial Present Value''
                     )) OUT_VW
            where IN_VW.WSNAME = OUT_VW.WSNAME
              and IN_VW.OPNAME = OUT_VW.OPNAME;
```

The input attribute annotation is expanded to include all superconcepts. The operation concept is not expanded because the example as modeled so far does not show any subconcepts for the measure calculation operation. The output parameter annotation is expanded to include all subconcepts.

The direction of semantic expansion of the query is indicated by the keyword "plugIn" in the pseudo-code query Q10. Q+10 is the rewritten expanded SQL query.

The algorithm as presented in Section 4.3.2 can be modified without increasing its overall complexity.

Code 4.9.

```
Query Expansion Algorithm Plug In
input:
 semantic SP-query q,
 inclusion assertions T
output:
 union of simple semantic queries P

% Preparation step %
T+ := transitive-closure(T);

% Expansion of first concept %
P := {q};
let C be the first semantic concept in q
if C is an input concept
then
  for each inclusion I = (A,B) in T+ do
    % A subsumed by B %
    if C = A
    then
      % Replace C with B %
      P := P union {q[C|B]};
else
   for each inclusion I = (A,B) in T+ do
    % A subsumed by B %
    if C = B
    then
      % Replace C with A %
      P := P union {q[C|A]};

% Expansion of second concept %
for each query q in P
  let C be the second semantic concept in q
  if C exists
  then
    if C is an input concept
    then (
      for each inclusion I = (A,B) in T+ do
        % A subsumed by B %
        if C = A
        then
          % Replace C with B %
          P := P union {q[C|B]};)
    else (
      for each inclusion I = (A,B) in T+ do
        % A subsumed by B %
        if C = B
        then
          % Replace C with A %
          P := P union {q[C|A]};)
return P;
```

Sequence Suggestions Semantic annotations can also be used to make suggestions to the designer and retrieve operations that can be executed in sequence with a given operation. For example, the designer searches for an operation that takes deferred payment loan as input and returns its initial present value. If such an operation does not exist, the search is split up in two parts. First, operations are detected that return an initial present value and their input parameters are determined, e.g., a cashflow. In this case, the second step of the search is executed to find operations that take a deferred payment loan as input and return a cashflow or a more specialized concept as output to combine the two operations found.

A graphical example is given in Figure 4.11. In general, a sequence search is described as follows. The developer searches for an exact match first:

Code 4.10.

```
Q11: Select * from
        (Select WSNAME, OPNAME from WSSCHEMA
          where SAO  = ``Measure Calculation''
            and TYPE = ``In''
            and SAA  = ``Deferred Payment Loan'')
               IN_VW,
        (Select WSNAME, OPNAME from WSSCHEMA
          where SAO  = ``Measure Calculation''
            and TYPE = ``Out''
            and SAA  = ``Initial Present Value'') OUT_VW
```

```
      where IN_VW.WSNAME = OUT_VW.WSNAME
        and IN_VW.OPNAME = OUT_VW.OPNAME;
```

Based on the returned result, the user starts a new search to find all operations that can be plugged into the input parameter of the operation found:

Code 4.11.

```
Q12: Select subsuming WSNAME, OPNAME from WSSCHEMA
      where TYPE = ``Out''
        and SAA  = ``Deferred Payment Loan'';
```

Alternatively, the user searches for all operations that the output parameters of the operation found can be plugged into:

Code 4.12.

```
Q13: Select plugIn WSNAME, OPNAME from WSSCHEMA
      where TYPE = ``In''
        and SAA  = ``Initial Present Value'';
```

This shows that the suggested simple semantic terminology can be used flexibly for different search use cases to compute matching Web service schemata based on semantic annotations. The search can be expressed as semantic queries which can be rewritten as a union of simple semantic queries to include the knowledge of the semantic terminology. The different search types have been examined separately for ease of exposition to show that the algorithm presented in Section 4.3.2 can be flexibly applied to different search situations. It is possible, to combine the different search types and to generate a unified result.

So far, the semantic terminology expresses generalizations only without any further constraints or information. It exists independently of the implemented Web service operations. Extensions to this type of semantic terminology are analyzed in the next section.

4.4 Terminological Extensions

The terminology as used so far serves as a controlled vocabulary for querying the relational schema repository. It does not imply any restrictions on the developers' usage of it.

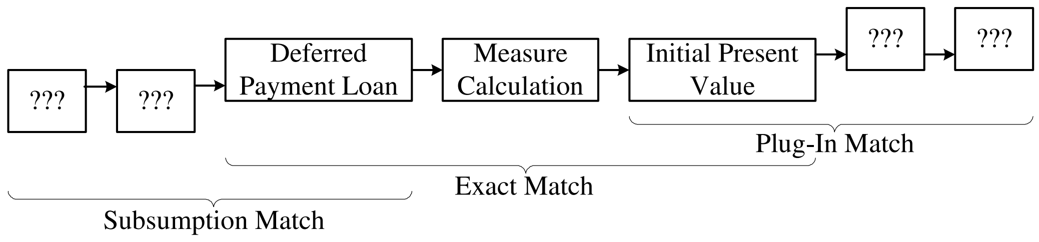


Figure 4.11: Sequence of match queries.

In Section 4.4.1, a distinction between operations and attributes is added. In the simple semantic terminology, the designer is not restricted in the use of the annotations, but can freely annotate each operation and attribute with any concept. As the terminology does not distinguish operation concepts from attribute concepts, it is even possible to use an operation concept for attribute annotation. An extension to the simple semantic terminology is added to provide a consistency check.

In Section 4.4.2 structured attribute concepts are introduced and also used for query rewriting. A concept like a loan often consists of other concepts that are a part of it, e.g., the loan amount or the interest rate. In object-oriented data models, this is expressed as attributes of a class. In other programming languages, this is modeled as a structured record. In the given relational model, it is not possible to represent structured attributes, and in the semantic model, the parts of a concept are not described either. Therefore, the simple semantic terminology is extended to contain information about properties of a concept.

4.4.1 Distinguishing between Operations and Attributes

So far the semantic terminology does not distinguish between operation and attribute concepts, but treats all semantic annotations alike. The only way to find out if a concept is an operation is to find an interface description that uses this concept for semantic annotation of an operation name. This information is only contained in the relational schema repository. This is a source of inconsistency because in the relational model it is even possible that a concept is used to annotate an input or output attribute and an operation name at the same time. Neither the semantic terminology nor the relational model prevent this inconsistent usage.

Therefore, the distinction between operation and attribute concepts is, first of all, a support for the programmer who wants to annotate an existing Web service operation. Additionally, the distinction between operation and attribute concepts splits the terminology into two parts. Thus, it helps to prevent the degenerated terminology cases as discussed in Section 4.3.2 because the query expansion of an attribute concept excludes all operation concepts and vice versa.

A desirable extension to the semantic model is the capability to distinguish between operation and attribute concepts. To achieve this distinction, the definition of the semantic model and the semantic annotations is extended as follows:

Definition 4.10. An *extended semantic terminology* $\mathcal{M} = (\mathcal{C}, \mathcal{T})$ is a semantic terminology as defined in Definition 4.5. $\mathcal{C}$ contains the universal concept $\uparrow$, the bottom concept $\downarrow$, a distinguished atomic operation concept OP, a distinguished atomic attribute concept AT, $OP, AT \in \mathcal{C}$ with $AT \sqsubseteq \neg OP$, $OP \sqsubseteq \neg AT$, and $OP, AT \sqsubseteq \uparrow$. For the set $\mathcal{T}$ of acyclic inclusion axioms the following holds: $\nexists C \in \mathcal{C} \setminus \{\uparrow, \downarrow\} : AT \sqsubseteq^+ C$ and $\nexists C \in \mathcal{C} \setminus \{\uparrow, \downarrow\} : OP \sqsubseteq^+ C$.

This means that the set of concepts $\mathcal{C}$ contains two disjoint subsets: ${\mathcal{C}_\mathcal{O}}^\mathcal{I}$ contains operations, ${\mathcal{C}_\mathcal{A}}^\mathcal{I}$ contains attributes, $\mathcal{C}_\mathcal{O}^\mathcal{I} \cap {\mathcal{C}_\mathcal{A}}^\mathcal{I} = \emptyset$. All concepts are either operation concepts or attribute concepts. The necessary change in the semantic terminology of the example scenario is depicted in Figure 4.12. The hierarchy of concepts is split into two disjoint directed acyclic graphs with a general operation concept reps. attribute concept as root.

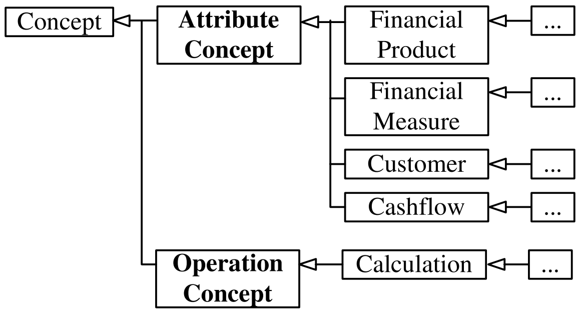


Figure 4.12: Extended concept hierarchy.

The formal semantics used so far in Definition 4.6 needs to be extended to include the interpretation of negation:

Definition 4.11. The *formal semantics* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of the extended terminology $\mathcal{M} = (\mathcal{C}, \mathcal{T})$ is defined as in Definition 4.6 with the following extension:

$$(\neg C)^\mathcal{I} \quad = \quad \Delta \setminus C^\mathcal{I}$$

The definition of semantic annotations needs to be extended to express the following two constraints:

- Operation names may be annotated with operation concepts only.
- Attribute names may be annotated with attribute concepts only.

Definition 4.12. Semantic annotations for operation names in $\mathcal{O}$, given an extended semantic model $\mathcal{M}$, are defined as a function sa_O from $\mathbf{O_S} \subseteq \mathcal{O}$ to $\mathcal{C}_\mathcal{O}$, $sa_O : \mathbf{O_S} \rightarrow \mathcal{C}_\mathcal{O}$. $\mathbf{O_S}$ is the subset of operation names for which semantic annotations exist. $\mathcal{C}_\mathcal{O}$ is the set of operation concepts in the extended terminology $\mathcal{M}$.

Semantic annotations for attribute names in $\mathcal{A}$, given a semantic model $\mathcal{M}$, are defined as a function sa_A from $\mathbf{A_S} \subseteq \mathcal{A}$ to $\mathcal{C}_\mathcal{A}$, $sa_A : \mathbf{A_S} \rightarrow \mathcal{C}_\mathcal{A}$. $\mathbf{A_S}$ is the subset of attribute names for which semantic annotations exist. $\mathcal{C}_\mathcal{A}$ is the set of attribute concepts in the extended terminology $\mathcal{M}$.

The algorithm for query expansion does not need to be extended because negative inclusion axioms do not contribute to a solution. They are not used in the inclusion algorithm. However, before the algorithm can be executed it must be checked if the knowledge base of concepts, positive and negative inclusion axioms and of semantic annotations is satisfiable.

As already shown in Lemma 4.2 a model can be constructed from the semantic annotations that fulfills the membership assertions and the positive inclusion axioms. With the negative inclusion axioms it must be ensured that the negative inclusions do not make the knowledge base unsatisfiable.

For this, it must be tested if there exists a semantic annotation of an attribute name that is annotated with an operation concept or an operation name that is annotated with an attribute concept. In preparation of this test, the set of inclusion assertions must be closed with respect to the following rules [CGL+04]:

$$\begin{aligned} A \sqsubseteq B \wedge B \sqsubseteq \neg C &\quad\Rightarrow\quad A \sqsubseteq \neg C \\ A \sqsubseteq B \wedge C \sqsubseteq \neg B &\quad\Rightarrow\quad A \sqsubseteq \neg C \end{aligned}$$

Lemma 4.5. The original knowledge base and the knowledge base closed with respect to the closure rules have the same models [CGL+04].

Proof. For each pair of inclusion axioms $A \sqsubseteq B$ and $B \sqsubseteq \neg C$ the following holds:

$$\begin{aligned} & A \sqsubseteq B \wedge B \sqsubseteq \neg C \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqcap C = \emptyset \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqcap C = \emptyset \wedge A \sqcap C = \emptyset \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqsubseteq \neg C \wedge A \sqsubseteq \neg C \end{aligned}$$

For each pair of inclusion axioms $A \sqsubseteq B$ and $C \sqsubseteq \neg B$ the following holds:

$$\begin{aligned} & A \sqsubseteq B \wedge C \sqsubseteq \neg B \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqcap C = \emptyset \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqcap C = \emptyset \wedge A \sqcap C = \emptyset \\ \Leftrightarrow\ & A \sqsubseteq B \wedge B \sqsubseteq \neg C \wedge A \sqsubseteq \neg C \end{aligned}$$

Therefore, the original knowledge base and the knowledge base closed with respect to the above closure rules have the same models. □

The only negative inclusion assertions in the extended semantic terminology are $AT \sqsubseteq \neg OP$ and $OP \sqsubseteq \neg AT$. Therefore, closing the inclusion assertions with respect to the above rules leads to a set of negative inclusion assertions of the form $C \sqsubseteq \neg OP$ and $C \sqsubseteq \neg AT$ where C is some subconcept of AT, respectively OP. For each negative inclusion assertion $C \sqsubseteq \neg OP$ the following conjunctive query is executed: `Select * from WSSCHEMA where SO = C`. For each negative inclusion assertion $C \sqsubseteq \neg AT$ the following conjunctive query is executed: `Select * from WSSCHEMA where SA = C`.

Lemma 4.6. The knowledge base is consistent if each query returns the empty set.

In this case, the knowledge base is satisfiable and the model can be constructed as in Lemma 4.2.

The distinction between operation and attribute concepts in the semantic model influences the set of available semantic annotations for Web service operations. The information of the ontology allows to restrict the semantic annotations and make a consistency check once the programmer has annotated the interface. This consistency check can prevent the programmer from using operation concepts as annotations for attributes and vice versa. This enhances the quality of the program design and also of the ontology because the designers of both are forced to reflect which role a concept has. This is the first extension suggested for the semantic model.

The next extension integrates structured attribute concepts that possess other attribute concepts as parts.

4.4.2 Using Attributes with Properties

In the given relational model attributes with properties are not defined explicitly. If a cashflow for a loan is to be computed in an operation the input parameter can either be a single input parameter annotated with the atomic concept "loan" or the operation can have several input parameters such as net value, opening date, closing date, installment, installment periodicity, and interest rate, each annotated with its own atomic concept of the semantic model. These concepts represent a loan as well because a loan has the property to have a net value, an opening date, a closing date, etc. If the user searches for an operation that takes a net value as input parameter then operations that have an annuity loan or an amortization loan as input parameter are missed although every such loan has the property to have a net value. Neither the semantic model nor the interface description provide this information about properties of concepts.

In the following the semantic terminology is extended to include participation constraints, stating that all instances of a concept take part in a relation as first component denoting that all instances of this concept have a certain property. For example, all annuity loans and also all amortizable loans have a gross interest rate (GIR). They take part in the relation *hasGIR* on one side and there might be even more concepts that have a gross interest rate. On the other side, the concept gross interest rate is always the second concept that belongs to the relation *hasGIR*. This is a type constraint. The following definition formalizes the example.

Definition 4.13. A *structured semantic terminology* $\mathcal{M} = (\mathcal{C}, \mathcal{R}, \mathcal{T})$ is a semantic terminology as defined in Definition 4.10 with an additional set $\mathcal{R}$ of binary relations called *roles*.

The concepts $C \in \mathcal{C}$ are defined as follows:

$$C := \uparrow \mid \downarrow |AT|OP|A|\neg A|\exists R|\exists R^-$$

with A atomic concept, negations of a concept $\neg A$, $\exists R$ the set of instances that are first components of a binary role, $\exists R^-$ the set of instances that are the second component of a binary role. In particular, the universal concept $\uparrow$, the bottom concept $\downarrow$, the distinguished atomic operation concept OP, and the distinguished attribute concept AT are elements of $\mathcal{C}$.

The elements of $\mathcal{T}$ consist of acyclic inclusion assertions of the form:

- $A \sqsubseteq B$ where A, B are atomic concepts,
- $OP \sqsubseteq \neg AT$ stating that the distinguished concept OP is disjoint from the distinguished concept AT
- $A \sqsubseteq \exists R$, stating that all instances of concept A participate in the relation R as first component,
- $\exists R^{-} \sqsubseteq A$, stating that the second component of R is of type A,
- $R \sqsubseteq P$, stating that relation R is a subrelation of P.

Roles are graphically depicted as arrow with a round head, indicating that two concepts form a role-relation as depicted in Figure 4.13. The concept that the arrow head points to is the left-hand side the role, the other concept is the right-hand side of the role. For example, in Figure 4.13 an annuity loan has a gross interest rate. The role between the concepts annuity loan and gross interest rate is called *hasGIR*. As a gross interest rate is a specialization of an interest rate an annuity loan also has an interst rate. This relation is called *hasIR*. *hasGIR* is a sub-role of *hasIR*.

The formal semantics of the extended semantic model needs to be extended a second time to cover the structured semantic model.

Definition 4.14. The *formal semantics* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of the structured semantic model $\mathcal{M} = (\mathcal{C}, \mathcal{T})$ is defined as in Definition 4.11 with the following extension:

$$\begin{array}{rcl}(R)^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (\exists R)^{\mathcal{I}} & = & \left\{c|\exists c' : (c, c') \in R^{\mathcal{I}}\right\} \\ (\exists R^{-})^{\mathcal{I}} & = & \left\{c|\exists c' : (c', c) \in R^{\mathcal{I}}\right\} \\ (R \sqsubseteq P)^{\mathcal{I}} & = & (R)^{\mathcal{I}} \subseteq (P)^{\mathcal{I}}\end{array}$$

Notice that the definition of semantic annotations is not extended to include roles. It is only possible to annotate a Web service operation with concepts but not with roles. This means it is not possible to use $\exists R$, $\exists R^{-}$, or R as annotation.

Lemma 4.7. The knowlede base $\mathcal{K} = (TBox, ABox)$ as represented by the semantic terminology $\mathcal{M} = (\mathcal{C}, \mathcal{R}, \mathcal{T})$ and the semantic annotation function sa_a, sa_o is satisfiable.

Proof. As the relations are not used for semantic annotations the proof starts with a closure of the TBox as shown in Lemma 4.5 to test consistency. Then the model is constructed as shown in Lemma 4.2. □

Figure 4.13 shows a part of the financial product hierarchy. A fixed condition loan (FCL) has an installment (I), periodicity of installment payments (ILP), and an interest rate (IR). The interest rate has two subconcepts, net interest rate (NIR) and gross interest rate (GIR). A deferred payment loan (DPL) has a total debt (TD) whereas a annuity loan (ANL) and an amortizable loan (AML) both have a net value (NV). In addition, an amortizable loan has a periodicity of interest payments (IRP).

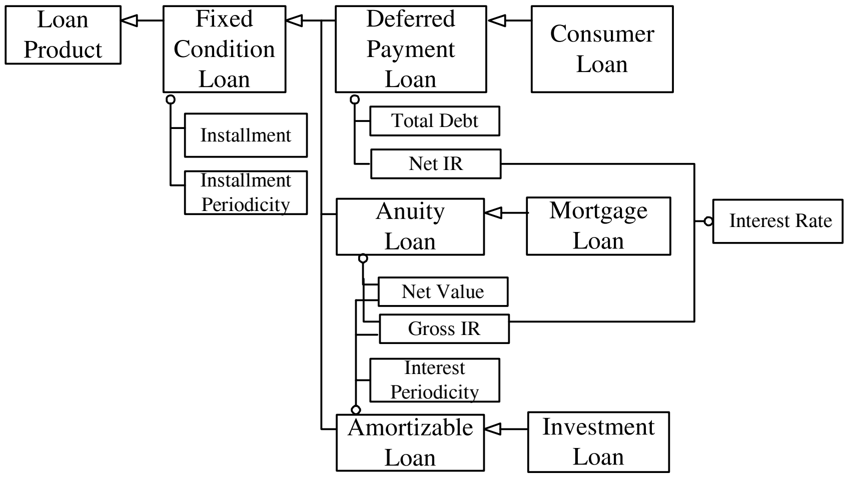


Figure 4.13: Financial product hierarchy with roles.

The graphical depiction of the semantic model is formalized as follows:
Atomic concepts: FCL, I, ILP, IR, NIR, GIR, DPL, TD, ANL, AML, NV, IRP
Roles: hasI, hasILP, hasIR, has GIR, hasNIR, hasTD, hasNV, hasIRP
Atomic concept inclusions:

$$\begin{aligned} DPL &\sqsubseteq FCL \\ ANL &\sqsubseteq FCL \\ AML &\sqsubseteq FCL \end{aligned}$$

Role inclusions:

$$\begin{aligned} hasGIR &\sqsubseteq hasIR \\ hasNIR &\sqsubseteq hasIR \end{aligned}$$

Participation inclusions:

$$\begin{aligned} FCL &\sqsubseteq \exists hasI \\ FCL &\sqsubseteq \exists hasILP \\ DPL &\sqsubseteq \exists hasNIR \\ DPL &\sqsubseteq \exists hasTD \\ ANL &\sqsubseteq \exists hasGIR \\ ANL &\sqsubseteq \exists hasNV \\ AML &\sqsubseteq \exists hasNV \\ AML &\sqsubseteq \exists hasIRP \\ AML &\sqsubseteq \exists hasGIR \end{aligned}$$

Type inclusions:

$$\begin{aligned} \exists hasI^{-} &\sqsubseteq I \\ \exists hasILP^{-} &\sqsubseteq ILP \\ \exists hasIR^{-} &\sqsubseteq IR \\ \exists hasTD^{-} &\sqsubseteq TD \\ \exists hasNV^{-} &\sqsubseteq NV \\ \exists hasIRP^{-} &\sqsubseteq IRP \end{aligned}$$

As the concept interest rate has the sub-concepts net interest rate and gross interest rate a deferred payment loan which has a net interest rate also has an interest rate. All sub-concepts inherit the roles of the super-concepts, e.g., an amortizable loan also has an installment like a fixed condition loan. A net value is part of an amortizable loan and an annuity loan.

Note that this type of unqualified existential quantification does not allow to restrict the domain of a sub-property according to the context in which it appears. The above model expresses the following: All deferred payment loans have a net interest rate. All amortizable and annuity loans have a gross interest rate. As having a net or a gross interest rate is a sub-property of having an interest rate, all deferred payment loans, all amortizable and all annuity loans also have an interest rate. This does not completely define the domain of the property *hasIR*. It only says that the domain of this property contains all deferred payment loans, all amortizable and all annuity loans. This need not be all classes that have an interest rate.

Intuitively, a query that searches operations with a net value *as part of* the input concept ought to take the participation constraints into account as shown in the following code example:

Code 4.13.

