Technische Universität Hamburg-Harburg

Project Work

User-Centered Web Service Discovery Support

Submitted by:

Lidia Khmylko

Supervised by:

Prof. Dr. Joachim Schmidt

Dipl.-Inf. Patrick Hupe

Software, Technology and Systems Institute

Hamburg University of Technology

GERMANY

June 2006

I declare that this work has been prepared by myself, all literary or content-related quotations from other sources are clearly pointed out, and no other sources or aids than the ones that are declared are used.

Hamburg, 29.06.2006

Lidia Khmylko

Acknowledgments

My special thanks go to Prof. Dr. Joachim W. Schmidt providing the opportunity to work on this student project, to the research assistant Dipl.-Inf. Patrick Hupe for his essential and kind advice, guidance and encouragement throughout this student project work, to Prof. Dr. Ralf Möller for his advice and new suggestions, and all the research assistants at the Software, Technology and Systems Institute (STS) of the Hamburg University of Technology (TUHH) for their help and assistance with ontology links and query languages.

Table of Contents

Chapter 1 Introduction

Web services are Internet-based, distributed modular service abstractions which provide standard interfaces and communication protocols for efficient and effective service integration. Web services were invented to bring a new level of integration to the computing industry and its networked communities. The main advantage aimed at by Web services is that they enable service-based applications to interoperate despite being developed in different programming languages, at various times, by different people, with designs based on various assumptions.

Very strong initial success of Web services was mostly perceived in the area of integration within, and (to a lesser extent) between, businesses. An increasing number of organizations are using the Web services technology as a standardized infrastructure for interoperation of disparate software components within the organization, fulfillment of transactions between organizations, and sharing of corporate resources with customers and partners. Not only the automatic inter- and intraenterprise integration processes, but also human users may become Web services clients, thus, bringing forth the problem of understanding between humans and machines.

1.1 Motivation

Second-generation Web services specifications under development, such as ebXML [Ewe06] and BPEL4WS, [ACD03], should enhance the usability, scope, and expressiveness of Web services. However, there is an increasing realization that the syntactic level they treat Web services on is not enough and technologies from the Semantic Web [BHL01] can also make crucial contributions to Web services frameworks. Semantic Web Services [MSZ01] take up on this idea, introducing ontologies to describe, on the one hand, the concepts in the services' domains (e.g., flights and hotels, tourism, libraries), and, on the other hand, characteristics of the services themselves (e.g., control flow, data flow) and their relationships to the domain ontologies (via inputs and outputs, preconditions and effects, and so on). These semantically rich descriptions enable automated machine reasoning over service and domain descriptions, thus supporting automation of service discovery, composition, and execution and reducing manual configuration and programming efforts. The field of Semantic Web services is still in an early stage, and adoption has been slow.

Thus, the main disadvantage of the actual description standards for services including Web services is that they predominantly deal with the service description at a syntactical level (i.e., how a service should be called) and lack the formalization of semantics (what the service is responsible for) which is crucial for Web service human users.

One of the key problems associated with Web services is service discovery. Before any Web service can be accessed, it should be made known to the party that wants to use it. The problem field can be differentiated into

a) computer-supported matching of the service providers with service requests (service matchmaking) which is widely researched in Semantic Web community and
b) support of the human users in their search and discovery of services.

This work makes a contribution to the latter point. The goals and objectives for it will be set in the next section.

1.2 Objectives

The goal of this project work is a conceptual design of an architecture providing services in heterogeneous service infrastructures to support human users of the system.

This work will focus on services in digital library environments. As a case study, a digital library system for teachers (Hamburger Medienindex[1], HMI) will be used. To achieve the goal set the following tasks will be performed:

- the study of the state-of-the-art service-oriented architectures, their principles, related technologies and standards as well as of the higher abstract description languages for services (e.g., OWL-S) and service discovery approaches;
- the analysis of service discovery in library environments including the analysis of the HMI platform specifics and the requirements added to the services layer based on the users' needs in service discovery;
- the conceptual design of the architecture that will include the following steps:
  - presenting the technical model of the system as a domain-specific ontology;
  - modeling the services, their structures and operational sequences accessible from the outside;
  - designing the query modules that supports service discovery by human users;
- the evaluation of the designed architecture, its effect, usefulness, and enhancement possibilities.

The envisioned conceptual architecture will provide library users with a semantic service discovery support.

1.3 Structure of This Work

The background technologies that will serve as a basis for the conceptual design later are analyzed in Chapter 2. It centers around the discussion of service-oriented architectures (Section 2.1) and their current implementation with Web services (Section 2.2) as well as general Semantic Web concepts and standards (Section 2.3). The proposed standards for the semantic description of Web services are compared in detail in Section 2.4 and the existing approaches for the semantic Web service discovery are introduced and analyzed in Section 2.5.