```
Q14:  Select WSNAME, OPNAME from WSSCHEMA
       where TYPE = ''In''
         and SAA  = ''exists hasNV'';

Q+14: Select WSNAME, OPNAME from WSSCHEMA
       where TYPE    = ''In''
         and (   SAA = ''Annuity Loan''
              or SAA = ''Mortgage Loan''
              or SAA = ''Amortizable Loan''
              or SAA = ''Investment Loan'';
```

For the replacement, the following inclusions are applied:

- $MortgageLoan \sqsubseteq ANL \sqsubseteq \exists hasNV$
- $InvestmentLoan \sqsubseteq AML \sqsubseteq \exists hasNV$

Finally $\exists hasNV$ is removed from the query as the schema repository does not contain this type of concept definitions as semantic annotation but only atomic concepts are allowed.

Searching for all operations with an interest rate as part of the input concept ought to take the inclusion axioms and the participation constraints into account:

Code 4.14.

```
Q15:  Select WSNAME, OPNAME from WSSCHEMA
       where TYPE = ''In''
         and SAA  = ''exists hasIR'';

Q+15: Select WSNAME, OPNAME from WSSCHEMA
```

```
       where TYPE    = ``In''
         and (   SAA = ``Deferred Payment Loan''
              or SAA = ``Consumer Loan''
              or SAA = ``Annuity Loan''
              or SAA = ``Mortgage Loan''
              or SAA = ``Amortizable Loan''
              or SAA = ``Investment Loan'';
```

For the replacement, the following inclusions are applied:

- $ConsumerLoan \sqsubseteq DPL \sqsubseteq \exists hasGIR \sqsubseteq \exists hasIR$
- $MortgageLoan \sqsubseteq ANL \sqsubseteq \exists hasNIR \sqsubseteq \exists hasIR$
- $InvestmentLoan \sqsubseteq AML \sqsubseteq \exists hasNIR \sqsubseteq \exists hasIR$

The algorithm is extended to include subproperties and participation inclusions:

Code 4.15.

```
Query Expansion Algorithm
iinput:
 semantic SP-query q,
 inclusion assertions T
output:
 union of simple semantic queries P

% Preparation step %
T+ := transitive-closure(T);

% Expansion of first concept %
P := {q};
let C be the first semantic concept in q
  for each inclusion I = (A,B) in T+ do
    % A subsumed by B %
    if C = B
    then
      % Replace C with A %
      P := P union {q[C|A]};

% Expansion of second concept %
for each query q in P
  let C be the second semantic concept in q
  if C exists
  then
    for each inclusion I = (A,B) in T+ do
      % A subsumed by B %
      if C = B
      then
        % Replace C with A %
        P := P union {q[C|A]};