1 http://www.sts.tu-harburg.de/projects/entry.html#HMI

Chapter 3 analyzes the requirements for the conceptual architecture of the user-centered service discovery support in the HMI library. It introduces the HMI architecture (Section 3.1) and by means of the use case analysis (Section 3.2.1) and domain analysis (Section 3.3) defines the guidelines for the conceptual design taking place in Chapter 4. The conceptual architecture designed is evaluated in Section 4.3 and the work finishes with a summary and outlook.

Chapter 2 Conceptual and Technological Background

Service-oriented architecture (SOA) has emerged as the most significant shift in how business applications are designed, developed and implemented in the last 10 years, eclipsing the shift to client-server. In fact, Gartner, Inc. predicts that by 2008, "SOA will provide the basis for 80 percent of new development projects" [Hay05].

SOA is an architectural style which aims to allow interaction of diverse applications regardless of their platform, implementation languages and locations by utilizing generic and reliable services that can be used as application building block. SOA includes methodologies and strategies to follow in order to develop sophisticated applications and information systems.

2.1 Principles of Service-Oriented Architectures

Before speaking about SOA, one needs to consider the software development background for its appearance. Over the last decades, object-oriented (OO) and component-oriented architectural styles have firmly established themselves in all kinds of software projects which is surely a good argument in their favor and SOA does not put an end to any of the previous technologies. But the wide acceptation of object and component orientation has also revealed their shortcomings. That is why, a brief revision of both architectural styles is a good reference point to start speaking about SOA.

Object-oriented development supports the development of software by encapsulating both data and behavior into abstract data types, called *classes* [Boo97]. Instances of classes are formed into small modules, called *objects*. Any changes in data representation only affect the immediate object that encapsulates the data. Classes can live for ever, while objects have a limited lifetime.

Objects communicate with each other through messaging. Object based development advances software design by providing more support for hiding behavior and data through objects and classes. There is almost no dependency between objects, however a large number of interconnected objects create dependencies that can be difficult to manage.

The well-known principles of the object-oriented development are modularity, encapsulation, separation of the interface and implementation, information hiding, and polymorphism [Boo97].

Components are more sophisticated software modules than objects. A software *component* is defined as a functional unit with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [Szy02]. It can be realized as a group of objects which has a specified interface that work together to provide an application function.

The term component may refer to many different software constructs, from single application logic to an entire functional system. In all cases, a component is a software package with one or more well defined interfaces. A component is executed in a component execution environment provided by an application server, such as a J2EE container, which provides the functions required by the component for execution in the environment, such as transaction management and database connection pooling.

Components overlap the properties of object orientation, such as encapsulation and polymorphism, except it reduces the property of inheritance. In component thinking, inheritance is tightly coupled and unsuitable for most forms of packaging and reuse. Instead, components reuse the functionality by invoking other objects and components rather than inheriting from them. In component terminology, these invocations are called delegations [MaM04].

Component specifications, i.e., their public interfaces, can be reused. The reuse of component specifications is a form of polymorphism. Preferably, component specifications are local or global standards that are widely reused throughout a system, an enterprise, or an industry. Components may be integrated to create a larger entity which could be a new component, a component framework, or an entire system. This is called composition [MaM04].

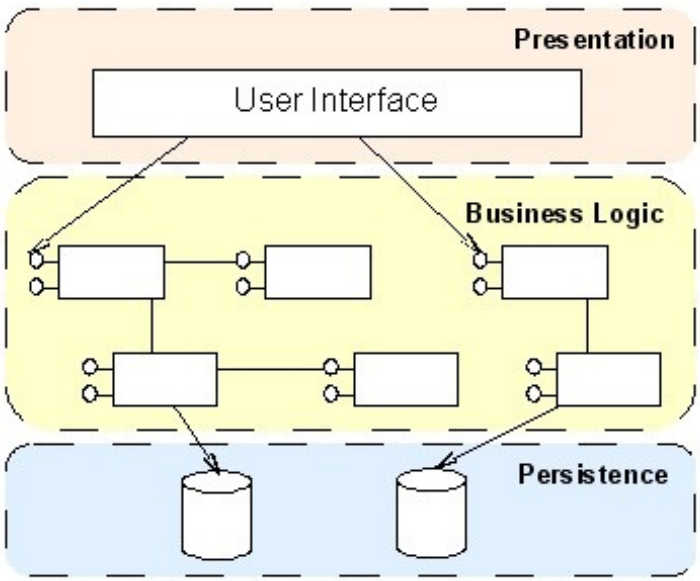

Figure 2.1: A typical 3-tier application architecture (cf. [Ire03])

Reusable components are good reflections of effective software design. The architecture provides the design context in which the components are built and reused. Another important aspect of components is that the development of software architecture based on component specifications supports parallel and independent building of the system parts. These computational boundaries that define an individual system part are testable subsystems and can be divided for one or more distributed project teams. A good architecture emphasizes the separation of responsibilities. In a common three-tier architecture (Figure 2.1), the presentation tier manages presentation components; the business objects tier manages business logic components; and the persistence tier manages data access components.

This separation and modularization provides for fault tolerance, easier maintenance, and future-proofing. A good service-oriented architecture is nothing new, just a smart way of separating (and exposing) a component's responsibilities.

Similar to objects and components, a service is an architectural building block. It comprises information and behavior, hides the internal implementation from outside and can be described by its relatively simple interface that can be remotely called. The World Wide Web Consortium[2], *W3C*, defines:

> A *service* is an abstract resource that represents a capability of performing tasks that represents a coherent functionality.

Services have been subject to research in business science long before the Internet hype came along and the term came to the computer science. Service has become a term loaded with different meanings at different circumstances, depending mostly on the authors' research domain. Researchers in business schools, for example, have been investigating the nature of services in the sense of business transactions for decades. They traditionally consider them to be business activities, deeds and performances of a mostly intangible nature [BGO04].

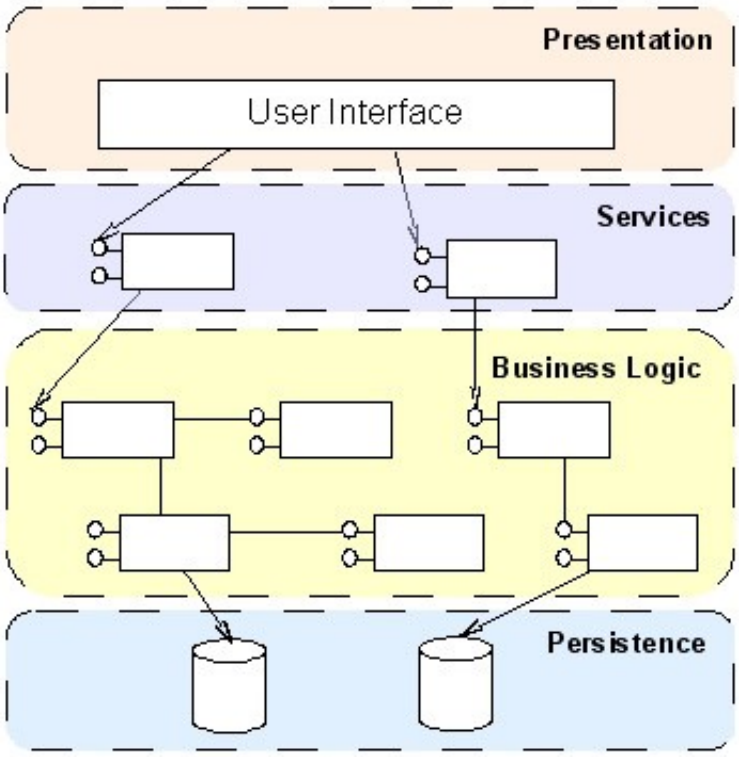

Figure 2.2: A service-oriented application architecture (cf. [Ire03])

A service-oriented architecture consists of a number of services. Services are loosely-coupled pieces of functionality that have well-defined, platform-independent interfaces and can be reused. SOA and Web services are often mentioned in the same context, and this leading to a misunderstanding. So it should be specially pointed out that SOA is as an architectural paradigm independent of Web services and can be realized basing on different technologies. The OASIS SOA Reference Model Technical Committee[3] is working on defining SOA independent of any specific technologies. Another misunderstanding is caused according to [Kay03] by the relationship between the object-oriented technology and SOA. Contrary to the wide-spread opinion, the object-oriented

2 http://www.w3.org/
3 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

technologies are not replaced by SOA. The application development is still based on the classic OO principles mentioned above. Web services or other SOA implementations can be set up over the object-oriented or other technologies. The basis technology used is hidden and is therefore irrelevant (Figure 2.2).

In a service-oriented architecture, clients are consuming services, rather than invoking discreet method calls directly. In a 3-tier model, objects are marshaled across process boundaries through the proxy/stub techniques. This provides benefits, such as location transparency. The basic philosophy is that one tier should only communicate with the tier contiguous to it.