% Removal of participation concepts %
  for each query q in P
    for each concept C in q do
      if C = 'exists␣X'
      then P := P - {q}
return P;
```

As before, for subsumption queries the graph is traversed as shown in the algorithm. For PlugIn queries the graph is traversed as shown in Code 4.9.

Lemma 4.8. Participation constraints and sub-role relationships do not increase the complexity of answering semantic queries over a given semantic terminology $\mathcal{M} = (\mathcal{C}, \mathcal{R}, \mathcal{T})$ and a relational Web service operation repository R with semantic annotations.

Proof. The first part of the algorithm has not changed. The additional loop to remove participation constraints is linear in the number of unions of conjunctive queries returned by the first part because each query contains at most two concepts. Therefore, the complexity is not increased. □

4.5 Summary

In this chapter a semantic terminology has been introduced that allows to express ISA-relationships, disjointness constraints, participation, and type constraints. These are common properties of many data models, such as ER-diagrams or UML-class diagrams. Further, taxonomies can be expressed with this terminology which are common classification schemes in many fields of science and industry.

The terminology has been used to define the meaning of semantic queries searching for matching Web service operations over a relational Web service operation repository which has been extended to contain semantic annotations. The proposed Web service matching approach based on semantic annotations has the following advantages:

- The semantic constructs used to model Web services arise from business semantics and are consistent with enterprise wide business knowledge.
- The Web service and the semantic annotations exist independently of each other.
- The approach is not restricted to a specific ontology or technical standard because the semantic model is a general model, yet, all DL constructs used in the approach can be mapped to, e.g., RDFS.
- The semantic model is applied for efficient query expansion.
- The query expansion is able to express subsumption and plug-in matches.
- It is also applicable to make sequence suggestions.
- The semantic model can be used to check the consistency of user annotations.
- The semantic model is extendable to contain roles and make use of roles for searching although roles are not used for annotations.

It has been shown that answering typical semantic queries over a relational Web service operation repository is in $O(f(m)n^c)$, with $f(m)$ polynomial in the number m of terminological concepts, n number of entries in the repository and c constant. This shows that the relational model of a Web service operation repository uses the standardization that WSDL interface descriptions have to offer and combines it with the efficiency of a relational database management system. Further, decoupling the technical WSDL interface description from its semantic description allows to exchange the semantic terminology without changing the technical specification.

Chapter 5

Related Work

In this chapter, research approaches related to this thesis will be discussed, which have not been mentioned in the previous chapters. The following sections encompass:

- Web service search on the public Internet as opposed to an intra-enterprise setting,
- semantic Web service standards and selected research on semantic Web services,
- relational Web service models and their application to research problems,
- software component retrieval and tool support,
- other approaches to detect matching Web services.

5.1 Web Catalogs and Search Engines on the Web

As Web services are built upon the infrastructure of the Internet it is close at hand to use techniques that have been developed for Internet search to find Web services. Besides UDDI (see Section 2.2), Web catalogs and search engines are typical examples of such techniques. Surveys on Web service catalogs and the use of general purpose search engines have been conducted by [FK05, BSFT06, HLV07]. The main results are summarized briefly.

[FK05] come to the conclusion that discovery of Web services through public catalogs is inhibited because developers who register their services do not make enough use of the available description tags. Many registry entries are described by less than five words and the chosen terms are very general. The survey shows that service description is a manual task that does not seem to be embraced by developers but it is the only vehicle to transport information for the human user searching for a service in a Web catalog.

[BSFT06] focus on the search functionality, the size of the Web catalog and additional features that go beyond the central search functionality of the UDDI search API. The survey shows that this search functionality can be integrated into Web catalogs with additional offerings that help the user to make a decision for or against the service. A detailed, human readable service description is the most important feature. Additional information such as user ratings, user comments and price information are also considered to be a desirable surplus. General purpose search engines are regarded as an alternative to Web catalogs that has to be considered as well. [BSFT06] see a promising development in Web service search support through Web catalogs and search engines.

[HLV07] show that Web catalogs are not growing any longer. Web catalogs have not advanced since [BSFT06]. First of all, the offer of a catalog must be attractive. Most public catalogs offer very common functionality, such as unit conversion services. Second, the offer must be described carefully. Finally, the Web catalog provider must take care to keep available services up to date. Many entries in public Web service catalogs are invalid and their descriptions make a derelict impression. General purpose search engines have not improved the support of Web service search, either. For example, Google returns far too many irrelevant links when searching for WSDL files. Search for operations is not possible.

An alternative to Web service catalogs and general purpose search engines are search engines, specialized on Web services. A research prototype called Woogle[1] has been implemented to augment keyword based search with template search and composition search as documented in [DHM+04]. The user specifies a Web service template by defining the names of the desired functionality as well as the names of the input and output parameters. This template is then used to search for Web service operations with similar names of operations, input, and output parameters. Further, this algorithm can also be used to search for Web services that are suitable to be composed with a given service. This search functionality treats composition as concatenation of services. The search returns Web services whose input parameters match the output parameters of the search template or vice versa. In this respect, the search functionality is comparable to the sequence suggestions examined in Section 4.3.3.

[1]http://data.cs.washington.edu/webService/

The underlying algorithm of Woogle is a clustering algorithm. As similarity measure *Term Frequency/Inverse Document Frequency* (TF/IDF) [SB88] is used. To compute the similarity between two Web services the similarity of their operations and of their input and output parameters is computed. The computed similarities are combined linearly according to chosen weights.

This approach relies on the same kind of information as the relational matching approach proposed in Section 3.3 except for domains which are not explicitly taken into account by Woogle. Instead of a semantic enhancement as used in Section 4.3, the approach relies on clustering and is therefore implicitly based on the assumption of redundancy. In an Internet setting, redundancy of services is a result of competition for customers. In an Intranet setting, a Web service repository is used to avoid redundant software development. Therefore, the conditions of internal search are not adequate for a clustering approach.

5.2 Semantic Web Services

Different approaches towards semantic Web services can be distinguished in the research literature. Many of them are based on the Web Ontology Language (OWL) or its predecessors. Therefore, OWL and the underlying frameworks RDF and RDFS are presented briefly. They can also be used as a technical implementation vehicle for semantic annotations of Web service operations as presented in Chapter 4.

Three research directions, semantic annotations for WSDL, OWL-S, and WSMF are summarized succinctly to show divergent approaches that make use of the afore presented underlying techniques. Each approach reveals its own attitude towards the idea of a semantic Web service.

Resource Description Framework (RDF)

RDF is a general purpose XML-based description framework to represent information about resources on the Web, in particular meta-data about Web resources, e.g., a homepage or its owner. RDF can also be used for information about things that are not network-accessible such as people, companies, or abstract concepts as long as they can be identified on the Web. RDF is serializable in XML. A complete documentation of RDF is given in [Wor04d, Wor04g, Wor04j, Wor04e].

Resources posses properties and RDF specifies these properties in simple subject-predicate-object statements, e.g., "`http://www.example.org/index.html` has creator whose value is John Smith". In this sentence the subject is

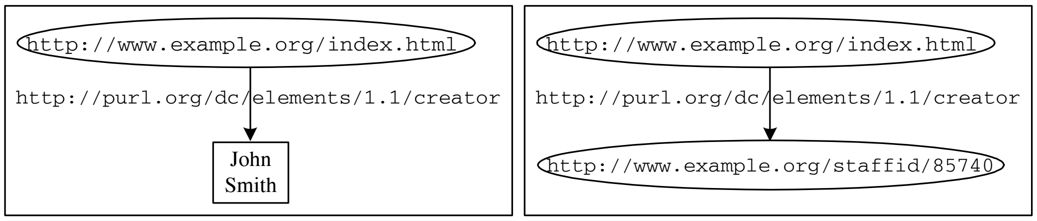

http://www.example.org/index.html
http://purl.org/dc/elements/1.1/creator
John Smith
http://www.example.org/index.html
http://purl.org/dc/elements/1.1/creator
http://www.example.org/staffid/85740

Figure 5.1: RDF graph according to [Wor04d].

the homepage, the predicate is "has creator" and the object is John Smith. This can be graphically depicted as shown on the left-hand side in Figure 5.1. The Web page in this example is identified by its URL. The abstract concept of a creator is represented using a *URI reference*[2] (URIRef) which is a URI, optionally followed by a fragment identifier. The URI `http://purl.org/dc/elements/1.1/creator` is an element of a pre-defined RDF vocabulary, the Dublin Core Meta-Data[3] (`dc`).

The URIrefs in RDF play the role of a *vocabulary* and help to uniquely identify a resource. Although RDF offers a standardized way to make statements about resources it does not specify the meaning of the URIs used. If the `dc:creator` URI is used, an application or a programmer familiar with the Dublin Core Meta-Data knows the meaning of it. For applications or programmers unfamiliar with this vocabulary, the string does not mean anything and RDF does not help in understanding the meaning. The means to define a vocabulary are provided by RDF Schema (RDFS).

Resource Description Framework Schema (RDFS)

RDFS is documented in [Wor04f]. In the example above, the existence of a class `homepage` that has the property `creator` needs to be defined for a vocabulary about homepages. The object of the `creator` property must be an instance of a class called `person`. This can be formalized as RDF statement as shown in Figure 5.2.

As the schema definition is an RDF statement it is serializable as an XML document. RDFS allows to define classes, subclasses, properties, and subproperties. Thus, RDFS provides the means to define domain-specific vocabularies, but it does not provide them. Well-known examples of RDFS vocabularies besides Dublin Core are RSS [4] or the Gene Ontology[5]. RDFS is a lim-

[2]http://ftp.ics.uci.edu/pub/ietf/uri/
[3]http://dublincore.org/
[4]http://www.rssboard.org/rss-specification
[5]http://www.geneontology.org/

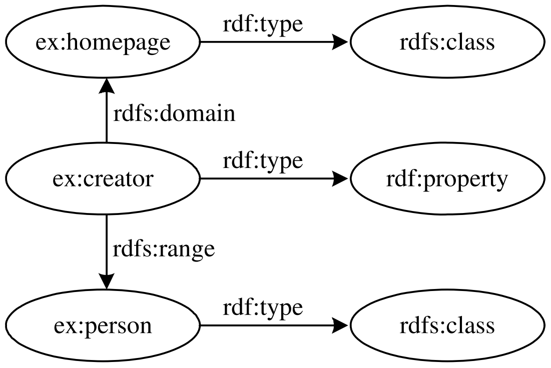


Figure 5.2: RDF schema as graph.

ited language because it is not able to express cardinality constraints, range restrictions depending on classes, properties of properties such as transitivity or disjointness. To express such constraints a richer language is needed. The Web Ontology Language (OWL) is an example of such a language and introduced next.

Web Ontology Language (OWL)

The Web Ontology Language (OWL) is based on RDF and RDFS adding more expressions to describe classes and properties. The language is designed to model machine processable ontologies and is fully documented in [Wor04b, Wor04a, BvHH^{+}04, Wor04c]. OWL contains three increasingly expressive sub-languages:

OWL Lite allows to express simple constraints and a classification hierarchy. It is intended as an easy migration path for thesauri and taxonomies. Its vocabulary is reduced compared to OWL DL. Cardinality constraints are restricted to 0 and 1. OWL Lite has a lower complexity than OWL DL.

OWL DL provides the same language constructs as OWL Full but with some constraints that guarantee that OWL DL remains decidable, e.g., a class cannot be an instance and a subclass of another class. OWL DL corresponds to description logics in its complexity.

OWL Full provides the full language support for OWL giving up decidability. OWL Full constructs can augment the pre-defined RDF and OWL vocabulary.

The formal semantics of OWL is defined in [Wor04c]. The language constructs permit to make logical inferences. For example:

- Derived classification: If an instance belongs to a class A and A is a subclass of class B then the instance also belongs to class B.
- Derived identity: If a property is declared to be functional and an instance has two different range values assigned for it, the two values must have the same meaning, i.e., denote the same instance.
- Derived equivalence: If class A is equivalent to class B and class B is equivalent to class C then A is equivalent to C.

The description logic DL Lite, presented as example in Chapter 4, is expressible with OWL Lite [CGL$^+$04]. The terminology used to describe the financial taxonomy can also be technically implemented as OWL Lite ontology. Therefore, the approach suggested in Chapter 4 can be realized with the help of existing standards without adding an additional layer to the Web service standard stack.

An alternative approach, which also avoids the creation of a new Web service standard, starts from the existing Web service standards UDDI and WSDL and adds semantics within this framework. This direction of research is presented next.

Semantic Annotations for WSDL

Semantic Annotations for WSDL (SAWSDL) is a current W3C candidate recommendation [Wor07a] that has emerged from a member submission called WSDL-S by the LSDIS lab[6] and IBM research. Basically, WSDL files are annotated with references to ontological concepts. The annotated WSDL file can further be turned into a UDDI entry making use of the implicit support of ontologies in UDDI via `tModels`. The discovery of semantically annotated Web services is based on these extended UDDI entries. This approach is compatible with current Web service standards. SAWSDL is based on WSDL 2.0, but an extension for WSDL 1.1 is also defined. The following presentation is based on WSDL 2.0.

SAWSDL defines two types of extension attributes as shown in Figure 5.3:

modelReference is an extension attribute that defines a connection between a WSDL interface, an operation, XML Schema complex type

[6]http://lsdis.cs.uga.edu/

definitions, simple type definitions, element declarations, attribute declarations and a concept in some semantic model. This can be an RDF-S vocabulary, an OWL ontology, or other kinds of semantic conceptualizations. Applications that understand the vocabulary may use this additional information for service detection.

liftingSchemaMapping, loweringSchemaMapping are attribute extensions that define a mapping between XML Schema element declarations, complex type definitions, simple type definitions, and semantic data. These expressions can be in any language such as SparQL [Wor07b], XSLT [Wor99], or XQuery [Wor07f]. They are used after service detection for mediation purposes.

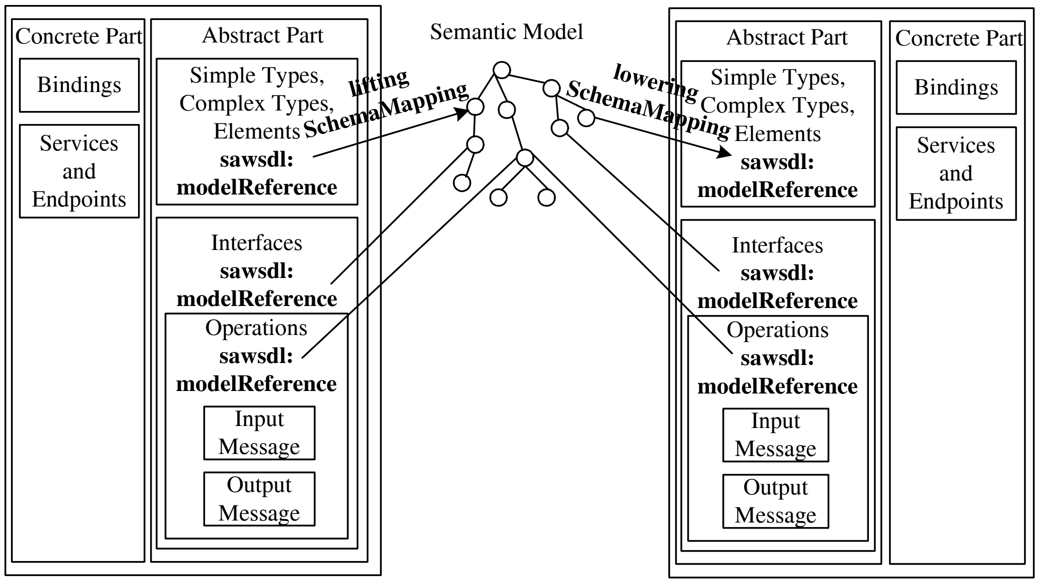


Figure 5.3: SAWSDL overview.

As SAWSDL does not demand the usage of any specific semantic model `loweringSchemaMappings` and `liftingSchemaMappings` depend on the chosen model. If a semantic model based on RDF is used, `liftingSchemaMappings` can be defined using XSLT or XQuery. The inverse `loweringSchemaMappings` is more complex as the XML serialization of RDF triples is not unique. Several equivalent XML representations of an RDF model exist. First, RDF must be pre-processed using a query language such as SparQL to produce an XML table of variable bindings. Then an XSLT transformation can be applied to produce the needed XML data format.

[Ver06] has used SAWSDL to create executable processes from given abstract process specifications and to adapt the process during execution in case of a technical failure. The approach has been applied to a supply chain scenario for computer hardware. The impact of the expressiveness of the ontology for the complexity of the search by semantic annotations has not been studied so far. In this respect, this thesis makes a contribution to the SAWSDL approach.

Other approaches to give semantics to Web services take a different turn. They define an additional layer of semantic Web service standards that are mapped to UDDI and WSDL, or can be used to generate WSDL files and UDDI entries. Two representatives are OWL-S and the Web Service Modeling Framework (WSMF).

Web Ontology Language for Services (OWL-S)

The semantic Web language OWL DL has been used by the OWL-S coalition of the DARPA Agent Markup Language (DAML) Program[7] to create a machine understandable description of services based on an OWL service ontology. This description is intended to support service discovery, invocation, composition, and interoperation. The service ontology is designed as a standard language for describing services, consisting of a set of basic classes and properties. The current OWL-S 1.2 Pre-Release is documented in [MBH$^+$06].

OWL-S provides an abstract top-level ontology[8] for describing a service as an OWL class `Service` that is described by a class called `ServiceModel`, presented by a class called `ServiceProfile` and supported by a class called `ServiceGrounding`. This top-level ontology is graphically depicted in Figure 5.4. The `ServiceModel` provides a process description of how the service works. The `ServiceGrounding` describes how to use a service. The `ServiceProfile` contains information to support service discovery.

Grounding a service using WSDL relies on the following three correspondences which are summarized in Figure 5.5:

1. An atomic process of a ServiceModel corresponds to a WSDL operation.

2. The input parameters of an atomic process correspond to an input message of a WSDL operation. The output parameter correspond to an output message respectively.

3. OWL-S classes as DL-based types are regarded as abstract types that can be used in WSDL messages.

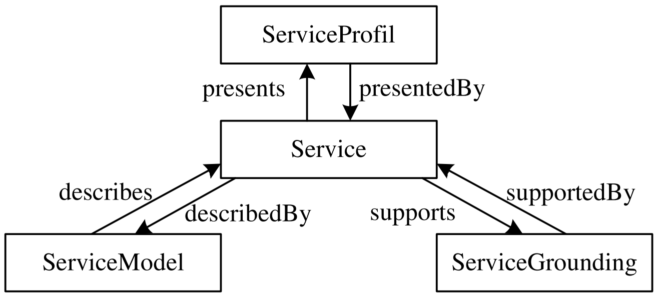


Figure 5.4: OWL-S service ontology.

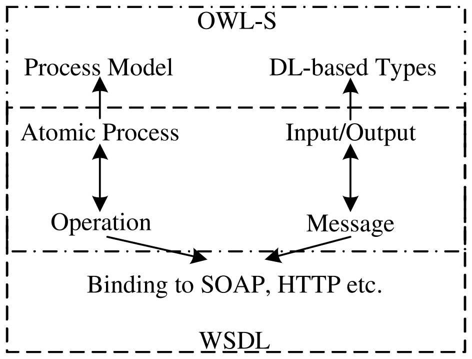


Figure 5.5: Overview of OWL-S GroundingProfile according to [MBH+06].

The OWL-S ServiceProfile aims at creating a service registry for service detection. Thus, its goal is similar to UDDI. In contrast to UDDI the ServiceProfile does not specify registry APIs, only registry entries in the form of ServiceProfiles. The businessEntity element of a UDDI entry contains contact information similar to the ServiceProfile contactInformation property. A reference to a tModel allows to retrieve the service interface information from a WSDL file which contains the input and output parameters of the service operations. This information is directly contained in a ServiceProfile. It permits to express rules as pre- and postconditions. Therefore, it is semantically richer than a UDDI entry. From the side of the semantic Web service community proposals have been made to map a Service Profile to a UDDI entry, thereby deriving the UDDI entry from the ServiceProfile [MBL+03]. Only recently, the W3C working group for WSDL has published a note on the inverse mapping from WSDL to RDF [Wor07e].

The OWL-S ontology languages is a representative of the approach to describe Web service capabilities using semantic Web techniques as an additional layer on top of or as replacement for Web service standards. OWL-S has been used in a number of research approaches on Web service description, composition, and matching, e.g. [LH03, MBE03, BHL+05, KFS06, SPS06].

Another standard that also adds an additional standard layer to the Web service standard stack is presented in the following paragraph.

Web Service Modeling Framework

The Web Service Modeling Framework (WSMF) [FB02] is a conceptual modeling framework for the development and description of Web services, especially in e-commerce, that tries to combine the principle of loose coupling of services with a strong mediation to enable cooperation among Web services. The framework consists of four main components: ontologies as common terminology, goal repositories to describe problems to be solved by Web services, Web service descriptions, and mediators to overcome interoperability obstacles. The Web Service Modeling Ontology (WSMO) [LdBKL06] is the conceptual basis of the WSMF and refines the four core artifacts *ontologies*, *goals*, *Web services* and *mediators*. The ontologies of the WSMO provide formal and explicit specifications of the vocabularies which are used to describe goals, Web services and mediators.

- Ontologies in WSML consist of concepts forming the basic terminology

[7]http://www.daml.org/services/owl-s/

[8]http://www.ai.sri.com/daml/services/owl-s/1.2/Service.owl

with different attributes, relations between concepts, instances of concepts, relation instances and axioms to refine concepts and relations.

- Web services are described from the provider perspective by their capabilities and their interfaces. The capabilities are expressed as preconditions that hold in the information space before service execution and postconditions which describe the state after execution. The pre- and postconditions can be formulated using a WSML ontology. Therefore, goals are also expressed in terms of capabilities and interfaces.

- Mediators are designed to bridge differences in representation formats, encoding styles, and business protocols between Web services, goals and ontologies.

The WSMO is described in a specialized ontology language called Web Service Modeling Language (WSML). There exist five different variants that vary in expressiveness, summarized according to [dBLPF06]:

- WSML Core: Intersection of Horn Logic [Gup98] and Description Logic,

- WSML DL: Description Logic,

- WSML Flight: Rule language based on a logic programming variant of F-Logic [AL04],

- WSML Rule extension of WSML Flight,

- WSML Full: First order unification of WSML DL and WSML Rule.

A concise overview is given in, e.g., [RdBM$^+$06]. Both, WSMO and WSML, have been submitted to the W3C [dBBD$^+$05, dBFK$^+$05] for consideration. The research initiative behind this submission is primarily lead by DERI[9] and also lays the foundation for their current research effort in Semantically Enabled Service-oriented Architectures (SESA) [ABdB$^+$06].

The WSMO approach has been applied to e-business [dBFKL05], especially an investment banking scenario in [Lar06, KLL$^+$05, LF05], searching for Web services that supply information about investment funds from different European countries in a two-step approach. First, the number of candidate services is restricted based on the output description of the service. Then, the result is refined based on a semantic description of the input.

Semantic Web approaches are also used to describe quality of service properties and to use semantic descriptions of non-functional service aspects

[9]http://www.deri.org/

for Web service matching, e.g., in [DOH$^+$01, OEtH02, OEtH05] a semantic framework for non-functional Web service description is presented that models, e.g., temporal and spatial availability, communication channels, payment, pricing, security, and reputation. This description is not limited to Web services, but is extended to services in general.

5.3 Relational Web Service Models

Relational models of Web services have been used to study Web service composition and properties of Web service executions.

A relational view on Web services has been taken in [LN04] to study the execution of Web service compositions. In this approach, a Web service operation is regarded as a function with n input and m output arguments. It is represented as a relation with access pattern. The access pattern defines if an attribute of the given relation is an input or output argument of the Web service operation. A conjunctive query is regarded as a declarative description of a Web service composition. The central question in this setting is, if the composition is feasible, i.e., if the conjunctive query can be executed observing the given access patterns of the involved relations disregarding the specific syntactic form of the query.

In general, it is possible to formulate conjunctive queries with limited access patterns which are not executable while observing the given syntactic order. However, if it is possible to compute an equivalent conjunctive query that is executable, the original query is feasible. Deciding feasibility is NP-complete [Li03] and a variation of the query containment problem [AHV95]. In [NL04], approximative algorithms for containment checking are presented, which avoid worst case complexity for unions of conjunctive queries with negation. The executable conjunctive query represents a variation of the original Web service composition plan.

A relational Web service representation is also used in [DSVZ06] to verify Web service compositions. A Web service is modeled as a set of relations consisting of an underlying database, a state relation, an input relation, an action relation, and two relations representing input and output queues. In this model, Web services communicate asynchronously by messages exchange. Human interaction with the Web services is also possible. Rules are used to express which messages are sent and received and into which state the Web service changes. The internal state, the internally available information and the content of the exchanged messages are modeled explicitly. This representation abandons the idea of a Web service as a black box.

The model is used to verify properties of Web service compositions expressed as first order temporal logic properties and as conversation protocols [DSV07]. Different sub-cases of bounded and unbounded input and output queues as well as lossy and lossless channels are distinguished. This Web service model is also used to verify properties of runs over sequences of Web pages with human interaction [DSV04].

In earlier work [AVFY00], Web services are represented as relational transducers for electronic commerce [Spi03], assuming a finite number of internal states and describing state changes as well as output messages as datalog rules. The exchanged messages are saved in a log relation. This model is used to verify runs of programs based on their log, to verify temporal properties and to decide goal reachability. In contrast to the work in [DSV07], the number of states in [AVFY00] is finite.

5.4 Software Component Retrieval

The Web service search problem can be regarded as a variation of the earlier research subject of software component retrieval for software libraries. Here, two aspects of this line of research are presented by example of selected publications:

- signature and specification matching, which is related to the approach presented in this thesis and
- search by refinement with a quantitative match classification, which is a complementary approach to this thesis.

In software component retrieval, often a two-step search strategy is applied. First, the signature is matched, which describes the operations with their input and output parameters. Then, the specification is matched, which describes the intended behavior of the component. Prominent examples are found in [MZW93, MZW95, MZW97]. In this work, the signature of a software component is described by its ML[10] interface. It consists of one or more functions in combination with the input and output data types. Different degrees of matching signatures, exact, generalized, and specialized are examined in [MZW95]. An exact signature match between two functions requires at most renaming of parameters. For a generalized match, the library component has more general types than the user request signature. A specialized match is the inverse of a generalized match.

The specification of a software component is described by pre- and post-conditions that need to be fulfilled before, respectively after the execution

[10]ML interfaces are similar to Modula-3 interface modules.

of the component. The pre- and postconditions are described using first-order logic in [MZW97]. Match or mismatch is then decided using a theorem prover, distinguishing different match degrees. An exact specification match presumes a matching signature. The pre- and postconditions of the two components are equivalent. For a plug-in match, the precondition of the first component implies the precondition of the second component and vice versa for the postcondition. For a generalized match, the conditions of the second component are more general and imply the conditions of the first component. A specialized match is the inverse of a generalized match.

In this respect, the classification hierarchy of [MZW97] is similar to the different match types presented in this thesis. However, an operational description of the match is not added. Sequence suggestions based on the different match types are not considered by [MZW97]. Further, partial matches are not taken into account. Finally, the approach is based on an extensive specification of components, comparable to the formal specification of abstract data types. The approach has been applied to the specification of basic data types such as stack, queue, or list. For Web services, such extensive descriptions are not necessarily available.

A coupled approach of signature and specification matching has also been applied to the Web service search problem in [KKR04]. Here, a state-oriented service description is used for description of service request and service offer. This approach also accepts partial matches, comparable to the partial matches defined in Section 3.4, which do not have a completely matching signature.

Other research approaches do not categorize a match based on a qualitative notion, as has been done in this thesis, but they focus on computing a quantitative distance between software components and computing an approximation by refinement if an exact match is not available. An example of this line of research, which also takes a relational approach towards specification matching, is found in [MMM94]. A specification of a software component is represented as a relation of all accepted pairs of input and output parameters. Two specifications are compared by comparing their relational representations. A specification is "more-defined" than another specification if the underlying relation has a larger domain.

Based on this criterion of comparison, a partial ordering of specifications is constructed. The more instances two relational specifications have in common the closer they match each other. Computing a (partial) match to a given specification is turned into the search of a minimal relation that maximizes the set of common elements between itself and the relation of the component sought. This approach is elaborated further in [LJDF$^+$97]. The approach has been applied to compute the distance between different Pascal compilers.

5.5 Other Web Service Matching Approaches

Other Web service matching approaches can be distinguished by:

- the model they use to represent a Web service,
- the use case for searching, design-time search or run-time search, and
- the intended usage of the match, e.g., for service composition, for workflow planning, or for quality of service negotiations.

An overview of different automata-based and model theoretic approaches towards Web service modeling, match computation and composition is given in [Dra01, NM02, HBCS03, HS04, HS05, Hul05]. Other approaches to Web service matching are based on different concepts that do not fall into one of the categories mentioned above, e.g., contract-based [ELS05], graph-based [HCL04], or model-driven [HKC05], to name just a few examples.

Finally, the support offered by case tools needs to be mentioned. Case tools are often designed to support a specific modeling language or design methodology for software engineering. For example, the conceptual, logical, and physical design of relational databases is supported by Sybase PowerDesigner[11]. It allows the graphical design of ER-diagrams with a stepwise refinement down to code generation for different database systems. The UML notation is supported by case tools such as IBM Rational Rose[12] or Borland Together, to name just a few.

So far, a specific service-oriented design methodology with a tailored case tool support has not emerged. Different suggestions are reported, for example, domain-oriented design (e.g., [HVH06]), pattern-based approaches (e.g., [GAA$^+$06]) or UML-variations (e.g., [SB05, MdCV03]). Nevertheless, software vendors support Web service standards in their tool suites. At least, the automatic generation of a WSDL file from existing code is possible.

An example to increase reuse on a meta-level is the reusable asset browser in the IBM Rational Professional Bundle [Ger05]. To facilitate SOA design, it supports the OMG standard RAS 2.2 for the description of reusable assets [Obj05]. Reusable assets are artifacts that provide a solution to a given problem in a given context. They describe design patterns, models, or simply "recipes" that describe how to combine different assets. Reusable assets can

[11]http://www.powerdesigner.de/

[12]http://www-306.ibm.com/software/de/rational/design.html

be stored in an enterprise wide repository. Search is supported through an hierarchical asset browser.

Chapter 6

Conclusions and Perspectives

Finding Web services within an enterprise-wide development environment is a crucial step in the development process of service-oriented applications to increase service reuse and to avoid proliferation of services in the service-oriented architecture.

6.1 Summary of Results

The starting point for this thesis has been the observation that current Web service standards are not seamlessly integrated with service-oriented software development. Standards such as UDDI are technical standards that have been created with B2B service brokerage for e-commerce as major application scenario. The central aim of standards such as UDDI is that many service providers offer their Web services globally through central registries. These registries may contain many versions of equivalent Web services to give the consumers a choice. Redundancy as a consequence of competition is wanted and not avoided.

This is in contrast to the motivation of enterprises that invest in service-oriented applications. They want to save costs by reducing development and maintenance of redundant functionality, as market research shows. They want to avoid the duplication of functionality that is caused by heterogeneous environments. Therefore, detecting duplicates when software programs are already implemented is too late. As a consequence, a central repository of Web services within an enterprise must, first of all, support the detection of available functionality at development time. Such a repository can be built up step by step while developing the service-oriented applications or added afterwards.

During the case study, the logical design step of a structured development process has been identified as the step that benefits most from a central Web service operation repository at the level of operations. The major concern of such a repository is the internal administration of interface information. Therefore, a relational database approach has been chosen. The information of the WSDL files is turned into the content of the relational Web service operation repository. This avoids the creation of a new technical standard but reuses the existing standard approach.

The relational representation in the repository is used to express the Web service search problem as a relational schema matching problem. A syntactic search approach based on relational schema matching algorithms has been suggested. The analysis of this purely syntactic approach has revealed that it is possible to adapt existing match algorithms to the problem of Web service operation matching. This has been shown using two classical match algorithms in combination. The match result is described as an extended relational algebra expression. This is regarded as the description of a transformation Web service which turns the messages of one Web service into the format of the other. Thus, the match operation executed on two relational Web service schema descriptions returns a relational Web service schema as result. The matching expression has served to define a hierarchy of matches. The approach works self-contained on the available interface information and does not need any additional meta-information. This relieves the developer of documentation overhead. As the approach is purely syntactic, it relies on naming conventions for operations, attribute, and domains.

Therefore, the repository has been further enhanced with semantic annotations at the level of operation and attribute names. The case study from financial industry has shown that the necessary taxonomies often exist, e.g., in terms of data warehouse dimensions, reporting categories, or product catalogs. The semantic annotations must be added at Web service development time and need maintenance. Once added, they provide a richer search functionality than purely syntactic matching. An efficient search algorithm has been presented that exploits the semantic annotations and the taxonomy to compute matches of different degrees. The match computation takes semantic dependencies into account and supports the automatic generation of alternatives. The operation retrieval is expressed as a semantic extension of SQL. This approach combines Web service standards with semantic Web standards. Further, the taxonomy can be used to increase the quality of the technical documentation and definition of suitable criteria for encapsulation of Web services.

The case study from financial industry has shown that functionality is especially suited to be offered as a Web service if:

- it encapsulates central, non-trivial business logic,
- the business logic is needed in more than one application context, e.g., operational and back-office systems,
- consistency of results is needed across different applications contexts, possibly with different stakeholders.

In the ongoing discussion about a service-oriented development process, this thesis closes a gap in the logical design step of service-oriented architecture. The relational operation repository with semantic annotations does not only provide structured algorithmic support for Web service operation search from the point of view of the developer, but also documentation related to the development of the service landscape.

6.2 Outlook on Research Perspectives

The semantic syntactic search support of the relational Web service repository is intended for search at design time. The search takes place at the logical design level. Run time search support is a different research direction with different needs, e.g., service redundancy is wanted in this setting to compensate for technical failure of individual services. The search takes place on the physical level of the service-oriented application. Still, similarities between the two search scenarios exist. Search at run time must ultimately generate a Web service operation call which might include the transformation of input and output messages to make them fit the WSDL service interface. Such transformation instructions could be computed before-hand and stored in an extension to the relational Web service registry that contains syntactic matches and relational match expressions.

This thesis has shown that it is possible to create search support for service-oriented development and design without adding new standards to the already existing standards for Web services. Other aspects of the service-oriented software life-cycle such as the test and roll-out phase or the maintenance and governance phase can also benefit from structured meta-information support as presented in this thesis in terms of semantic annotations. They may also be used, e.g., for planning the next development projects. These development aspects are still regarded as a subject for software consulting.

Finally, the semantic extension of the relational Web service operation repository relies on semantic annotations. The developer must make additional effort to add semantic annotations to the implemented Web service, and there is additional effort involved in maintaining the taxonomies. Annotations, or tags, are also a popular phenomenon of Web 2.0. There, tags help users to structure their own digital information, e.g., photo collections. In Web 2.0 communities, the acceptance of common tags and their meaning is based on public consensus. Every user creates his or her own tags. Popular tags are reused by others. This is different in an enterprise setting where the meaning of software components must be clearly defined. In Web 2.0, tagging is accepted and popular, whereas in software-development additional documentation is often dreaded by developers. This rises the questions how incentives and experience of the social Web 2.0 can be used to enhance semantic annotation of Web services and to support a comfortable annotation process for developers.