One disadvantage to object-orientation at an architectural level is the number of communication links. Client code is responsible for traversing complex object models and understanding details about domain-specific logic. In a service-oriented model, a further "layer of indirection" is introduced. This alleviates some of the pain associated with traversing complex object models. The services layer, the layer added between the presentation layer and the business logic layer in Figure 2.3, provides black-box functionality.

In a service-oriented design, services should be course-grained. Course-grained services are modeled after and align to business processes. Objects should be fine-grained and align to real business entities. These discreet objects provide the detailed business logic. Specificity is good when building discreet business objects. This is also a very successful way to codify organizational knowledge. Each business object is responsible for its own behavior and business rule implementation, such as updating a database table, sending an email, or placing a message on a queue.

Services provide the orchestration of the detailed business objects to expose a full service to the consumer. Services are responsible for orchestrating calls to discreet business objects, managing the responses, and acting accordingly. Service methods may invoke and manage several business objects. Service methods align to business processes by design. Class methods align to detailed object-level operations by design.

Consider the example architecture presented in Figure 2.3. A single consumer application (perhaps interacting with the company over the Internet) wants to engage the company in some business process. To facilitate that business process, the company internally invokes processing that spans two discreet systems (A and B). However, through a service-oriented architecture, the entire end-to-end business process is exposed to the consumer application as a single service.

Figure 2.3 also illustrates the granularity differences on the various layers. The coarse-grained services that can be accessed from the outside are based on the finer grained services of the internal services layer, and further on the fine-grained objects and database calls reflecting the grade of detail necessary for the users of the corresponding layers.

Still there is a debate in the SOA community concerning how fine or coarse the services should be. While there's no standard way to quantify service granularity, some ideas from component-based design can be used, such as the number of function points or data elements affected by the invocation of a service. If a service needs to be called too many times in a business application or if only a small part of its functionality is typically used, it's likely that the service is too coarse. If too many parameters for a service are required, the service is most likely too low level and fine grained. Striving for an appropriate granularity will maximize ease of use, reuse, and manageability; an appropriate service is not necessarily either fine or coarse, but one that maximizes business value [DHK05].

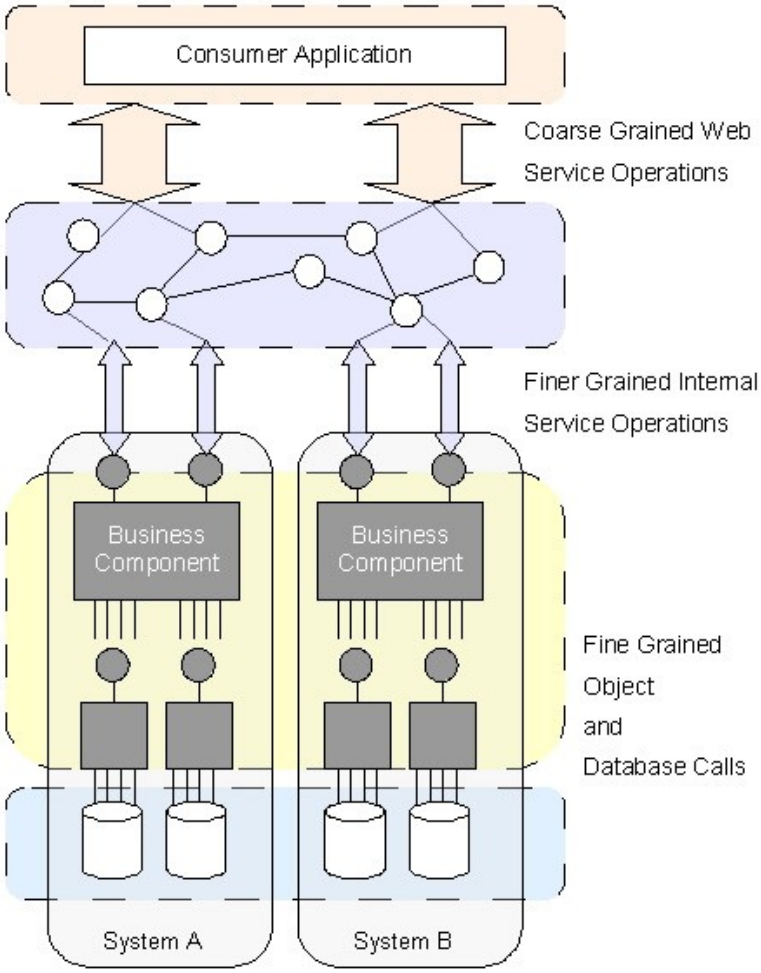

Figure 2.3: Exposing separate applications as a single service (cf. [Ire03])

As already mentioned the service-oriented architecture came to leverage the advantages of object-oriented architectures, but also to overcome its disadvantages. The disadvantages that stay in the way of many object-oriented projects are:

- a very high complexity,
- software chunks with a very high coupling arise, as all the functionality should be wrapped inside objects,
- low separation of concerns, as the same object model is generally used in both the Business Tier and the Client Tier),
- the business management principle to separate the business processes and data cannot be met.