Bibliography

[ABdB$^+$06] D. Anicic, M. Brodie, J. de Bruijn, D. Fensel, T. Haselwanter, M. Hepp, S. Heymans, J. Hoffmann, M. Kerrigan, J. Kopecky, R. Krummenacher, H. Lausen, A. Mocan, and J. Scicluna. A Research Roadmap for DERI Innsbruck. Technical Report deri-tr-2006-09-15, DERI, 9 2006. URL: http://www.deri.at/fileadmin/documents/deri-tr-2006-09-15.pdf (2007-10-01).

[ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications.* Springer-Verlag, Berlin, Germany, 2004.

[ACM03] D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns. Best Practicies and Design.* Prentice Hall, Upper Saddle River, NJ, USA, 2003.

[ADMR05] D. Aumueller, H. Hai Do, S. Massmann, and E. Rahm. Schema and Ontology Matching With COMA++. In *Proc. of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, 14-16 June 2005*, pages 906–908, 2005.

[AGH05] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language.* Addison-Wesley, Reading, MA, USA, 2005.

[AH05] A. Arsanjani and K. Holley. *Increase Flexibility with the Service Integration Maturity Model (SIMM).* IBM Corporation, 2005. URL: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-simm/ (2007-10-01).

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesly, Reading, MA, USA, 1995.

[AL04] J. Angele and G. Lausen. Ontologies in F-logic. In Staab and Studer [SS04], pages 29–50.

[Ars04] A. Arsanjani. Service-Oriented Modeling and Architecture. How to Identify, Specify, and Realize Services for Your SOA. Technical report, IBM, 2004. URL: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/ (2007-10-01).

[Ars05] A. Arsanjani. Toward a Pattern Language for Service-Oriented Architecture and Integration. Technical report, IBM, 2005. URL: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-soi/ (2007-10-01).

[AVFY00] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational Transducers for Electronic Commerce. *Journal of Computer and System Sciences*, 61:236–269, 2000.

[Bal00] B. Balzert. *Lehrbuch der Software-Technik*, volume 1. Spektrum Akademischer Verlag, Heidelberg, Germany, 2 edition, 2000.

[Bar06] J. Barnes. *Programming in ADA 2005*. Addison-Wesley, Reading, MA, USA, 2006.

[BBC$^+$99] P. A. Bernstei, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft Repository Version 2 and the Open Information Model. *Information Systems*, 24(2):71–98, 1999.

[BBHN05] R. Breu, M. Breu, M. Hafner, and A. Nowak. Web Service Engineering - Advancing a New Software Engineering Discipline. In *Proc. of the 5th International Conference on Web Engineering (ICWE), Sydney, Australia, 27-29 July 2005*, volume 3579 of *Lecture Notes in Computer Science*, pages 8–18, 2005.

[BBZ$^+$05] M. H. Burstein, C. Bussler, M. Zaremba, T. W. Finin, M. N. Huhns, M. Paolucci, A. P. Sheth, and S. K. Williams. A Semantic Web Services Architecture. *IEEE Internet Computing*, 9(5):72–81, 2005.

[BEA03] BEA Systems Inc. *Customer Case HewlettPackard*, 2003. URL: http://h71028.www7.hp.com/enterprise/downloads/HP%20IT%20and%20Workshop.pdf(2007-10-01).

[BEA06] BEA Systems Inc. *Customer Case Study Schiphol Airport*, 2006. URL: http://www.bea.com/content/news_events/white_papers/BEA_Schiphol_Airport_cs.pdf(2007-10-01).

[Bel06] J. Belger. Die Service Oriented Platform der Deutschen Post. *Java Spektrum*, (1):22–25, 2006.

[Ber96] P. A. Bernstein. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, 1996.

[BH06] C. Baroudi and F. Halper. Executive Survey: SOA Implementation Satisfaction. Technical report, Hurwitz and Associates, 2006. URL: http://www2.mindreef.com/hurwitz.aspx?cid=aebc8bb4-8a15-48fa-94f3-473f9fc4c96c (2007-10-01).

[BHB05] J. Becker, S. Hallek, and C. Brelage. Fachkonzeptionelle Spezifikation konfigurierbarer Geschäftsprozesse auf Basis von Web Services. Technical Report WWU-WI-113, Arbeitsberichte des Instituts für Wirtschaftsinformatik, Westfälische Wilhelms-Universität Münster, 2005.

[BHL$^+$05] B. Benatallah, M. S. Hacid, A. Léger, C. Rey, and F. Toumani. On Automating Web Services Discovery. *VLDB Journal*, 14(1):84–96, 2005.

[BKK$^+$05] J. Beatty, G. Kakivaya, D. Kemp, T. Kuehnel, B. Lovering, B. Roe, C. S. John, J. Schlimmer, G. Simonnet, D. Walter, J. Weast, Y. Yarmosh, and P. Yendluri. *Web Services Dynamic Discovery (WS-Discovery)*. Microsoft Corporation Inc., 2005. URL: http://schemas.xmlsoap.org/ws/2004/10/discovery/ws-discovery.pdf (2007-10-01).

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 2001(May issue), 2001.

[BN03] F. Baader and W. Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.

[BP98] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.

[BR00] P. A. Bernstein and E. Rahm. Data Warehouse Scenarios for Model Management. In *Conceptual Modeling, Proc. of the 19th International Conference on Conceptual Modeling (ER), Salt Lake City, Utah, USA, 9-12 October 2000*, volume 1920 of *Lecture Notes in Computer Science*, pages 1–15, 2000.

[Bri05a] British Standards Institution. *Structured vocabularies for information retrieval. Guide. Definitions, symbols and abbreviations*, 2005.

[Bri05b] British Standards Institution. *Structured vocabularies for information retrieval. Guide. Thesauri*, 2005.

[BSFT06] D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma. Web Service Discovery - A Reality Check. Technical Report DERI-TR-2006-01-17, DERI, 1 2006. URL: http://www.deri.at/fileadmin/documents/DERI-TR-2006-01-17.pdf (2007-10-01).

[BvHH$^+$04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference. W3C Recommendation*. World Wide Web Consortium, 2004. URL: http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (2007-10-01).

[CCMW01] E. Christensen, F.o Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) Version 1.1. W3C Note*. World Wide Web Consortium, March 2001. URL: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (2007-10-01).

[CGL$^+$04] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, and G. Vetere. DL-Lite: Practical Reasoning for Rich Dls. In *Proc. of the 2004 International Workshop on Description Logics (DL), Whistler, British Columbia, Canada, 6-8 June 2004*, volume 104 of *CEUR Workshop Proceedings*, 2004.

[CJ02] D. Chappell and T. Jewell. *Java Web Services.* OReilly, Cambridge, MA, USA, 2002.

[Cod70] E. F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):2, 1970.

[dBBD+05] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. Knig-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. *Web Service Modeling Ontology. W3C Member Submission.* World Wide Web Consortium, 2005. URL: http://www.w3.org/Submission/WSMO/ (2007-10-01).

[dBFK+05] J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. *Web Service Modeling Language. W3C Member Submission.* World Wide Web Consortium, 2005. URL: http://www.w3.org/Submission/WSML/ (2007-10-01).

[dBFKL05] J. de Bruijn, D. Fensel, U. Keller, and R. Lara. Using the Web Service Modeling Ontology to Enable Semantic e-Business. *Communications of the ACM*, 48(12):43–47, 2005.

[dBLPF06] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service Modeling Language WSML: An Overview. In *The Semantic Web: Research and Applications, Proc. of the 3rd European Semantic Web Conference (ESWC), Budva, Montenegro, 11-14 June 2006*, volume 4011 of *Lecture Notes in Computer Science*, pages 590–604, 2006.

[Dev06] B. Devlin. Opening the Door to a Service-Oriented Architecture. Technical report, IBM Dublin Software Laboratory, 2006. URL: ftp://ftp.lotus.com/pub/lotusweb/wplc/WPLC_SOA_white_paper.pdf (2007-10-01).

[DHM+04] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *(e)Proc. of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, 31 August - 3 September 2004*, pages 372–383, 2004.

[DJMZ05] W. Dostal, M. Jeckle, I. Melze, and B. Zengler. *Service-orientierte Architekturen mit Web Services. Konzepte-Standards-Praxis.* Spektrum Akademischer Verlag, Heidelberg, Germany, 2005.

[DMDH04] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy. Ontology Matching: A Machine Learning Approach. In Staab and Studer [SS04], pages 385–404.

[DOH$^+$01] M. Dumas, J. O'Sullivan, M. Hervizadeh, D. Edmond, and A. H. M. ter Hofstede. Towards A Semantic Framework for Service Description. In *Semantic Issues in E-Commerce Systems, IFIP TC2/WG2.6 Ninth Working Conference on Database Semantics, Hong Kong, 25-28 April 2001*, volume 239 of *IFIP Conference Proceedings*, pages 277–291, 2001.

[DR02] H. Hai Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. of 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, 20-23 August 2002,* , pages 610–621, 2002.

[Dra01] V. Draluk. Discovering Web Services: An Overview. In *Proc. of 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, 11-14 September 2001*, pages 637–640, 2001.

[DSV04] A. Deutsch, L. Sui, and V. Vianu. Specification and Verification of Data-driven Web Services. In *Proc. of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Paris, France, 14-16 June 2004*, pages 71–82, 2004.

[DSV07] A. Deutsch, L. Sui, and V. Vianu. Specification and Verification of Data-Driven Web Applications. *Journal of Computer and System Science*, 73(3):442–474, 2007.

[DSVZ06] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of Communicating Data-Driven Web Services. In *Proc. of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Chicago, Illinois, Maryland, USA, 26-28 June 2006*, pages 90–99, 2006.

[eJC06] The OASIS ebXML Joint Committee. The Framework for eBusiness. An OASIS White Paper. Technical report, OASIS, 2006. URL: http://www.oasis-open.org/committees/download.php/17817/ebxmljc-WhitePaper-wd-r02-en.pdf (2007-10-01).

[ELS05] G. Engels, M. Lohmann, and S. Sauer. Design by Contract zur semantischen Beschreibung von Web Services. In *INFORMATIK 2005 - Informatik LIVE! Band 2, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Bonn, 19. bis 22. September 2005*, volume 68 of *LNI*, pages 612–616, 2005.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[Fer03] R. Ferber. *Information Retrieval.* dpunkt.verlag, Heidelberg, Germany, 2003.

[FK05] J. Fan and S. Kambhampati. A Snapshot of Public Web Services. *SIGMOD Record*, 34(1):24–32, 2005.

[GAA$^+$06] J. Ganci, A. Acharya, J. Adams, P. Diaz de Eusebio, G. Rahi, D. Strachan, K. Utsumi, and M. Washio. *Patterns: SOA Foundation Service Creation Scenario.* IBM Redbooks, 2006.

[Gau05] W. Gaus. *Dokumentations- und Ordnungslehre. Theorie und Praxis des Information Retrieval.* Springer-Verlag, Berlin, Germany, 5 edition, 2005.

[Ger05] B. Gerson. Build and Test IT Applications with the IBM Rational Professional Bundle. Technical report, IBM Rational Software, 2005. URL: ftp://ftp.software.ibm.com/software/rational/web/white-papers/btia-probundle.pdf (2007-10-01).

[GGMO03] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WORDNET with DOLCE. *AI Magazine*, 24(3):13–24, 2003.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

[Gra01] S. Graham. *The Role of Private UDDI Nodes in Web Services, Part 1: Six Species of UDDI.* IBM, 2001. URL: http://www-128.ibm.com/developerworks/webservices/library/ws-rpu1.html (2007-10-01).

[Gru95] T. R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing? *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[GSY04] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: An Algorithm and an Implementation of Semantic Matching. In *The Semantic Web: Research and Applications, Proc. of the 1st European Semantic Web Symposium (ESWS), Heraklion, Crete, Greece, 10-12 May 2004*, pages 61–75, 2004.

[GSY05] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic Schema Matching. Technical Report DIT-05-014, University of Trento, Italy, 2005. URL: http://dit.unitn.it/ p2p/RelatedWork/Matching/CoopIS05-SSM-GSY.pdf (2007-10-01).

[Gua98] N. Guarino. Formal Ontology and Information Systems. In *Proc. of the 1st International Conference of Formal Ontology in Information Systems (FOIS), Trento, Italy, June 1998*, 1998.

[Gup98] G. Gupta. Horn Logic Denotations and Their Applications. In K. R. Apt, V. W. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm. A 25-Year Perspective*, pages 127–160. Springer-Verlag Berlin, Germany, 1998.

[HBCS03] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: a look behind the curtain. In *Proc. of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA, USA, 9-12 June 2003*, pages 1–14, 2003.

[HCL04] R. Heckel, A. Cherchago, and M. Lohmann. A Formal Approach to Service Specification and Matching Based on Graph Transformation. *Electronic Notes in Theoretical Computer Science*, 105:37–49, 2004.

[HGHHC05] T. Hong Gao, J. Huffman Hayes, and H. Cai. Integrating Biological Research through Web Services. *IEEE Computer*, 38(3):26–31, 2005.

[HHH$^+$05] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *Proc. of the ACM SIGMOD International Conference on*

Management of Data, Baltimore, Maryland, USA, 14-16 June 2005, pages 805–810, 2005.

[HKC05] Y. Huang, S. Kumaran, and J. Y. Chung. A Model-Driven Framework for Enterprise Service Management. *Information Systems and E-Business Management*, 3(2):201–217, 2005.

[HLV07] S. Hagemann, C. Letz, and G. Vossen. Web Service Discovery - Reality Check 2.0. Technical Report 5, ERCIS, Münster, Germany, July 2007.

[HM03] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. In *Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON) held at the 2nd International Semantic Web Conference (ISWC), Sundial Resort, Sanibel Island, Florida, USA, 20th October 2003 (Workshop day)*, volume 87 of *CEUR Workshop Proceedings*, 2003.

[HS04] R. Hull and J. Su. Tools for Design of Composite Web Services. In *Proc. of the ACM SIGMOD International Conference on Management of Data, Paris, France, 13-18 June 2004*, pages 958–961, 2004.

[HS05] R. Hull and J. Su. Tools for Composite Web Services: A Short Overview. *SIGMOD Record*, 34(2):86–95, 2005.

[HSHG06] Y. Huang, A. Slominski, C. Herath, and D. Gannon. WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing. In *Proc. of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Singapore, 16-19 May 2006*, pages 166–173, 2006.

[Hul05] R. Hull. Web Services Composition: A Story of Models, Automata, and Logics. In *Proc. of the IEEE International Conference on Web Services (ICWS), Orlando, FL, USA, 11-15 July 2005*, 2005.

[Hüs05] B. Hüsemann. *Entwurf und Realisierung von Ontologien für Multimedia-Anwendungen*. PhD thesis, Westfälische Wilhelms-Universität Münster, Germany, 2005.

[HVH06] B. Humm, M. Voss, and A. Hess. Regeln für serviceorientierte Architekturen hoher Qualität. *Informatik Spektrum*, 29(6):395–411, 2006.

[IDC04] IDC Marketing Services. *Toyota Australia Drives Out Cost and Increases Visibility with BEA Solution for Dealers*, 2004. URL: http://www.bea.com/content/news_events/white_papers/IDC_ToyotaAustralia_cs.pdf(2007-10-01).

[Kas06] P. Kastner. *Enterprise Service Bus and SOA Middleware. Benchmark Report.* Aberdeen Group, 2006. URL: http://www.aberdeen.com/summary/report/benchmark/RA_IT_ESB_PK_3170.asp (2007-10-01).

[KFS06] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proc. of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, 8-12 May 2006*, pages 915–922, 2006.

[KKR04] Michael Klein and Birgitta König-Ries. Coupled Signature and Specification Matching for Automatic Service Binding. In *Proc. of the European Conference on Web Service (ECOWS), Erfurt, Germany, 27-30 September 2004*, volume 3250 of *Lecture Notes in Computer Science*, pages 183–197, 2004.

[KLL$^+$05] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Proceedings of the Second European Semantic Web Conference, ESWC 2005*, volume 3532 of *Lecture Notes in Computer Science*, pages 1–16, Heraklion, Crete, Greece, May 2005. Springer.

[Kre01] H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). Technical report, IBM Software Group, 2001. URL: http://www.cs.uoi.gr/ zarras/mdw-ws/WebServicesConceptualArchitectu2.pdf (2007-10-01).

[Lar06] R. Lara. Two-phased web service discovery. In *Proceedings of the Workshop for AI-Driven Technologies for Services-Oriented Computing at the Twenty First National Conference on Artificial Intelligence (AAAI-06), Boston, USA, July 16th, 2006*, Boston, USA, July 2006.

[LD04] P. C. Lockemann and K. R. Dittrich. *Architektur von Datenbanksystemen.* dpunkt.verlag, Heidelberg, Germany, 2004.

[LdBKL06] H. Lausen, J. de Bruijn, U. Keller, and R. Lara. Semantic Web Services with WSMO. *Upgrade, European Journal for the Informatics Professional, Special issue on Web Services*, VII(5):34–37, 9 2006.

[Ley03] F. Leymann. Web Services: Distributed Applications Without Limits. In *BTW 2003, Datenbanksysteme für Business, Technologie und Web, Tagungsband der 10. BTW-Konferenz, Leipzig, Germany, 26.-28. February 2003*, volume 26 of *LNI*, pages 2–23, 2003.

[LF05] R. Lara and B. Foncillas. Economic and financial information management and the semantic web. In *Proceedings of the Industry Forum at the Second European Semantic Web Conference 2005 (ESWC 2005), Crete, Greece, 30 May, 2005.*, Heraklion, Crete, Greece, May 2005.

[LH03] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003. ACM, 2003*, pages 331–339, 2003.

[Li03] C. Li. Computing Complete Answers to Queries in the Presence of Limited Access Patterns. *VLDB Journal*, 12(3):211–227, 2003.

[LJDF$^+$97] L. Labed Jilani, J. Desharnais, M. Frappier, R. Mili, and A. Mili. Retrieving Software Components that Minimize Adaptation Effort. In *Proc. of the International Conference on Automated Software Engineering (ASE), Lake Tahoe, CA, USA, 2-5 November 1997*, pages 255–262, 1997.

[LN04] B. Ludäscher and A. Nash. Web Service Composition Through Declarative Queries: The Case of Conjunctive Queries with Union and Negation. In *Proc. of the 20th International Conference on Data Engineering (ICDE), Boston, MA, USA, 30 March - 2 April 2004*, page 840, 2004.

[LP72] D. Lorge Parnas. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[LZ05] J. Liu and F. Zhao. Service-Oriented Computing in Sensor Networks. In *Proc. of the 1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), Marina del Rey, CA, USA, 30 June - 1 July 2005*, volume 3560 of *Lecture Notes in Computer Science*, pages 397–398, 2005.

[Mar06] D. L. Margulius. Banking on SOA. *InfoWorld*, 29(2006-07-17), 2006.

[MBE03] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *VLDB Journal*, 12(4):333–351, 2003.

[MBH$^+$06] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. *OWL-S: Semantic Markup for Web Services*. OWL-S Coalition, 2006. URL: http://www.ai.sri.com/daml/services/owl-s/1.2/ (2007-10-01).

[MBL$^+$03] D. Martin, M. Burstein, O. Lassila, M. Paolucci, T. Payne, and S. McIlraith. *Describing Web Services using OWL-S and WSDL*. OWL-S Coalition, 2003. URL: http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html(2007-10-01).

[MBR01] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proc. of 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, 11-14 September 2001*, pages 49–58, 2001.

[MCSZ01] S. A. McIlraith, T. Cao Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[MdCV03] E. Marcos, V. de Castro, and B. Vela. Representing Web Services with UML: A Case Study. In *Proc. of the First International Conference on Service-Oriented Computing (ICSOC), Trento, Italy, 15-18 December 2003*, volume 2910 of *Lecture Notes in Computer Science*, pages 17–27, 2003.

[Mel04] S. Melnik. *Generic Model Management*, volume 2967 of *LNCS*. Springer-Verlag, Berlin, Germany, 2004.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proc. of the 18th International Conference on Data Engineering (ICDE), San Jose, CA, 26 February - 1 March 2002*, pages 117–128, 2002.

[Mic03a] Microsoft Corporation Inc. *Enterprise UDDI Services: Three Usage Scenarios*, 2003. URL: http://www.microsoft.com/windowsserver2003/techinfo/overview/ uddiscen.mspx (2007-10-01).

[Mic03b] Microsoft Corporation Inc. *UDDI Services: Qwest Technical Case Study*, 2003. URL: http://www.microsoft.com/windowsserver2003/techinfo/overview/qwest.mspx (2007-10-1).

[Mil95] G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *Proc. of the 19th International Conference on Very Large Data Bases (VLDB), Dublin, Ireland, 24-27 August 1993*, pages 120–133, 1993.

[MMHM01] S. Macroibeaird, A. T. Manes, S. Hinkelman, and B. McKee. *Using UDDI to Find ebXML Reg/Reps*, 2001. URL: http://www.ebxml.org/specs/rrUDDI.pdf (2007-10-01).

[MMM94] A. Mili, R. Mili, and R. Mittermeir. Storing and Retrieving Software Components: A Refinement Based System. In *Proc. of the 16th International Conference on Software Engineering, Sorrento, Italy, 16-21 May 1994*, pages 91–100, 1994.

[MZW93] A. Moormann Zaremski and J. M. Wing. Signature Matching: A Key to Reuse. In *Proc. of the First ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT FSE), Los Angeles, California, USA*, volume 18(5) of *ACM SIGSOFT Software Engineering Notes*, pages 182–190, 1993.

[MZW95] A. Moormann Zaremski and J. M. Wing. Signature Matching: A Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, 1995.

[MZW97] A. Moormann Zaremski and J. M. Wing. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.

[NL04] A. Nash and B. Ludäscher. Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns. In *Advances in Database Technology - EDBT 2004, Proc. of the 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, 14-18 March 2004*, volume 2992 of *Lecture Notes in Computer Science*, pages 422–440, 2004.

[NM02] S. Narayanan and S. A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. of the Eleventh International World Wide Web Conference (WWW), Honolulu, Hawaii, USA, 7-11 May 2002.*, pages 77–88, 2002.

[Noe05] J. Noel. BPM and SOA. Better Together. Technical report, IBM, 2005. URL: ftp://ftp.software.ibm.com/software/bigplays/AP-BPMSOA-BTW-00.pdf (2007-10-01).

[OAS01a] OASIS. *ebXML Catalog of Common Business Processes v1.0. Approved Catalog*, May 2001. URL: http://www.ebxml.org/specs/ (2007-10-01).

[OAS01b] OASIS, UN/CEFACT. *Core Component Overview. Version 1.05 ebXML Core Components*, 2001. URL: www.ebxml.org/specs/ccOVER.pdf (2007-10-01).

[OAS04] OASIS. *UDDI Version 3.0.2, UDDI Spec. Technical Committee Draft*, October 2004. URL: http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf (2007-10-01).

[Obj05] Object Management Group. *RAS 2.2: Reusable Asset Specification. OMG Available Specification*, 2005. URL: http://www.omg.org/technology/documents/formal/ras.htm/ (2007-10-01).

[Oes01] B. Oesterreich. *Objektorientierte Softwareentwicklung: Analyse und Design mit der UML*. Oldenbourg Verlag, Munich, Germany, 5 edition, 2001.

[OEtH02] J. O'Sullivan, D. Edmond, and A. H. M. ter Hofstede. What's in a Service? *Distributed and Parallel Databases*, 12(2/3):117–133, 2002.

[OEtH05] J. O'Sullivan, D. Edmond, and A. H. M. ter Hofstede. The Price of Services. In *Service-Oriented Computing, Proc. of the Third International Conference (ICSOC), Amsterdam, The Netherlands, 12-15 December 2005*, volume 3826 of *Lecture Notes in Computer Science*, pages 564–569, 2005.

[OR23] C. K. Ogden and I. A. Richards. *The Meaning of Meaning. A study of the Influence of Language upon Thought and of the Science of Symbolism.* Routledge & Kegan Paul, London, England, 10 edition, 1923.

[PvdH05] M. P. Papazoglou and W.J. van den Heuvel. Service Oriented Architectures. to appear in VLDB Journal, 2005.

[QW04] J. Quantz and T. Wichmann. Report on Current Usage of Web Services and Semantic Web. Technical Report D12.1, DERI, 06 2004. URL: http://www.deri.at/fileadmin/documents/deliverables/DIP/D12.1.pdf (2007-10-01).

[RB01] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[RdBM$^+$06] D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, and D.r Fensel. WWW: WSMO, WSML, and WSMX in a Nutshell. In *The Semantic Web, Proc. of the First Asian Semantic Web Conference (ASWC), Beijing, China, 3-7 September 2006*, volume 4185 of *Lecture Notes in Computer Science*, pages 516–522, 2006.

[RE06] T. Ritter and R. S. Evans. *SOA Justification. A survey of financial justification among SOA adopters in North America and Western Europe.* GCR Custom Research, 2006. URL: http://www.bea.com/content/news_events/white_papers/BEA_Costs_and_Benefits_GCR_Survey_Final.pdf (2007-10-01).

[Res95] P. Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *Proc. of the 14th International*

Joint Conference on Artificial Intelligence (IJCAI), Montral, Qubec, Canada, 20-25 August 1995, pages 448–453, August 1995.

[RHDM04] E. Rahm, H. Hai Do, and S. Massmann. Matching Large XML Schemas. *SIGMOD Record*, 33(4):26–31, 2004.

[Rog05] R. Rogers. Reuse Engineering for SOA. Technical report, IBM, 2005. URL: http://www-128.ibm.com/developerworks/webservices/library/ws-reuse-soa.html (2007-10-01).

[RR98] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*. Springer, 1998.

[SB88] G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[SB02] D. Shasha and P. Bonnet. *Database Tuning. Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufman Publishers, San Francisco, CA, USA, 2002.

[SB05] Q. Z. Sheng and B. Benatallah. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In *Proc. of the International Conference on Mobile Business (ICMB), Sydney, Australia, 11-13 July 2005*, pages 206–212, 2005.

[Sch97] H. Schierenbeck. *Ertragsorientiertes Bankmanagement. Grundlagen, Marktzinsmethode und Rentabilitäts-Controlling*, volume 1. Gabler Verlag, Wiesbaden, Germany, 5 edition, 1997.

[SE05] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics*, 3730:146–171, 2005.

[SKS05] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, New York, USA, 5 edition, 2005.

[Sow00] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, USA, 2000.

[Spi03] M. Spielmann. Verification of Relational Transducers for Electronic Commerce. *Journal of Computer and System Sciences*, 66(1):40–65, 2003.

[SPS06] N. Srinivasan, M. Paolucci, and K. P. Sycara. Semantic Web Service Discovery in the OWL-S IDE. In *Proc. of the 39th Hawaii International International Conference on Systems Science (HICSS-39 2006), Kauai, HI, USA, 4-7 January 2006*, 2006.

[SS04] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer-Verlag Berlin, Germany, 2004.

[Ste02] The Stencil Group, Inc. *The Evolution of UDDI. UDDI.org White Paper*, 2002. URL: http://uddi.org/pubs/the_evolution_of_uddi_20020719.pdf (2007-10-01).

[TM07] A. Thomas-Manes. Tools unterstützen die SOA Verwaltung. *Computer Woche*, (1):22–23, 2007.

[UHMM04] S. Urman, R. Hardman, M. McLaughlin, and M. MacLaughlin. *Oracle Database 10g PL/SQL Programming*. McGraw-Hill, New York, USA, 2004.

[Usc03] M. Uschold. Where Are the Semantics in the Semantic Web? *AI Magazine*, 24(3):25–36, 2003.

[VB96] G. Vossen and J. Becker, editors. *Geschäftsprozessmodellierung und Workflow-Management. Modelle, Methoden, Werkzeuge*. mitp, Bonn, Germany, 1996.

[Ver06] K. Verma. *Configuration and Adaptation of Semantic Web Processes*. PhD thesis, Department of Computer Science, The University of Georgia, 2006.

[VH07] G. Vossen and S. Hagemann. *Unleashing Web 2.0 - From Concepts to Creativity*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2007.

[Vos00] G. Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. Oldenbourg Verlag, Munich, Germany, 4 edition, 2000.

[Vos06] G. Vossen. Have Service-oriented Architectures Taken a Wrong Turn Already? In *Proc. of the IFIP TC 8th International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS), Vienna, Austria, 24-26 April 2006*, volume 205 of *IFIP International Federation for Information Processing*, 2006.

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, Munich, Germany, 2005.

[Wes05] P. Westerkamp. *Flexible Elearning Platforms: A Service-Oriented Approach.* PhD thesis, Westfälische Wilhelms-Universität Münster, Germany, 2005.

[WKR+05] U. Wahli, T. Kjaer, B. Robertson, F. Satoh, F. J. Schneider, W. Szczeponik, and C. Whyley. *WebSphere Version 6 Web Services Handbook Development and Deployment.* IBM Red Books, 2005.

[WLWC05] L. Wang, P. Li, Z. Wu, and S. Chen. A Service Oriented Implementation of Distributed Status Monitoring and Fault Diagnosis Systems. In *Proc. of the 5th International Conference on Computational Science (ICCS), Atlanta, GA, USA, 22-25 May 2005, Part I*, volume 3514 of *Lecture Notes in Computer Science*, pages 568–575, 2005.

[Wor99] World Wide Web Consortium. *XSL Transformations (XSLT) 1.0. W3C Recommendation*, 1999. URL: http://www.w3.org/TR/xslt (2007-10-01).

[Wor03a] World Wide Web Consortium. *SOAP Version 1.2. Part 1: Messaging Framework. W3C Recommendation*, June 2003. URL: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ (2007-10-01).

[Wor03b] World Wide Web Consortium. *SOAP Version 1.2. Part 2: Adjuncts. W3C Recommendation*, June 2003. URL: http://www.w3.org/TR/2003/REC-soap12-part2-20030624/ (2007-10-01).

[Wor04a] World Wide Web Consortium. *OWL Web Ontology Language Guide. W3C Recommendation*, 2004. URL:

http://www.w3.org/TR/2004/REC-owl-guide-20040210/ (2007-10-01).

[Wor04b] World Wide Web Consortium. *OWL Web Ontology Language Overview. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/ (2007-10-01).

[Wor04c] World Wide Web Consortium. *OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/ (2007-10-01).

[Wor04d] World Wide Web Consortium. *RDF Primer. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (2007-10-01).

[Wor04e] World Wide Web Consortium. *RDF Semantics. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ (2007-10-01).

[Wor04f] World Wide Web Consortium. *RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (2007-10-01).

[Wor04g] World Wide Web Consortium. *Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (2007-10-01).

[Wor04h] World Wide Web Consortium. *Web Services Architecture Requirements. W3C Working Group Note*, 2004. URL: http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/ (2007-10-01).

[Wor04i] World Wide Web Consortium. *XML Schema Part 2: Datatypes Second Edition. W3C Recommendation*, 2004. URL: http://www.w3.org/TR/xmlschema-2/ (2007-10-01).

[Wor04j] World Wide Web Consortium (W3C). *RDF/XML Syntax Specification (Revised). W3C Recommendation*, 2004. URL: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (2007-10-01).

[Wor06] World Wide Web Consortium. *Namespaces in XML 1.0 (Second Edition). W3C Recommendation*, 2006. URL: http://www.w3.org/TR/REC-xml-names/ (2007-10-01).

[Wor07a] World Wide Web Consortium. *Semantic Annotations for WSDL and XML Schema. W3C Recommendation*, 2007. URL: http://www.w3.org/TR/2007/REC-sawsdl-20070828/ (2007-10-01).

[Wor07b] World Wide Web Consortium. *SPARQL Query Language for RDF. W3C Candidate Recommendation*, 2007. URL: http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/ (2007-10-01).

[Wor07c] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation*, June 2007. URL: http://www.w3.org/TR/2007/REC-wsdl20-20070626/ (2007-10-01).

[Wor07d] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts. W3C Recommendation*, June 2007. URL: http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626 (2007-10-01).

[Wor07e] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0: RDF Mapping. W3C Working Group Note*, 2007. URL: http://www.w3.org/TR/2007/NOTE-wsdl20-rdf-20070626/ (2007-10-01).

[Wor07f] World Wide Web Consortium. *XQuery 1.0: An XML Query Language. W3C Recommendation*, 2007. URL: http://www.w3.org/TR/2007/REC-xquery-20070123/ (2007-10-01).

[ZB96] V. Zeitham and M. Bitner. *Service Marketing.* McGraw Hill, New York, USA, 1996.

Appendix A

Case Study from Financial Industry

In the following, the application scenario for a service-based calculation kernel for pricing of financial products is summarized.

The operations of such a calculation kernel are used in different application contexts:

- The marketing department uses the algorithms to adjust the interest rates for loans and savings when the market conditions change. The aim is to keep the profit margin up.
- A corporate customer consultant must be able to make an individual loan offer for important customers, yet be able to calculate the profit margin with the same algorithms as for a private customer's standard loan.
- Later, when the loan is granted, the same algorithms are used to calculate the profit margin which is then imported into the data warehouse and used for reporting purposes. This ex-post calculation ought to be executed with exactly the same algorithms to ensure consistency between ex-ante and ex-post results.

These examples show that the functionality of a financial calculation kernel is destined to be reused, not only for technical, but also for conceptual consistency. Therefore, a standardized access as granted by Web service interfaces is desirable. Mathematically, a profit margin can be defined in many different ways, depending on the product type, the opportunity that the transaction is compared to, and additional regulatory restrictions. Different departments are the users of the same algorithms. Technically, the algorithms are provided by programs in the care of the IT department.

Therefore, the search categories of a UDDI such as the provider and its field of industry are insufficient classification criteria. It is neither important how the operations are grouped into individual services nor is it important who provides the service. For internal software development and maintenance, it is necessary to search on the level of operations and refine this search based on input and output parameters. The financial calculation kernel is used as an example of company-internal service-oriented software development.

Section A.1 gives an introduction to loan calculation and pricing mathematics. This scenario is then used to show how an application ontology is developed that supports search for operations, including their input and output parameters. As a financial calculation kernel is highly domain specific, an application ontology is modeled that combines domain and task ontologies. Section A.3.1 briefly summarizes the requirements for the financial calculation ontology. Section A.3.2 documents the conceptual design of the ontology for financial calculations.

The modeling approach for the ontology follows the design method as introduced by [Hüs05]. First, a conceptual model is designed. This is transformed into a logical model based on RDFS and OWL. How to derive this logical model is explained in [Hüs05] and is not shown here as the conceptual model is sufficient to motivate the suggested semantic search approach and to analyze its characteristics.

A.1 Introduction to Loan Pricing

This section gives a short introduction to different types of loans which are of interest for the following ontology model. For brevity, the presentation is restricted to loans with fixed interest rate and fixed duration. The explanations how these loans are calculated are based on examples. A detailed description of different products including variable condition loans and the related calculation methods is given in [Sch97].

A loan with fixed conditions, e.g., a mortgage loan is determined by the initial amount that the customer gets from the bank, the so-called net value, the installment, the interest rate, the payment periodicity, and the time that the customer has to pay back the money. This information is already sufficient to calculate the cashflow that is the sequence of in- and out-going payments. There are different types of fixed condition loans that vary in the way installment and interest are payed. Three basic types are introduced here to explain the importance of the chosen product type for all following calculations.

Deferred Payment Loan: A deferred payment loan is often chosen when consumer goods are financed. For example, a customer buys a new TV set for 2,000 Euros. The department store offers to finance this set for one year. The customer must pay money for the TV, the interest and a service charge. The sum of these three components is called the total debt. Assuming that the gross interest rate is 5.0% per year the customer must pay $2,000 * 1.05/12 =$ 175 Euros per month. The intial service charge is 60 Euros. Legally, the service charge must be regarded as part of the net interest rate which results in a net interest rate of $(2,000*0.05+60)/2,000 = 160/2,000 = 0.08$ per year. In total, the customer pays $2,160/12 = 180$ Euros per month as installment, which includes amortization, interest, and service charge. The total debt is 2,160 Euros. The cashflow is depicted in Figure A.1.

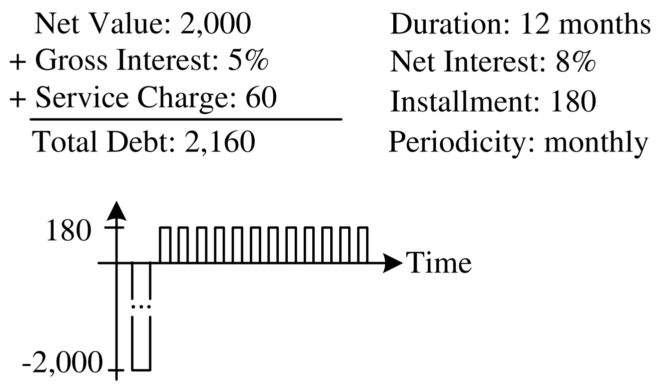
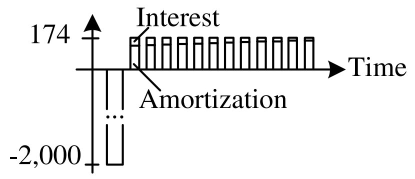


Figure A.1: Deferred payment loan.

Typically, the cashflow of a deferred payment loan does not allow to differentiate between installment, interest, and additional charges. The total debt is payed back in equal payments according to the payment periodicity.

Annuity Loan: An annuity loan is often used for mortgage loans. The net value of the loan is usually larger than a deferred payment loan. The customer pays back the loan in installments that consist partially of interest and partially of amortization. The interest portion of the installment decreases over time, whereas the amortization increases. For comparison with the consumer loan, the same example is calculated with the same interest rate and for one year. The monthly installment is 174 Euros. This is depicted in Figure A.2.

Net Value: 2,000
Duration: 12 months
Gross Interest: 8%
Installment: 173.98
Periodicity: monthly



Month	Balance	Interest	Amortization	Installment
0	2,000.00	0.00	0.00	0.00
1	1,839.36	13.33	160.64	173.98
2	1,677.64	12.26	161.71	173.98
3	1,514.85	11.18	162.79	173.98
4	1,350.97	10.10	163.88	173.98
5	1,186.00	9.01	164.97	173.98
6	1,019.93	7.91	166.07	173.98
7	852.75	6.80	167.18	173.98
8	684.46	5.69	168.29	173.98
9	515.05	4.56	169.41	173.98
10	344.50	3.43	170.54	173.98
11	172.82	2.30	171.68	173.98
12	0.00	1.15	172.82	173.98
	Total:	87.72	2,000.00	2,087.72

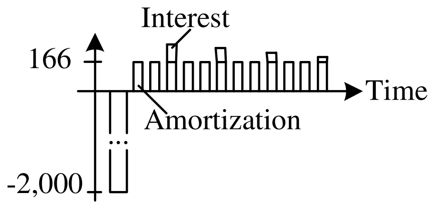
Figure A.2: Annuity loan cashflow.

Typically, the cashflow of a deferred payment loan consists of equal installments with varying interest and amortization parts. Interest and amortization are therefore always payed with the same periodicity.

Amortization Loan: An amortization loan allows to choose independent periodicities for amortization and interest payments. The amortization payments are fixed; the interest payment is payed independently and decreases throughout the payment period. Therefore, the total installment of amortization and interest varies. The given example of a 2,000 Euros loan with 8% interest per year is repaid with amortizations of 166 Euros per month and quarterly interest payments. This is shown in Figure A.3.

Operations in Loan Pricing: For every loan, the cashflow is the determining factor for pricing. Therefore, the cashflow calculation routines are central components of a financial calculation kernel and are to be exposed as Web service operations. The *initial cashflow* is the cashflow as calculated ex-ante for loan pricing. It is only calculated once at the beginning. The *remaining cashflow* is also calculated. It consists of all remaining payments that are still to be payed by the customer.

Net Value: 2,000
Duration: 12 months
Gross Interest: 8% per year
Interest Periodicity: quarterly
Amortization: 166.66
Amortization Periodicity: monthly



Month	Balance	Interest	Amortization	Installment
0	2,000.00	0.00	0.00	0.00
1	1,833.33	0.00	166.67	166.67
2	1,666.67	0.00	166.67	166.67
3	1,500.00	36.67	166.67	203.33
4	1,333.33	0.00	166.67	166.67
5	1,166.67	0.00	166.67	166.67
6	1,000.00	26.67	166.67	193.33
7	833.33	0.00	166.67	166.67
8	666.67	0.00	166.67	166.67
9	500.00	16.67	166.67	183.33
10	333.33	0.00	166.67	166.67
11	166.67	0.00	166.67	166.67
12	0.00	6.67	166.67	173.33
	Total:	86.67	2,000.00	2,086.67

Figure A.3: Amortizable loan cashflow.

A financial consultant does not only need a cashflow routine to calculate a payment plan for the customer but also different additional routines which compute some of the factors that influence the cashflow. Given a fixed interest rate which is determined by the marketing department the following situations are possible:

- The customer is willing to pay installments of a given amount for a given period of time and with a given periodicity. The consultant must calculate the net value of the loan that can be payed back under these conditions.
- The customer needs a loan of a given net value and wants to pay it back within a given period of time in a given payment frequency. The consultant must calculate the amount of the installments.
- The customer needs a loan of a given net value and is able to pay installments of a fixed amount in a given periodicity. The consultant must calculate how long it takes to pay back the loan.

These routines are also derived from the cashflow calculation operations. They can also be used in ETL processes if missing data is to be replaced in a data warehouse. Therefore, these operations are also to be exposed as Web service operations.

The price of a loan is calculated in many different ways, for example, as periodical profit margin or as present value of the expected cashflow. Both pricing methods are explained next using simple examples.

Profit Margin Calculation A financial product like any product that is sold on a market needs a price calculation method that allows to monitor if the producer of the product creates a revenue by selling the product. In industry, a producer has costs producing a product and sells it at a higher price. Simplified, the revenue is the difference between production costs and sales price.

A bank sells loan and savings products. The price that the customer pays for a loan is the interest rate that the bank charges. If the bank grants a loan the treasury department has to re-finance this investment with money from the inter-bank money market. The costs for the "production" of the loan is the interest rate that the bank has to pay at the inter-bank money market. The revenue is the spread between the interest rate that the bank charges the customer and the interest rate that the bank has to pay for refinancing. If the bank gives a one-year loan to a customer for an interest rate of 4% and has to pay 3.5% at the inter-bank market, the revenue is 0.5%, assuming that the two loans have exactly the same cashflow. This is shown in Figure A.4. The interest rate at the inter-bank market varies depending on the time period for which it is fixed. Under normal market conditions, the interest rate increases with increasing loan duration.

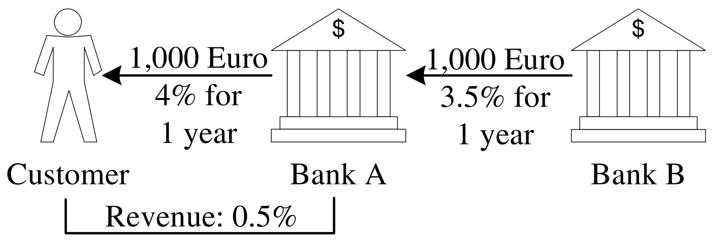


Figure A.4: Loan revenue.

This simple comparison between customer loan conditions and inter-bank market conditions is simplified as it assumes that the customer loan and the loan between two banks have the same cashflow. If this is not the case, the profit margin calculation is more complex, but the idea remains the same.

A second simplification is the difference in credit worthiness between an ordinary customer and a bank. In reality, the risk that a bank cannot pay back a loan issued by another bank is less than the risk that a customer is unable to pay back the money. To estimate the risk that the bank runs in giving money to a customer the bank evaluates personal information about the customer, compares this information to statistical findings, and assigns a score to each customer. This scoring expresses the probability that the customer will fail to pay back the money within a fixed time-period, usually within one year. Customers with a bad score have to pay a higher interest rate than customers with a good score. The surcharge on the interest rate is like an insurance fee against loan default. It is not considered as revenue, but as cost. This is depicted in Figure A.5.

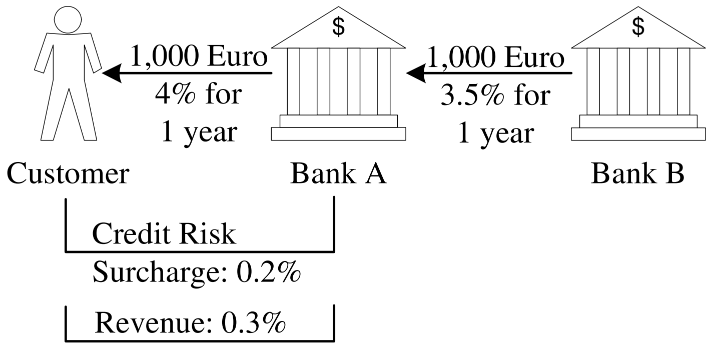


Figure A.5: Loan revenue and risk costs.

The information evaluated for scoring depends on the customer type. Private customers have to state what they earn, how many people they have to sustain with their income, etc. Corporate customers have to disclose their financial statements. The statistical information and the mathematical computations needed to calculate a scoring are usually contained in packaged applications by third party software vendors or by financial associations.

This simple example shows that the interest rate for the customer is determined by the conditions of the inter-bank market depending on two aspects: the loan cashflow and the customer score. As the interest rates at international markets vary, customer conditions must be updated regularly. Other aspects such as the amount of the loan and the pay back modalities also influence the interest rate.

Present Value Calculation: The profit margin, as explained above, distributes the revenue equally across the loan duration as yearly revenue. The present value is a time independent measure that calculates the value that the complete future cashflow has on the day the loan is granted. If a customer gets a loan of 1,000 Euros and pays it back after one year with 4% interest, i.e., the customer pays back 1,040 Euros, the bank re-finances this future payment at the inter-bank market for 3.5%. The question is: how much money can the bank borrow at the inter-bank market on the day the loan is issued to have 1,040 Euros at the end of one year with an interest rate of 3.5%? In the given example, the bank borrows 1,004.83 Euros on the day the loan is granted. When the customer pays back 1,040 Euros after one year the bank can repay its own loan at 3.5% interest because $1004.83 * 1.035 = 1,040$. Therefore, the bank has earned 4.83 Euros on the day the loan is granted. This is depicted in Figure A.6.

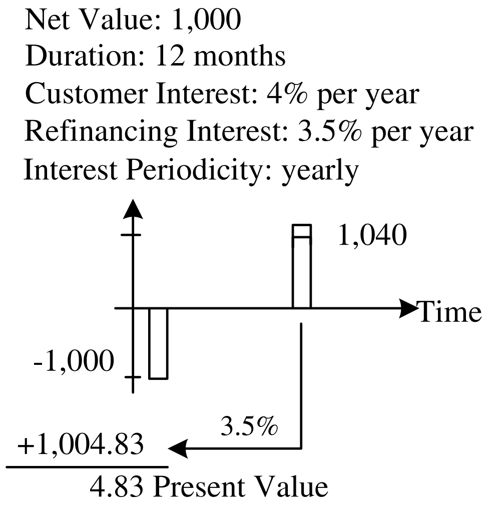

Net Value: 1,000
Duration: 12 months
Customer Interest: 4% per year
Refinancing Interest: 3.5% per year
Interest Periodicity: yearly
1,040
Time
-1,000
3.5%
+1,004.83
4.83 Present Value

Figure A.6: Present value calculation.

The inter-bank loan is a theoretical construction to calculate a present value for the customer loan. Each payment of the customer cashflow must be refinanced as shown above at the interest rate that is applicable for the given time period. The treasury of the bank is responsible for the refinancing strategy of all loans, savings, etc. Therefore, the treasury also needs the cashflow calculation routines of the financial calculation kernel, but the treasury is not forced to refinance every loan exactly as it is granted to the customer. The present value calculation as presented here results in an initial gross present value. In a second step, the value must be adjusted to reflect the customer rating. This results in an initial net present value that is risk-adjusted.

The different application contexts for financial calculations show that a financial calculation kernel offers a high potential for software re-use because many calculation routines are needed by the marketing department, the customer consultants, the ETL process of a data warehouse for internal controlling, and the treasury department. Therefore, it is justified to expose these operations as Web service operations. Further, the calculation routines are highly embedded into the application domain. Therefore, a semantic search for these operations that supports software designers must be based on a domain and application specific ontology. This is modeled in Appendix A.3.

A.2 Online Loan Application Case Study

Requirement Analysis

The requirement analysis has to identify business goals that drive the creation of a new application, the stakeholders that have an interest in the new application and the business domains that they represent. This can be derived from interviews and an analysis of the business use cases that the application is to support. The requirement analysis is conducted from the point of view of the end user.

The following business goals are identified for the loan application as shown in Table 2.2.

Table A.1: Loan application goals.

No.	**Description**	**Type**
1	Increase revenue	general
2	Increase number of consumer loans	general
3	Provide self-service loan application capability	functional
3.1	Provide user-friendly application experience	functional
3.2	Provide customer-specific loan offers	functional
4	Monitor usage of Internet application	functional

The first two business goal are general business goals. The other goals are application-specific functional goals that are likely to have an influence on the application realization and the services needed.

In a second step, candidate use cases are identified that are to be supported by the new application. The following use cases have been identified as shown in Figure A.7.

The use cases are described in detail in Table 2.3.

In a third step, the business domains are identified that are involved in the use-cases and in achieving the business goals as shown in Figure 2.24. Consumer loans belong to the group of retail banking products. Therefore, the customer front end belongs to the retail domain. Monitoring the usage of the application is a technical facility for marketing purposes. The information needed to offer a customer-specific loan comes from the controlling department. They monitor the interest rates and prices at the international banking markets and regularly compute new conditions for consumer loans that have to be used by the loan application.

In this stage, the analysis is conducted without taking existing services into account. Search support for services is not needed in the early requirement evaluation.

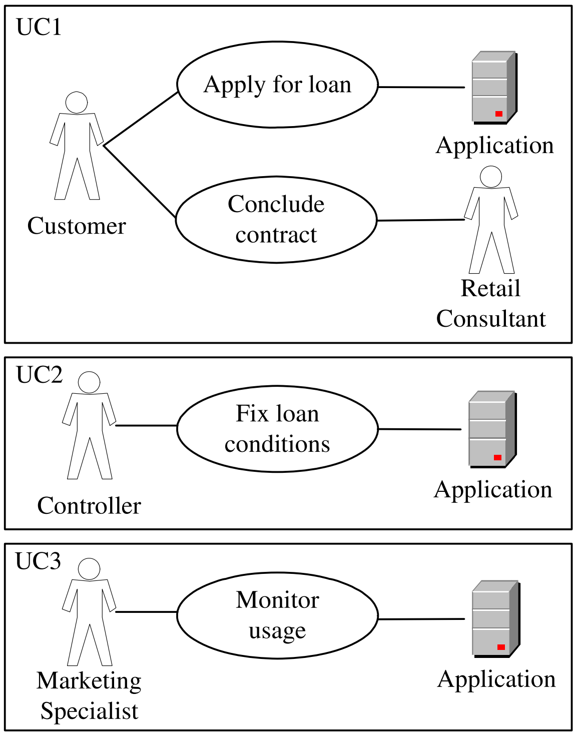


Figure A.7: Loan application use cases.

Table A.2: Loan application use cases.

No.	**Description**	**Actors**
UC1	The customer applies for a loan on the Internet, fills out the forms and gets a contract by post. This is sent back to the bank and checked by a sales consultant.	Customer, Application, Sales Consultant
UC2	A controller regularly updates the loan conditions used by the application according to the changes in interest rates.	Controller, Application
UC3	A marketing specialist monitors the application.	Marketing Specialist, Application

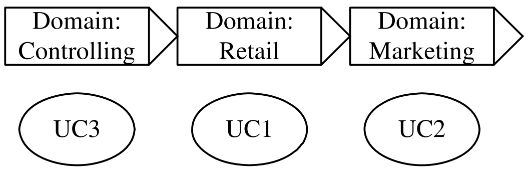


Figure A.8: Loan application business domains.

Conceptual Design

In the conceptual design step, the use cases of the requirement analysis are refined. From this model, the service provider identifies system components, models the internal behavior of each component, and also the flow between components. On the service consumer side, the results of the analysis are used to identify candidate services, the communication between them and other system components (e.g., a GUI), and to map candidate services to candidate components.

For a more detailed analysis, the business activities are modeled that are executed in each use case. The activities of the loan application use case are modeled in the activity diagram shown in Figure A.9. If an activity is executed successfully, the next activity is executed. If an error occurs, an error report is created. For a better readability, the error activities are depicted at the bottom of the diagram. All arrows that do not point to any activity represent error cases and are to be connected with the error diagram

In the first step of the application process, the customers state details about the loan they need, the net value of the loa, and the duration. Depending on this information, monthly installment and interest rate are calculated assuming that the customers have the best credit scoring possible. Next, the customers have to give details about their private and professional life to assess their credit worthiness by using a standardized scoring method. Finally, the customers enter their address data and ask for a loan application form that is then computed for the individual customer while taking the exact customer scoring into account. The application form is sent by traditional mail. All interactions between the customer and the online credit application are monitored. This is represented by the logging activities.

From this diagram, at least three components can be identified, a logging component, a component for loan condition calculation and a component for customer scoring calculation. The creation and mailing of the contract can either be assigned to a mailing component or executed by a human being.

The monitoring results are read and displayed when the marketing specialist requests them. This short activity sequence for the monitoring use

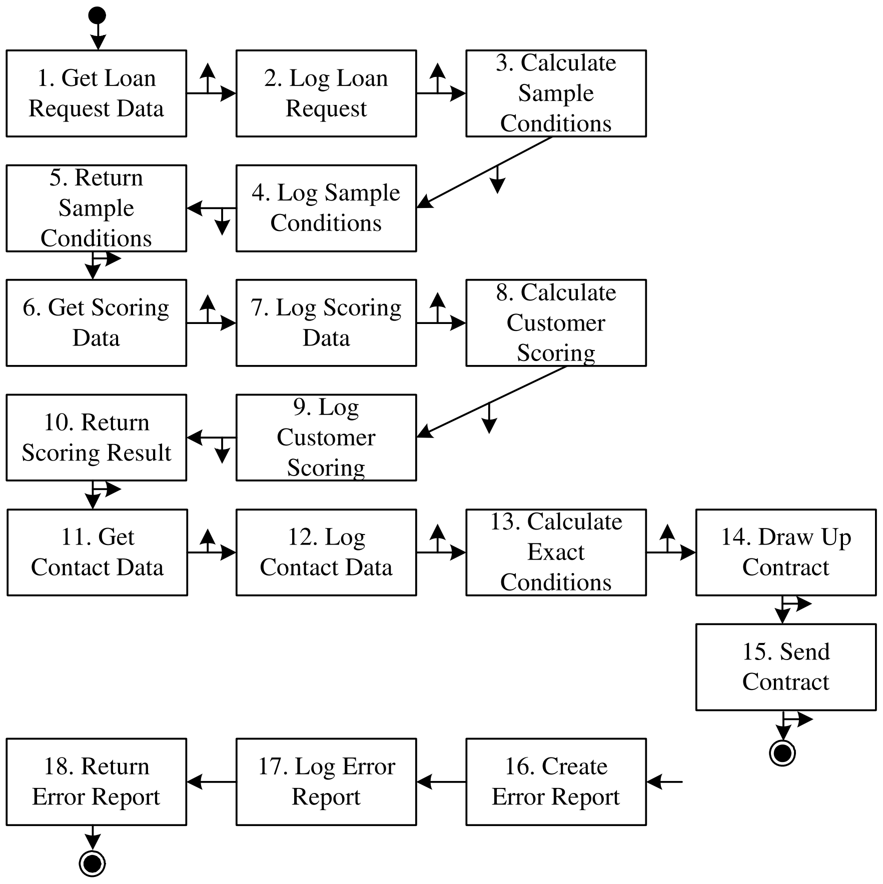


Figure A.9: Activity diagram for loan application use case (UC1).

case is shown in Figure A.10.

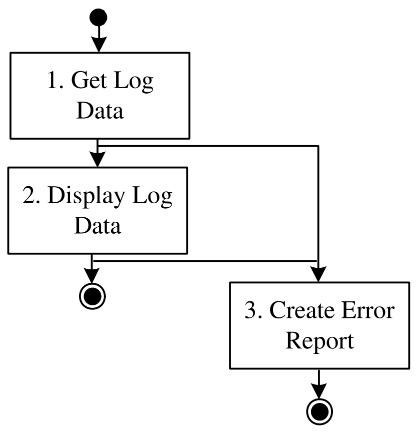


Figure A.10: Acticity diagram for monitoring use case (UC2).

The activities are executed by a monitoring component in the online application that offers a service to read the log data.

The update of the loan conditions is executed regularly by a controller. For this, interest rate information from an information provider is requested and stored in the interest rate table of the online application. Then, the relevant loan data is fetched from the basic configuration of the online application. This information consists of all pre-configured combinations of loan duration, net value, and customer scoring. For these combinations the new loan conditions, installment and interest rate, are computed. Finally, the loan condition table of the online application is updated. This is depicted in Figure A.11.

In this activity diagram, the online application, a component for loan condition calculation, and an information service provider for interest rates are involved.

In the conceptual design phase, the following system components and services have been identified as listed in Table A.3. An initial mapping of services to components has already taken place in this table.

The components reflect the service provider view and the logical grouping of functionality from the perspective of an application designer. The services show the external perception of the system by the client.

Logical Design

In the logical design step, the following questions must be answered for every identified service of the conceptual design step:

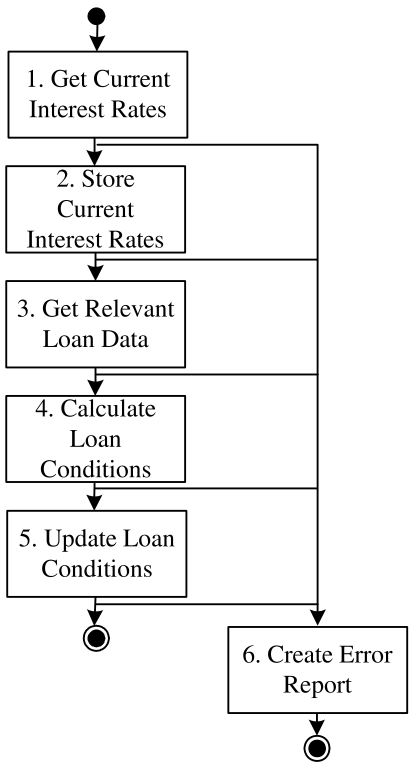

1. Get Current Interest Rates
2. Store Current Interest Rates
3. Get Relevant Loan Data
4. Calculate Loan Conditions
5. Update Loan Conditions
6. Create Error Report

Figure A.11: Activity diagramm for controlling use case (UC3).

Table A.3: Conceptual components and services.

Component (Service provider view)	**Service (Service consumer view)**
	Interest Rate Information Service
Loan Condition Component	Condition Calculation Service Condition Look-up Service
Scoring Component	Score Calculation Service
Contract Component	Contract Creation Service Mail Service
Monitoring Component	Logging Service Log Evaluation Service
Administration Component	Interest Rate Administration Service Loan Condition Administration Service
End User GUI	
Monitoring GUI	
Administration GUI	

- On which layer of the SOA architecture does the service logically reside? Is it a functional or a non-functional service? Is it an atomic or a composite service? Does it govern a business process?
- Is there an already existing service that can be reused for service realization? How can the desired service be mapped to existing service functionality (top-down)?
- If the service does not exist yet: is there an existing component that can be exposed as service (bottom-up)? Does the service have to be implemented from scratch?

A service profile for each service identified is compiled as shown in Table A.4.

Table A.4: Service profile.

Aspect	**Description**
Functionality	Which business aspects, processes or functions does the service support?
Accessibility	Which business process uses the service? Which GUI element uses the service? Which applications access the service?
Process	What is the relationship between the events that the service reacts to and the actions that the service takes?
Information	What data is sent to the service? Where from? What data is sent by the service? Where to?
Interaction	How does the calling application interact with the service? How does the service interact with other services or applications?

On the component side, the existing components that help to realize the application must be specified further. The identified sub-systems must be refined to reflect the individual components and the flow between them.

The result of this step is a mapping from needed services to existing services and components and from needed services to newly to be implemented functionality respectively. For every functionality identified, it is decided if it is published as new service for future reuse through other applications.

For the first use cas, a detailed process diagram is drawn that shows the interaction between the customer at the end user GUI and the application for the first subprocess in the application process, the calculation of sample conditions. The interaction between GUI and Loan Condition Component is shown in Figure A.12. For a better overview, the monitoring activities are omitted from this diagram.

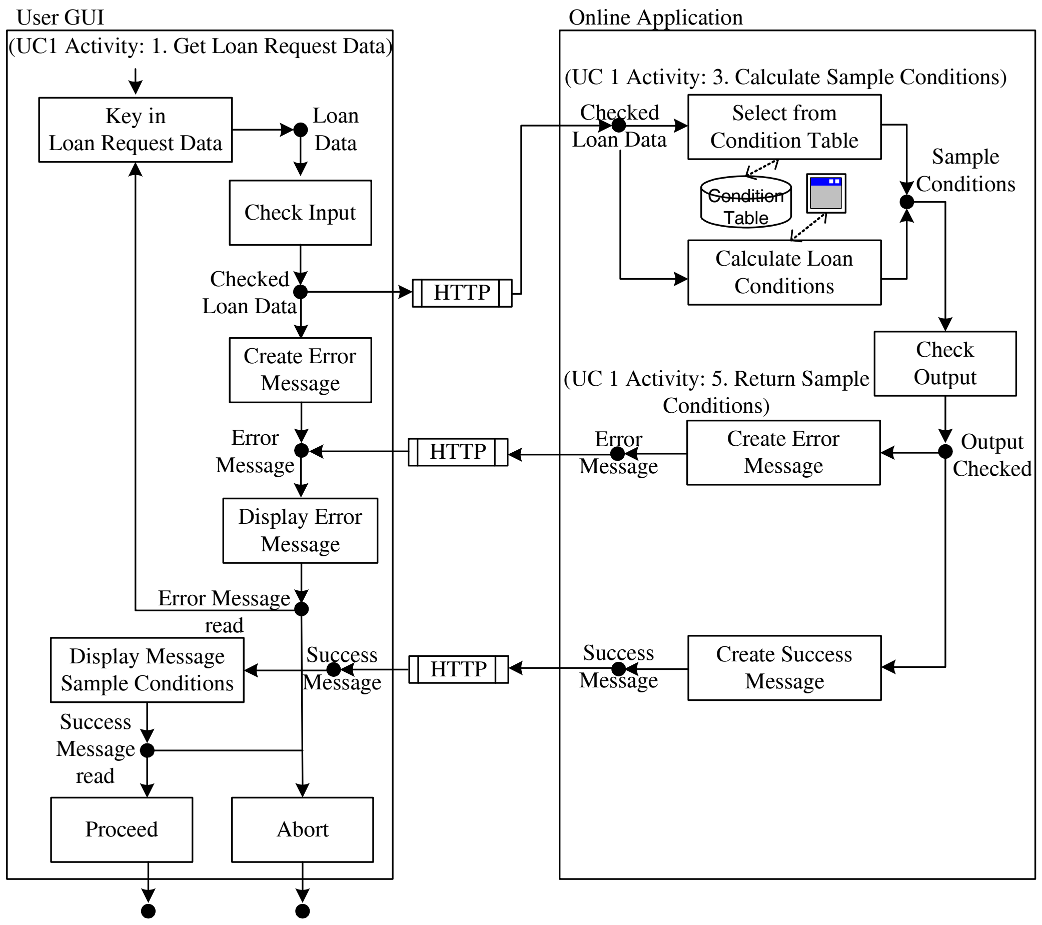


Figure A.12: Calculation of sample conditions.

To get a sample calculation, the customers must enter the net value and the duration of the loan. The net value may range from 1,000 to 60,000 Euros, the duration must be either 12, 24, 36, 48, or 60 months. If the customers enter invalid loan data, they receive an error message and can either abort or try again. If the loan data is valid, the monthly installment and interest rate are looked up in a database table that contains all relevant combinations of net value, duration, and customer scoring. This data retrieval operation assumes an optimal customer scoring by default. If the information is not available in the database, it needs to be computed. If the installment is less than 25 Euros, an error message is created. Otherwise, a success message is sent and the users are informed about the monthly installment and interest rate. Here, users may abort or proceed.

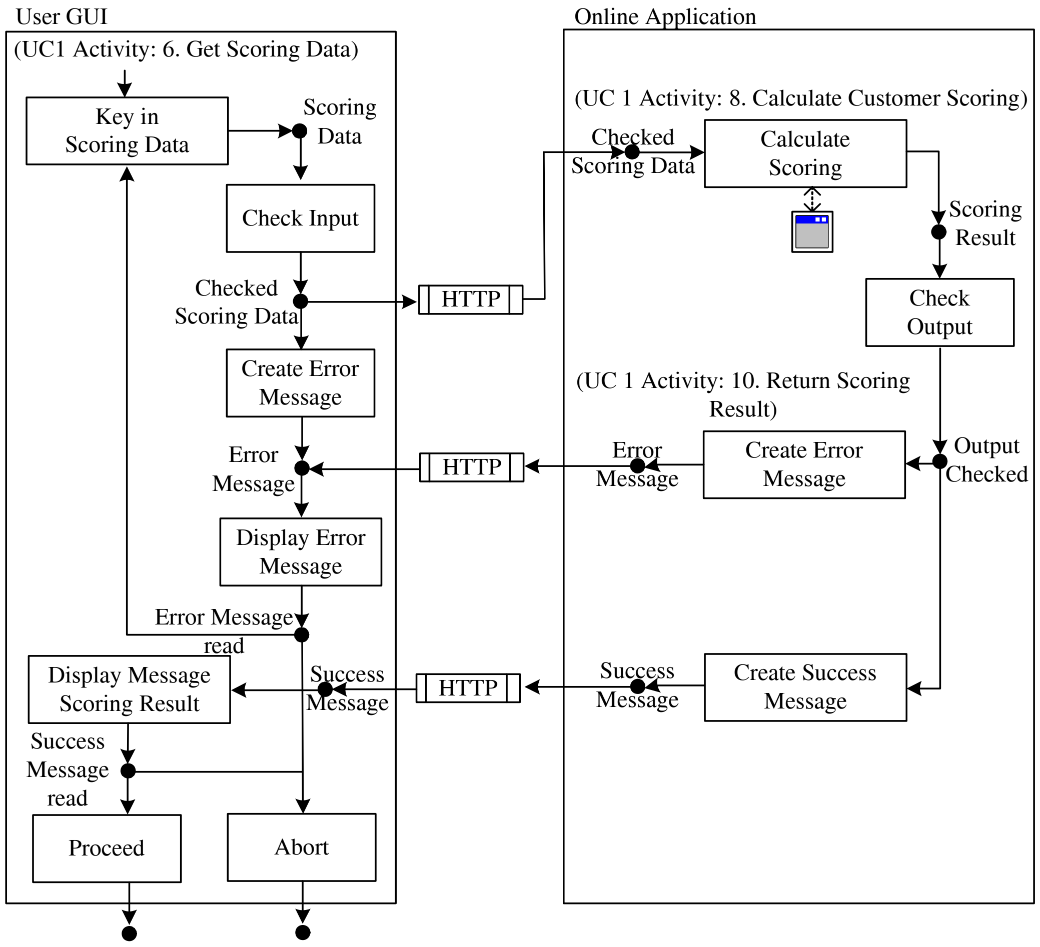


Figure A.13: Calculate customer scoring.

In the second subprocess, the exact customer scoring is calculated as shown in Figure A.13. The customers must enter details about their professional and private living conditions. If they omit a required information or make a mistake, they get an error message. Otherwise, the data is sent to the bank. There, a private customer scoring needs to be computed. If the customer has an acceptable score, a success message is sent back. Otherwise, an error message is created. At the end of this step, customers may abort or proceed.

The last subprocess is the final loan application as shown in Figure A.14. The users enter their contact data to receive a loan application form. If the users enter invalid data, they get an error message. If the data is valid,

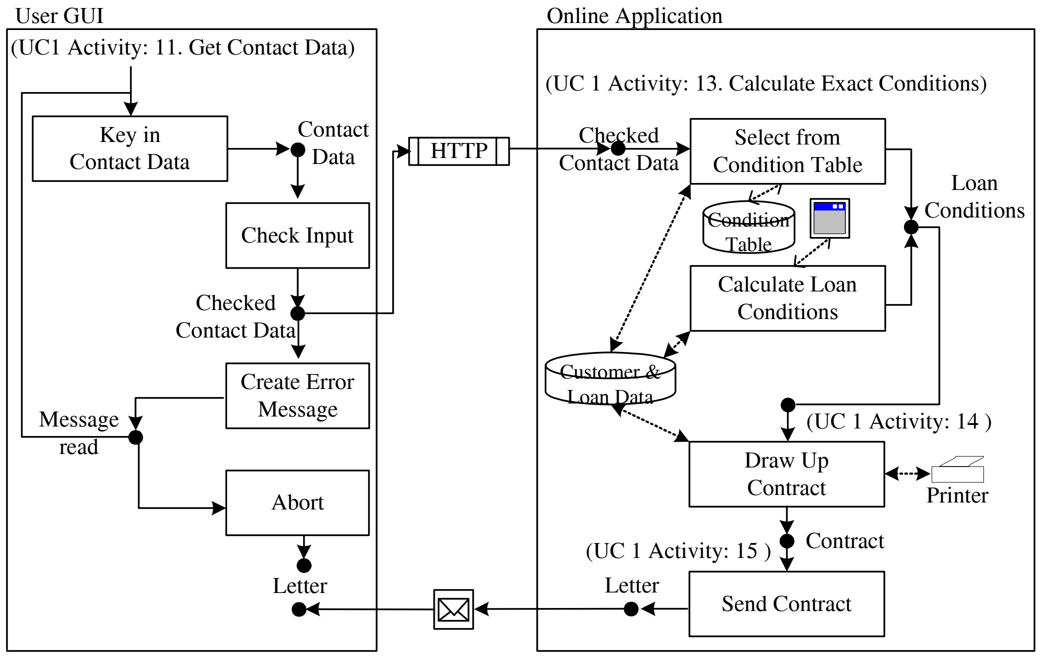


Figure A.14: Calculation of exact conditions.

the exact loan calculation is executed. Just like the sample calculation, the exact calculation can either look up the monthly installments and the interest rate in a table with precalculated results or calculate the conditions on the fly. This time, the exact customer scoring that has been computed in the step before is taken into account. All information is processed so that an application form can be printed. This is then sent to the customers by post.

These are the core processes of the application. Behind the scenes, an additional administration process is needed as shown in Figure A.15. As interest rates fluctuate on the international banking market, it is necessary to compute a new condition table once a month to keep up the profit margin. This calculation needs current interest rates for the Euro currency zone as the bank only offers loans in Euro currency. The calculation of the condition table repeatedly executes the step “Calculate Loan Conditions” for all relevant combinations of net values, durations, and customer scorings.

From the analysis of information processed in the business subprocesses the following data model can be derived as shown in Figure A.16. The table `InterestRates` contains the current interest rates for a given currency and a given duration. It is refreshed by the information service provider and used to calculate the entries in the table `LoanConditon`. This table contains interest and installment for a loan with a given net value, duration, and customer

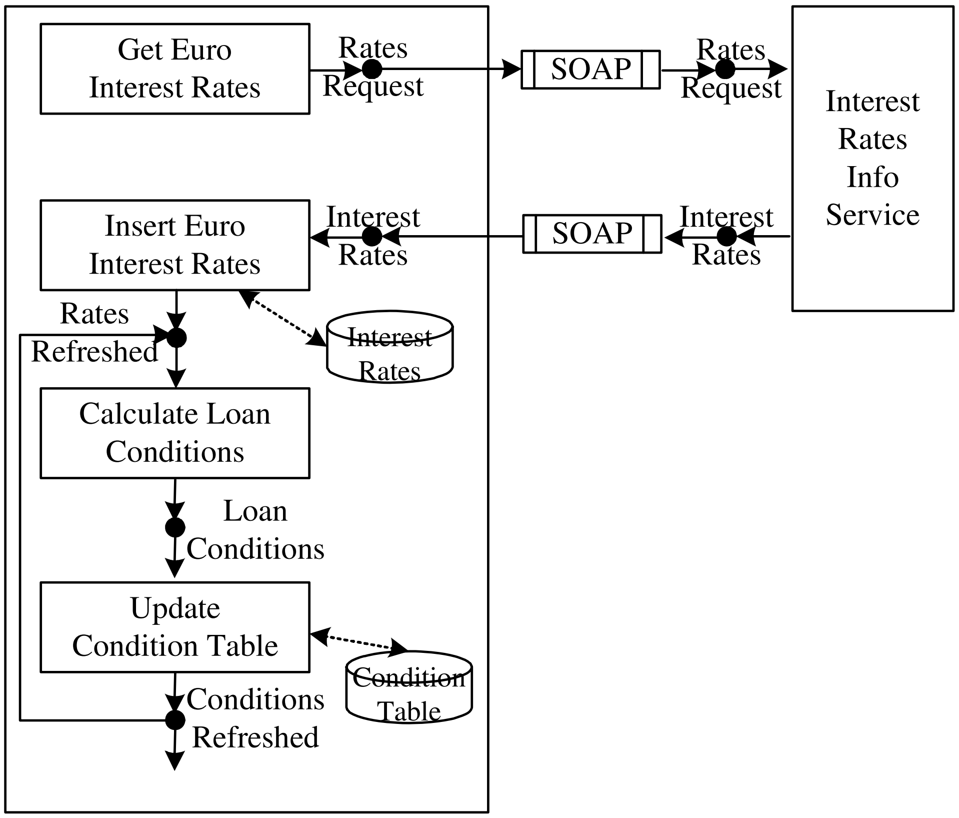


Figure A.15: Calculation of condition table.

scoring. A customer's scoring is determined depending on the date of birth, the marital status, the number of children, the profession, and the monthly income. This and the customer's address data is saved in the `Customer` table. The loan proposal used to draw up the contract is stored in the table `LoanProposal` and consists of the customer ID, the net value, the installment, the interest, and the duration. When the loan is contracted, an account is created in the `Account` table. The loan proposal data is copied except for the duration. This is turned into fixed opening and closing dates.

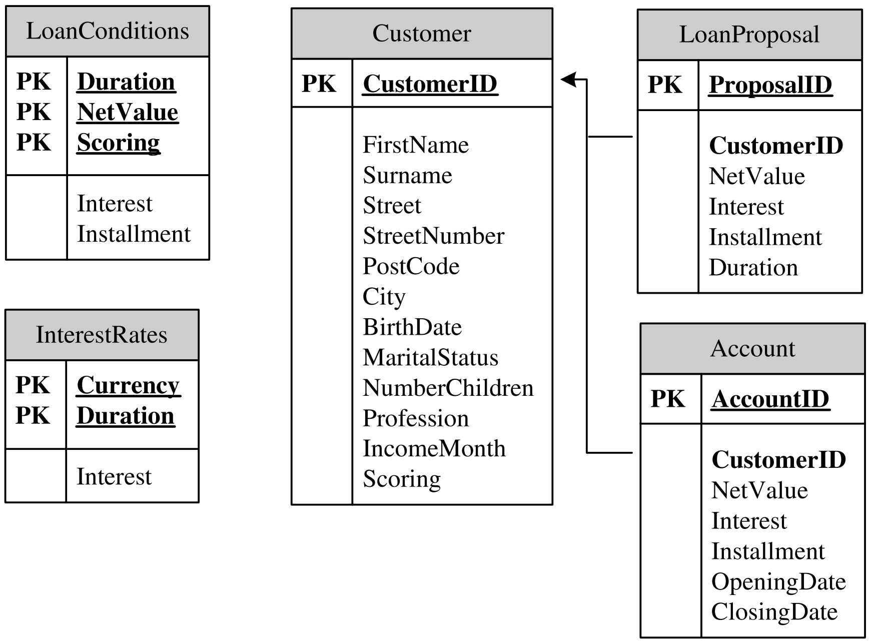


Figure A.16: Loan application data model.

From the analysis of the subprocesses and the data model, the following service profiles can be derived that are to be used within the application:

Euro Interest Rate Service

- Functionality: This service returns the current Euro interest rates. It is an atomic data-centric information service.
- Accessibility: It is called in the loan application administration process via the administrator GUI.
- Information:
  - Operation name: `get_Interest_Rate`

  - Input: Currency according to ISO 4217 Code
  - Output: List of (duration, interest) pairs

- Interaction: The returned data is stored in the table InterestRates.

Condition Look-up Service

- Functionality: This service calculates the sample and the exact condition for the consumer loan, either by look-up or on the fly. It can be regarded as a service of type business function service.

- Accessibility: It is called in the loan application process by the customer for the sample calculation and for the exact calculation. Both calls are executed via the internet front end GUI.

- Information:

  - Operation name: `get_Sample_Condition`
  - Input: net value, duration
  - Output: net value, duration, interest rate, installment
  - Operation name: `get_Exact_Condition`
  - Input: net value, duration, scoring
  - Output: net value, duration, scoring, interest rate, installment

- Interaction: The returned data is displayed on the internet front end GUI.

Score Calculation Service

- Functionality: This service calculates the private customer scoring based on information about private and professional circumstances. It is a business function service.

- Accessibility: It is called in the loan application process by the customer to prepare the exact calculation via the Internet front end GUI.

- Information:

  - Operation name: `calculate_Customer_Scoring`
  - Input: Date of birth, marital status, number of children, profession, income

  - Output: Date of birth, marital status, number of children, profession, income per month, scoring

- Interaction: The returned data is displayed on the internet front end GUI.

Condition Calculation Service

- Functionality: This service refreshes the condition table for consumer loans once a month. It must execute the loan pricing service for all possible combinations of net value, duration, and customer scoring and update the conditions table with the results. Therefore, this service is a process-oriented, composite service as it is based on the loan pricing service and controls its repeated execution.
- Accessibility: It is called in the administration process to refresh the condition table. Further, it is called in the loan application process if the condition look-up service does not return any data because the desired combination of net value, duration and customer scoring is not pre-calculated.
- Information:
  - Operation name: `calculate_Condition`
  - Input: net value, duration, scoring
  - Output: net value, duration, scoring, interest rate, installment
- Interaction: The returned data is stored in the condition table for administrative purposes and displayed for the customer if called via the front end GUI.

Contract Service

- Functionality: This service generates the text of the loan application form using the customer contact data, the loan data, and the scoring data. It is a visualizing service.
- Accessibility: It is called in the loan application process by the customer to finalize the loan contract.
- Information:
  - Operation name: `create_Contract`

  - Input: customer ID, first name, surname, street, street number, post code, city, net value, interest rate, duration, installment
  - Output: contract text as PDF document

- Interaction: The returned data is sent to a printer.

Mailing Service This service is not a Web service, but sending the printed contract to the customer must be executed manually. Therefore, it is out of the scope of the application.

Creating and displaying messages at the customer front end can also be encapsulated into services, but this is omitted in the following. The focus is on calculation intensive functionality, as this is more likely to be reused. The other services for monitoring and administration are not modeled further for this example.

The service profiles resulting from the logical design step have to be matched with existing services that are already implemented to avoid redundant implementation. If the services do not exist, they must be assigned to components that will later realize the services in the implementation. Further, the services must be assigned to the different SOA layers.

Physical Design

In the physical design step, it is decided if the software that realizes a new service will be reused from existing components or services, bought, custom-built, or maybe even outsourced. Further, technical standards for service realization are decided. If there is a SOA strategy already implemented, the designers often do not have a technical choice but have to use predefined standards, such as WSDL and SOAP.

For existing and new services other non-core business functions, such as security, performance, and monitoring, must be chosen and taken into account for physical design. If an ESB is already installed, these non-functional tasks will simply be delegated to the ESB. If an ESB is not used, other physical implementation patterns as described in [GAA^{+}06] will have to be studied and chosen.

Finally, an architecture overview document of the application summarizes the resulting design from all four design phases. Such a document reflects how the new application fits into the existing or aspired overall SOA architecture as depicted in Figure 2.21. It consists of the following aspects as listed in Table A.5.

In the physical design process, the components and services that do not exist yet, but have to be created, are modeled physically to prepare imple-

Table A.5: SOA overview document.

Layer	**Description**
Scope	Which line of business is supported by this architecture?
Operational systems	Packaged applications used by this architecture? Custom applications used by this architecture? Exposure and componentization decisions?
Enterprise components	Which business domain is supported by the component? Which process is supported by the component? Mapping of component to service?
Services	Which services are newly created and exposed? Which existing services are reused? Atomic or composite service realization?
Business process	Which business processes are represented as service choreographies? Which business process is wired into an existing application component? How do services collaborate?
Presentation	Which GUI component uses which service?
Integration	Service level agreements? Quality of service assurance? Security and authentication? Performance restrictions? Technology standards to be used? Monitoring tasks?

mentation. The existing services and components are integrated into the physical design.

At this stage also additional non-functional services are added that are not needed for the business logic but that must be provided by the SOA infrastructure. For example, in the case of the loan application the data between the customer and the application ought to be sent via a secure channel, e.g., using HTTPS via SSL to protect privacy. This is an example of a non-functional service provided by the SOA infrastructure.

A.3 Financial Web Service Ontology

A.3.1 Requirement Analysis and Specification

In the requirement analysis, organizational, procedural, and data aspects of the application are analyzed, which the ontology is designed to support. The ontology is used to annotate Web service interfaces and to discover Web service operations based on operation names, input and output parameters. An application that makes use of these annotations and supports a developer in adding semantic annotations and using them for semantic search could consist of the following components as shown in Figure A.17:

- All interface descriptions with semantic annotations are available to the application (1).
- The taxonomy provides an acknowledged vocabulary for annotations and for searching (2).
- A GUI permits to explore the taxonomy graphically to choose concepts for annotations and for searching (3).
- The development environment permits to annotate operations, input and output with concepts of the taxonomy (4).
- The development environment permits to extend and update the taxonomy (5).
- The development environment permits to search for available operations using concepts of the taxonomy (6).
- The search uses the taxonomy for inferences (7).
- The returned results are ranked according to their similarity with the query and displayed (8).

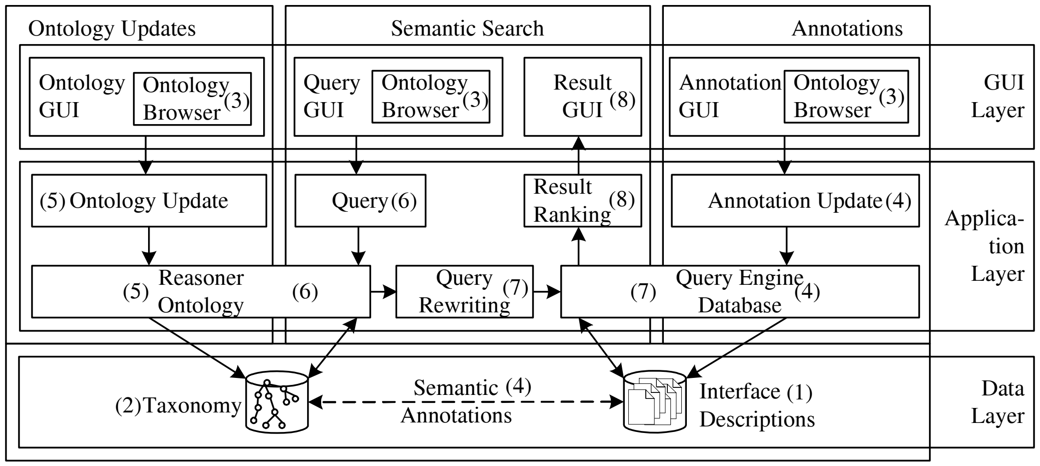


Figure A.17: Ontology-based search.

In the organizational analysis, stakeholders, potential users, and potential systems that will benefit from an ontology and semantic annotations of Web services are identified. This is summarized in Table A.6.

Table A.6: Target groups and systems.

Target Group	**Usage Description**
O1: Developers	Software developers are system designers or programmers. They want to use an ontology of Web services to optimize software re-use, especially for computation-intensive mathematical routines.
System	**Usage Description**
O2: Service-oriented Development Suite	A service-oriented development tool suite supports developers in building service-oriented applications. It can use ontologies to provide a graphical interface for Web service detection during development for search and annotation.

In the process analysis, the processes that the ontology is to support or is involved in are described as shown in Table A.7.

In the data analysis, available data sources for the conceptual design, e.g., related ontologies are identified. The result for the application scenario is briefly summarized in Table A.8.

After the requirement analysis is completed, a more detailed requirement specification is executed. This step aims at deriving an integrated view of the different analysis perspectives, the compilation of a glossary through

Table A.7: Processes and system functions.

Process	**Process Description**
P1: Service Search	The search for services is executed based on operation names and refined through input and output. This is to be supported by the ontology.
P2: Alternative Search	If the search does not return an exact match, the search for an alternative operation that supports input or output transformation is to be enhanced by the ontology.
System Function	**System Function Description**
P3: Consistency Check	If a developer assigns an existing operation concept from the ontology to a Web service operation, it can be checked if the concept is an operation concept or an attribute concept in the given semantic model.
P4: Registering Concepts	If a developer has implemented a new operation, which does not exist in the operation ontology, the ontology must be extended.

Table A.8: Data sources.

Text Document	**Description**
D1: Software Documentation	The conceptual software design of the financial calculation kernel contains the description of all implemented computation routines.
Ontologies	**Description**
D2: Product Taxonomy	For internal reporting purposes a taxonomy of financial products exists. The financial calculation kernel is able to calculate them.
D3: Measure Taxonomy	For internal controlling purposes a taxonomy of financial measures exists. The financial calculation kernel is able to compute them.

stepwise refinement of relevant terms, and a quality evaluation of the result. The integration of different use cases for the ontology is documented with a use case diagram in Figure A.18 and additional competence questions in Table A.9. Finally, a glossary of terms is derived.

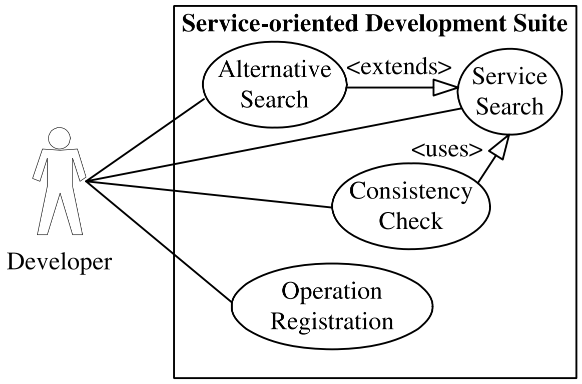


Figure A.18: Integrated use case diagram.

Table A.9: Competence questions.

Question	Description	Data Source
K1	Which Web services implement the chosen operation concepts?	
K2 6	Is there an attribute concept that matches the chosen input and output?	D2, D3
K3	Does the service implementation represent a valid operation concept?	
K4	Does an operation concept already exist?	

A.3.2 Conceptual Design

The conceptual design of the Web service ontology is also aligned with the approach in [Hüs05]. First, concept hierarchies are modeled as generalizations and aggregations using the notation introduced in Chapter 4. The concept hierarchies cover the aspects: financial products, financial measures, customers, cashflows, simulation, calculation of cashflows, and calculation of measures as indicated in Figure A.19. Simulations and calculations are operation concepts, whereas the other concepts represent attribute concepts.

In many banks a product hierarchy exists, e.g., in the data warehouse product dimension or in report definitions. The concept hierarchy for the

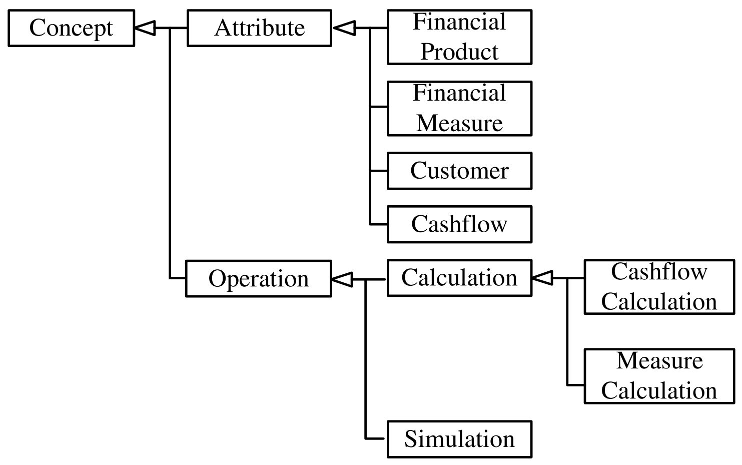


Figure A.19: General concepts for concept hierarchy.

given example is refined in Figure A.20. The concepts in grey shades have been used in the example in Section A.1. The different concepts are explained in the glossary given in Table A.10.

The second important concept hierarchy, which is used to build the Web service ontology, is the measure hierarchy that defines financial measures, e.g., for reporting purposes. This concept hierarchy is depicted in Figure A.21. The glossary explaining this ontology is given in Table A.11.

The customer concept hierarchy can also be derived from existing documents such as the customer data warehouse dimension or reporting categories for customer reporting. In the example scenario, only the customer rating is relevant for loan calculations. Therefore, only a small portion of the customer concept hierarchy is modeled here as shown in Figure A.22. The concepts are explained in Table A.12.

There is also a conceptual hierarchy for the different types of cashflows that are the basis for the calculation of different measures. This is depicted in Figure A.23 and explained in Table A.13.

Finally, the different financial calculations are modeled as concept hierarchies. As the calculation result is regarded as most important feature of each computation, the concept hierarchy is refined following the result hierarchy. The cashflow calculations are shown in Figure A.24. The measure calculations are shown in Figure A.25. Both concept hierarchies are restricted to the calculations as relevant for the application scenario.

Note that there are not only calculations for measures, but also for other loan related concepts such as the total debt or the net value. As the result concepts have already been explained in glossaries, an additional glossary for

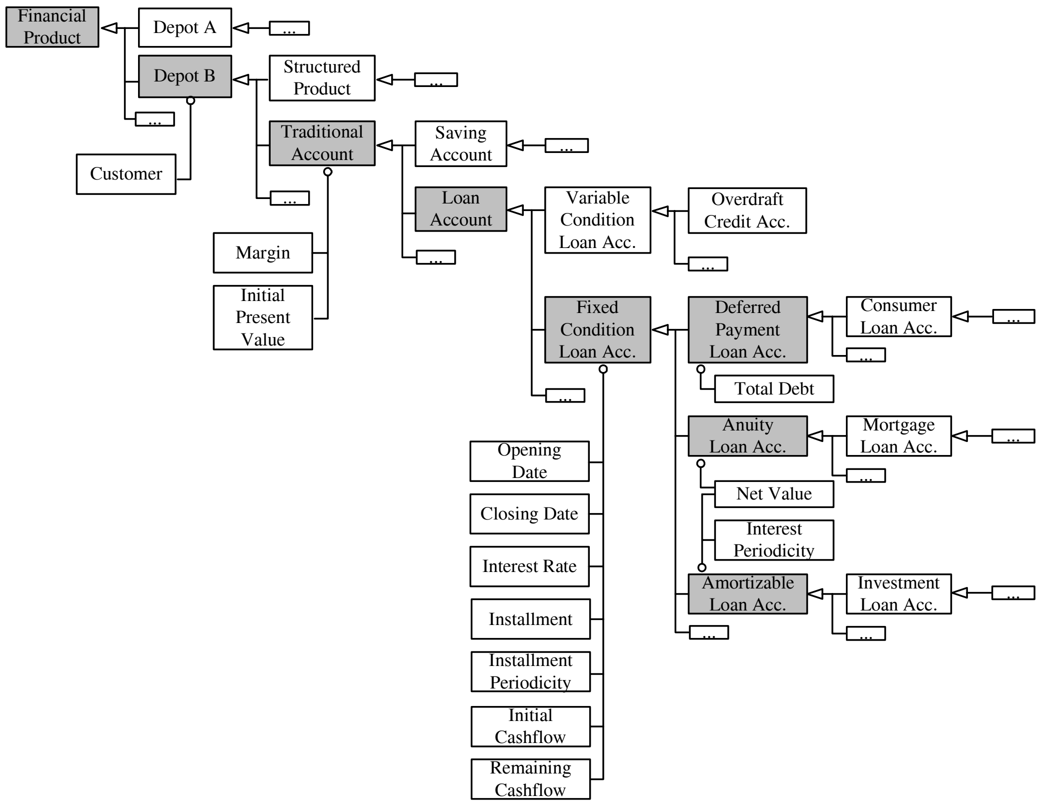


Figure A.20: Financial products.

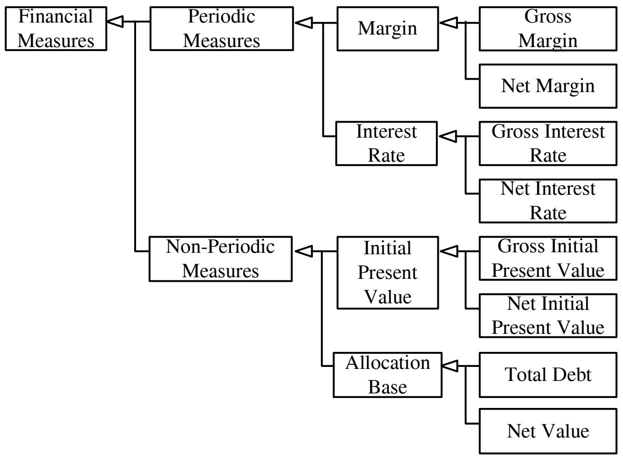


Figure A.21: Financial measures for reporting.

Table A.10: Financial products glossary.

Concept	**Description**	**Context**	**General Concept**
Financial Product	Bank product	Financial Product	Concept
Depot A	Bank product traded between banks	Financial Product	Financial Product
Depot B	Bank product sold from bank to customer	Financial Product	Financial Product
Structured Product	Combination of depot B products	Financial Product	Depot B Depot B
Traditional Account	Interest-based depot B product	Financial Product	Depot B Depot B
Saving Account	Account to deposit money at the bank	Financial Product	Traditional Product
Loan Account	Account to let money to the customer	Financial Product	Traditional Product
Variable Cond. Loan	Loan with uncertain interest and duration	Financial Product	Loan
Overdraft Credit	Loan granted on debit account	Financial Product	Variable Cond. Loan
Fixed Cond. Loan	Loan with fixed interest and duration	Financial Product	Loan
Deferred Payment Loan	Total debt including interest is payed back in installments	Financial Product	Fixed Condition Loan
Consumer Loan	Deferred payment loan to buy consumer goods	Financial Product	Deferred Payment
Annuity Loan	Net Value payed back in installments which include interest	Financial Product	Fixed Condition Loan
Mortgage Loan	Annuity loan to to buy property	Financial Product	Annuity Loan
Amortizable Loan	Net Value payed back in installments, interest payed independently	Financial Product	Fixed Condition Loan
Investment Loan	Amortizable loan to make investments	Financial Product	Amortizable Loan

Table A.11: Measure glossary.

Concept	**Description**	**Context**	**General Concept**
Financial Measure	Measure for financial product	Financial Measure	Concept
Periodic Measure	Time-based financial measure	Financial Measure	Financial Measure
Non-periodic Measure	Time-independent financial measure	Financial Measure	Financial Measure
Margin	Revenue; difference in customer and bank interest rate	Financial Measure	Periodic Measure
Interest Rate	Customer fee, charged for loan	Financial Measure	Periodic Measure
Initial Present Value	Present value of initial customer cashflow	Financial Measure	Non-periodic Measure
Allocation Base	Amount, reference for periodic measure	Financial Measure	Non-periodic Measure
Gross Margin	Margin without additional costs	Financial Measure	Margin
Net Margin	Risk-adjusted margin	Financial Measure	Margin
Gross Interest Rate	Interest without additional costs	Financial Measure	Interest Rate
Net Interest Rate	Interest incl. additional costs	Financial Measure	Interest Rate
Gross Initial Present Value	Value of loan compared to refinancing loan	Financial Measure	Initial Present Value
Net Initial Present Value	Risk-adjusted initial present value	Financial Measure	Initial Present Value
Total Debt	Sum of net value, interest, and additional costs	Financial Measure	Allocation Base
Net Value	Sum granted to customer with a loan	Financial Measure	Allocation Base

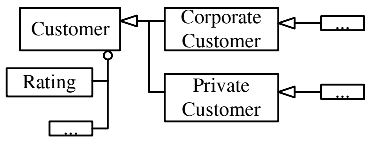


Figure A.22: Customer concept hierarchy.

Concept	**Description**	**Context**	**General Concept**
Customer	Client	Customer	Concept
Private Customer	Customer is an individual	Customer	Customer
Corporate Customer	Customer is an enterprise	Customer	Customer
Rating	Score for customer credit worthiness	Customer	Customer

Table A.12: Customer glossary.

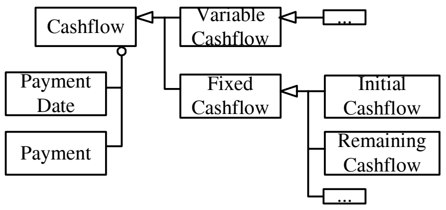


Figure A.23: Cashflow concept hierarchy.

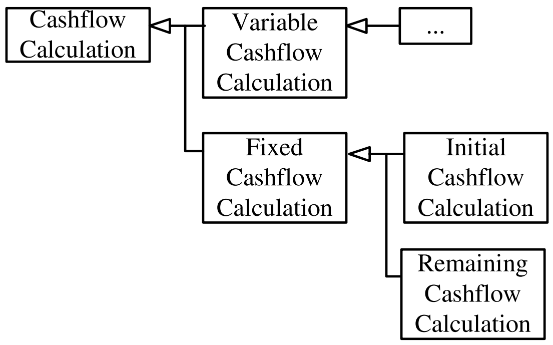

Cashflow Calculation
Variable Cashflow Calculation
...
Fixed Cashflow Calculation
Initial Cashflow Calculation
Remaining Cashflow Calculation

Figure A.24: Cashflow calculation concept hierarchy.

Concept	**Description**	**Context**	**General Concept**
Cashflow	Sequence of payments	Cashflow	Concept
Variable Cashflow	Cashflow with uncertain payments	Cashflow	Cashflow
Fixed Cashflow	Cashflow with fixed payments	Cashflow	Cashflow
Initial Cashflow	Cashflow at contracting time	Cashflow	Fixed Cashflow
Remaining Cashflow	Cashflow during contract duration	Cashflow	Fixed Cashflow
Payment Date	Date of a payment in a cashflow	Cashflow	Cashflow
Payment	Amount of money payed in a cashflow	Cashflow	Cashflow

Table A.13: Cashflow glossary.

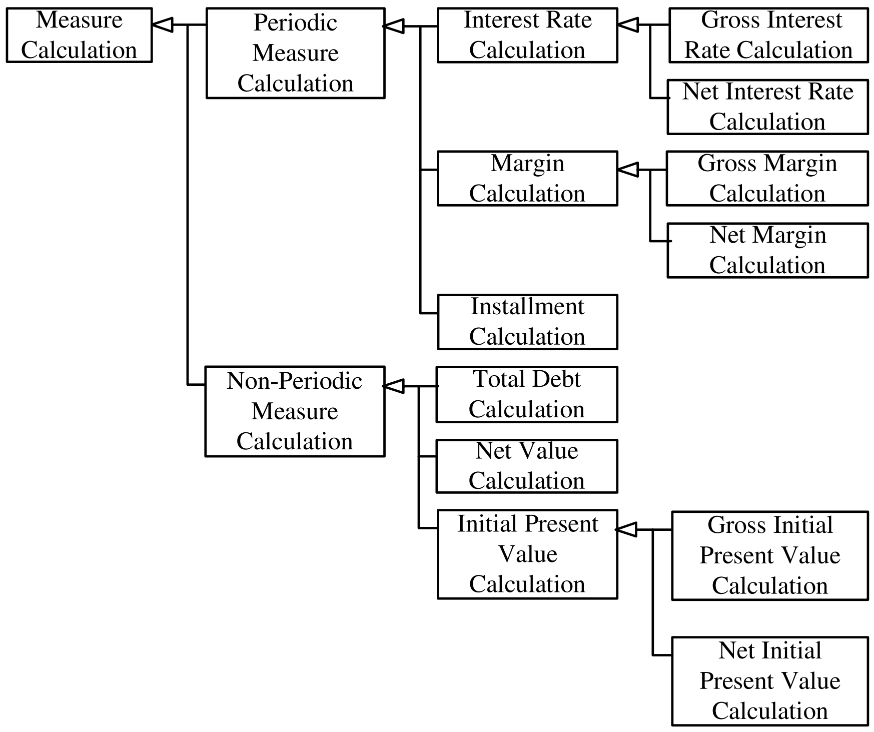

Measure Calculation
Periodic Measure Calculation
Interest Rate Calculation
Gross Interest Rate Calculation
Net Interest Rate Calculation
Margin Calculation
Gross Margin Calculation
Net Margin Calculation
Installment Calculation
Non-Periodic Measure Calculation
Total Debt Calculation
Net Value Calculation
Initial Present Value Calculation
Gross Initial Present Value Calculation
Net Initial Present Value Calculation

Figure A.25: Loan calculation concept hierarchy.

the operation concept hierarchies is omitted.

Following the modeling approach of [Hüs05], the next step assigns roles to concepts. A concept is either a class or a property or both. The analysis starts with the aggregation hierarchies of the concept model and then continues with the generalization hierarchies. The intermediate steps of this approach are omitted. Only the final result of the role assignment is presented. Figure A.26 shows the role assignment for financial products. Classes are denoted with a "C", properties with a "P". Concepts can appear in both roles, as classes and as properties of other classes.

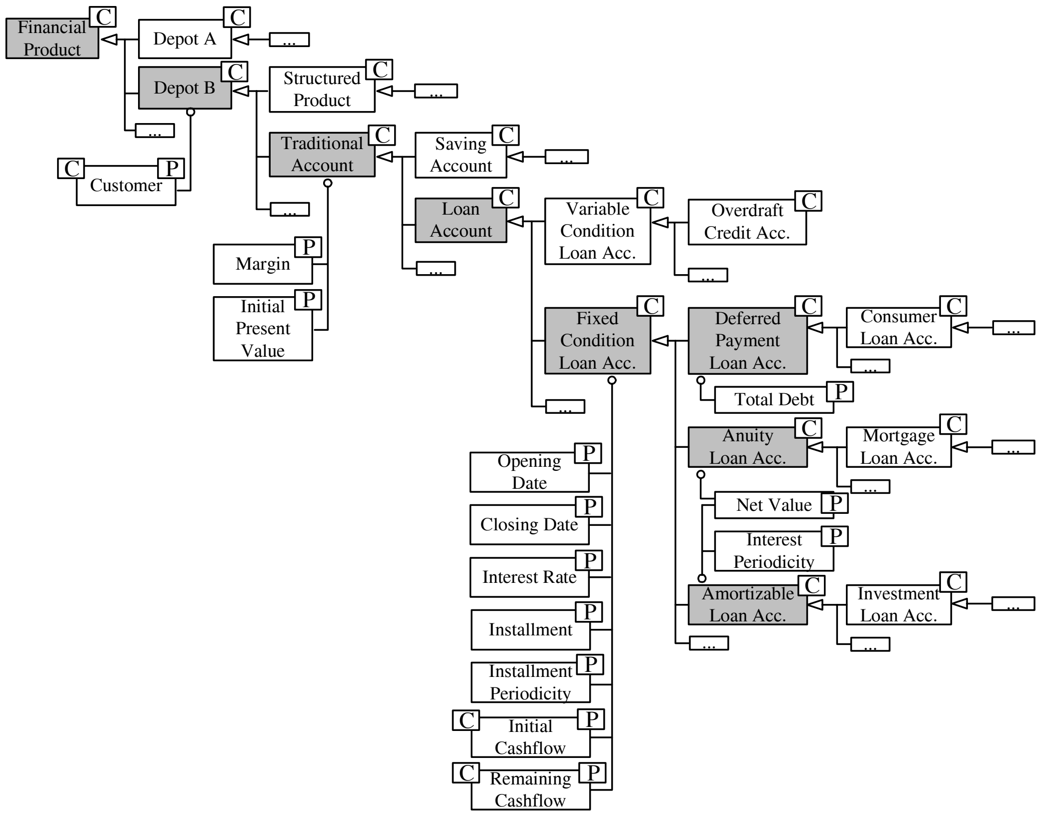


Figure A.26: Role assignment for financial products.

As financial products have measures as properties, the concept hierarchy of financial measures is turned into a property hierarchy as shown in Figure A.27.

The role assignment for customers and cashflows is straight forward as shown in Figures A.28 and A.29.

All calculation concepts are classes. Therefore, the figure showing this assignment is omitted. The calculation concepts represent a class hierarchy

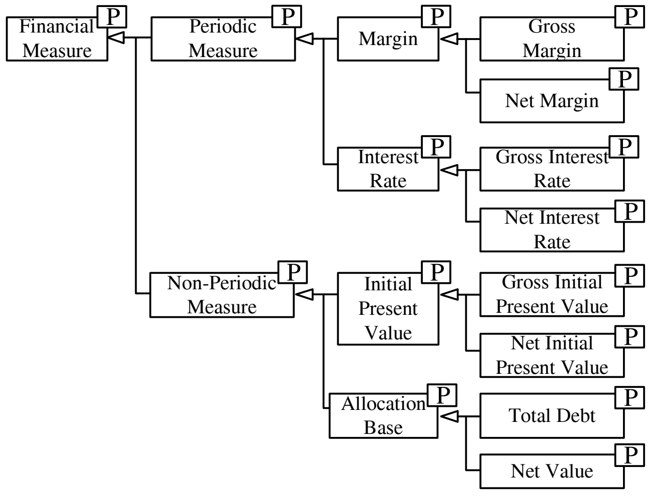


Figure A.27: Role assignment for financial measures.

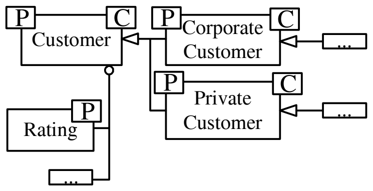


Figure A.28: Role assignment for customers.

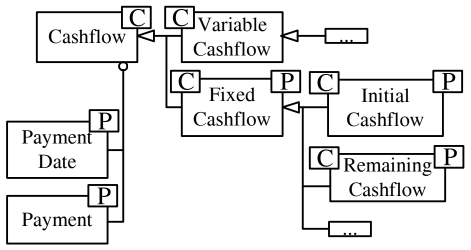


Figure A.29: Role assignment for cashflows.

independent of the other ontological classes and properties.

After these preparation steps a class hierarchy (see Figure A.30) and a property hierarchy (see Figure A.31) is derived. Note that the property hierarchy uses additional concepts from top-level ontologies, e.g., time instant. Top-level ontologies for time, currencies, or money in general are not shown here.

The conceptual model can be translated in a logical model such as given by RDFS and OWL. This is explained in [Hüs05] and not shown here because the conceptual model is already sufficient to analyze the search support that ontological references for Web service operations offer.

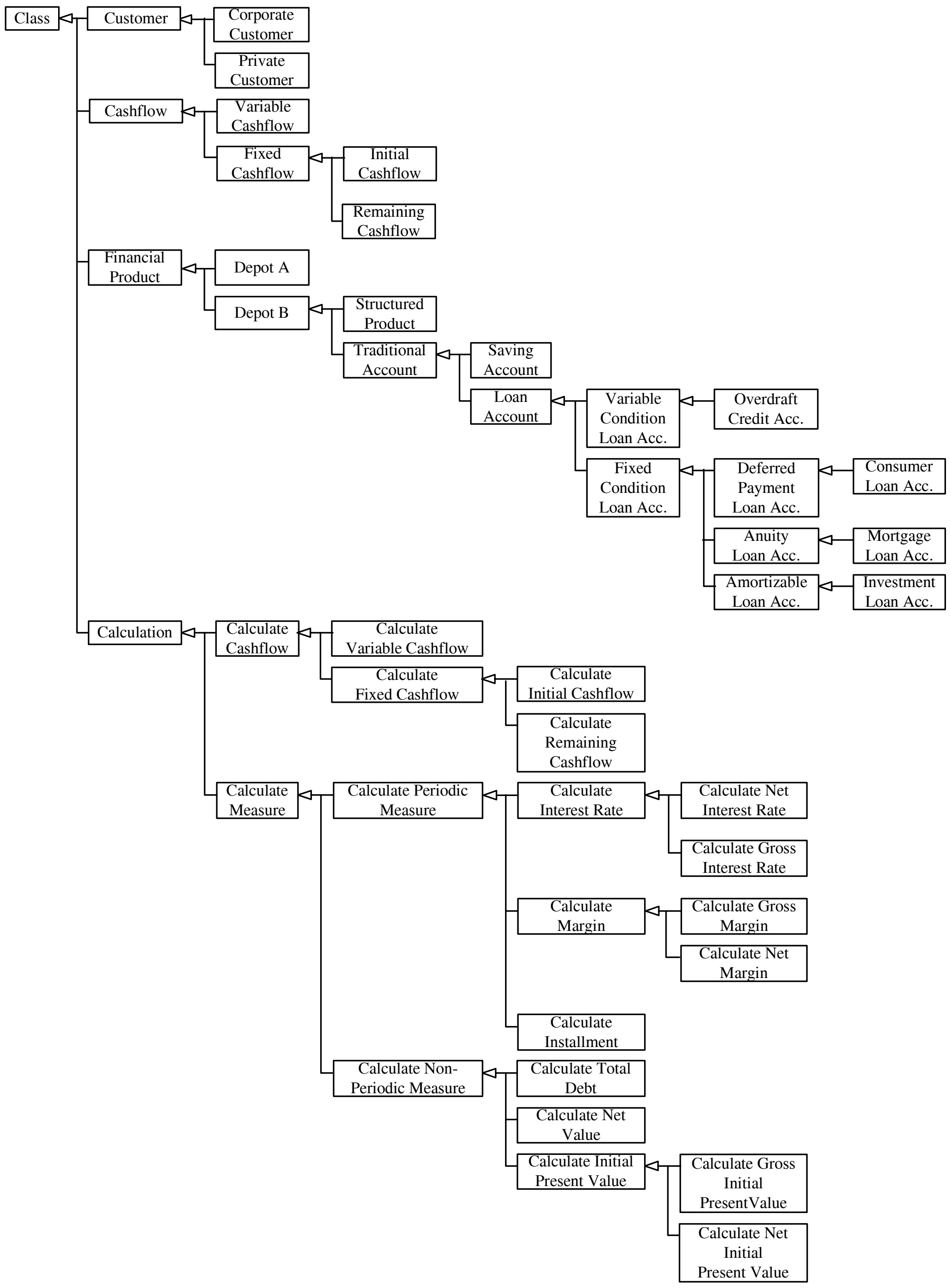


Figure A.30: Class hierarchy.

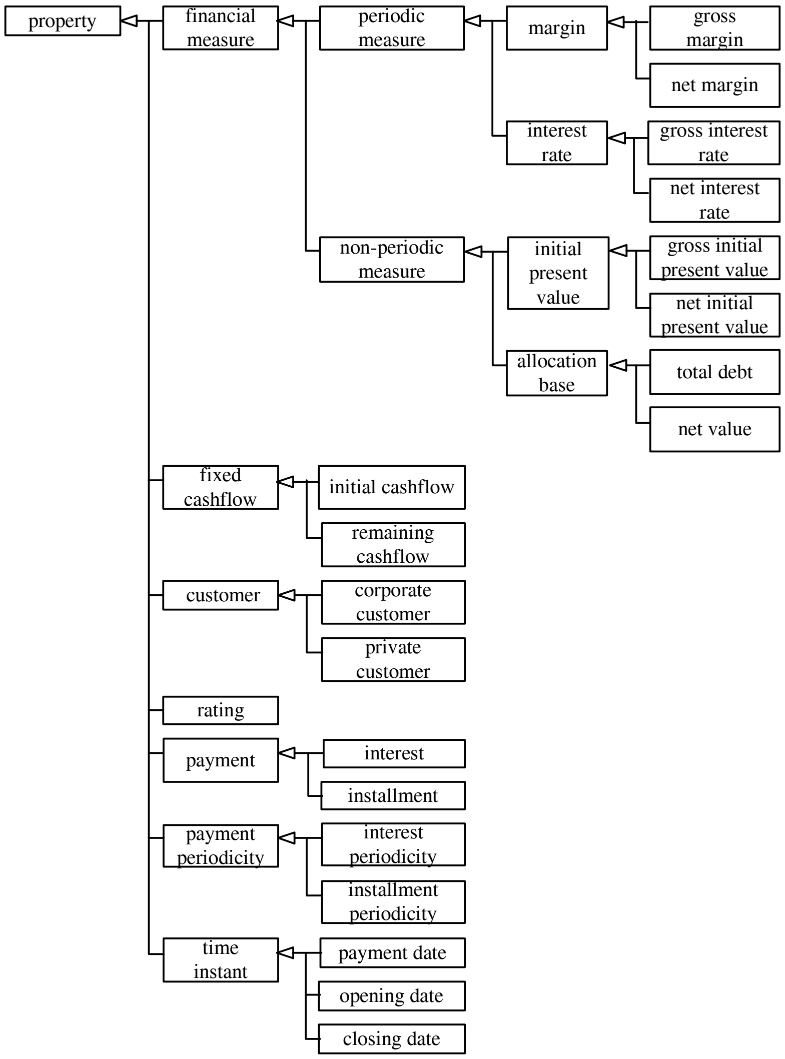


Figure A.31: Property hierarchy