In addition to overcoming the disadvantages of object-oriented approach mentioned in the beginning of the chapter, SOA aims at achieving a challenging set of advantages that includes [SiH05]:

- **Intra-enterprise interoperation**: to provide the tools modeling information and relating the models, constructing processes over the systems, asserting and guaranteeing transactional properties, adding in flexible decision-support, and relating the functioning of the component software systems to the organizations they represent.
- **Inter-enterprise interoperation:** to provide the same benefits as for intra-enterprise interoperation above. In addition, it provides the ability for the interacting parties to

choreograph their behaviors so that each may apply its local policies autonomously and yet achieve effective and coherent cross-enterprise processes.

- **Application configuration:** to enable the customization of new applications by providing a Web service interface that eliminates messaging problems and by providing a semantic basis to customize the functioning of the application.
- **Dynamic selection:** to enable dynamic selection of business partners based on quality-of-service criteria that each party can customize for itself.
- **Software fault tolerance:** to provide support for dynamic selection of partners as well as abstractions through which the state of a business transaction can be captured and flexibly manipulated; in this way, dynamic selection is exploited to yield application-level fault tolerance.
- **Grid computing:** to enable the efficient usage of Grid resources.
- **Utility computing:** to facilitate utility computing, especially where redundant services can be used to achieve fault tolerance.

To realize the above advantages, SOAs impose the following requirements [SiH05]:

- **Loose coupling.** No tight transactional properties generally apply among the components. In general, it is not appropriate to specify the consistency of data across the information resources that are parts of the various components. However, it is reasonable to think of the high-level contractual relationships through which the interactions among the components are specified.
- **Implementation neutrality.** The interface is what matters. We cannot depend on the details of the implementations of the interacting components. In particular, the approach cannot be specific to a set of programming languages.
- **Flexible configurability.** The system is configured late and flexibly. In other words, the different components are bound to each other late in the process. The configuration can change dynamically.
- **Long lifetime.** We do not necessarily advocate a long lifetime for our components. However, since we are dealing with computations among autonomous heterogeneous parties in dynamic environments, we must always be able to handle exceptions. This means that the components must exist long enough to be able to detect any relevant exceptions, to take corrective action, and to respond to the corrective actions taken by others. Components must exist long enough to be discovered, to be relied upon, and to engender trust in their behavior.
- **Granularity.** The participants in an SOA should be understood at a coarse granularity. That is, instead of modeling actions and interactions at a detailed level, it would be better to capture the essential high-level qualities that are (or should be) visible for the purposes of business contracts among the participants. Coarse granularity reduces dependencies among the participants and reduces communications to a few messages of greater significance.
- **Team-oriented view.** Instead of framing computations centrally, it would be better to think in terms of how computations are realized by autonomous parties. In other words, instead of a participant commanding its partners, computation becomes more a matter of business partners working as a team. This means, that instead of an individual, a team of cooperating participants is a better modeling unit. A team-oriented view is a consequence of taking a peer-to-peer architecture seriously.

The following table summarizes the characteristics and features of the object-oriented, component-based and service-based software architectural models.

	Object-Oriented Development	***Component-Based Development***	***Service-Based Development***
Granularity	fine	medium	coarse
Reusability	low	medium	high
Coupling	tight	loose	loose
Dependencies	at compile time	at compile time	only at run time
Building blocks	objects	components	services
Functionality description	on class level	by interface declarations	network addressable service declarations
Communication scope	intra-application	intra-application	inter-application

Table 2.1: Comparison of architectural development models

An addition of a more abstract layer to the existing architectures surely solves many problems. The real success of SOA will still mostly dependent on the quality of the implementing technology. The current implementation for SOA will be discussed in Section 2.3. Before that, a short summary of the benefits SOA brings for business that are explained with the help of the SOA Maturity Model comes.

SOA Maturity Model

While software engineering is quick to embrace the technical value of service-oriented design, development and implementation, the executives face the very different challenge of accurately managing the investment in technology as it relates to business value. These IT managers and decision makers need help and guidance in communicating the business value of their SOA vision and to be able to benchmark their SOA adoption within the organization.

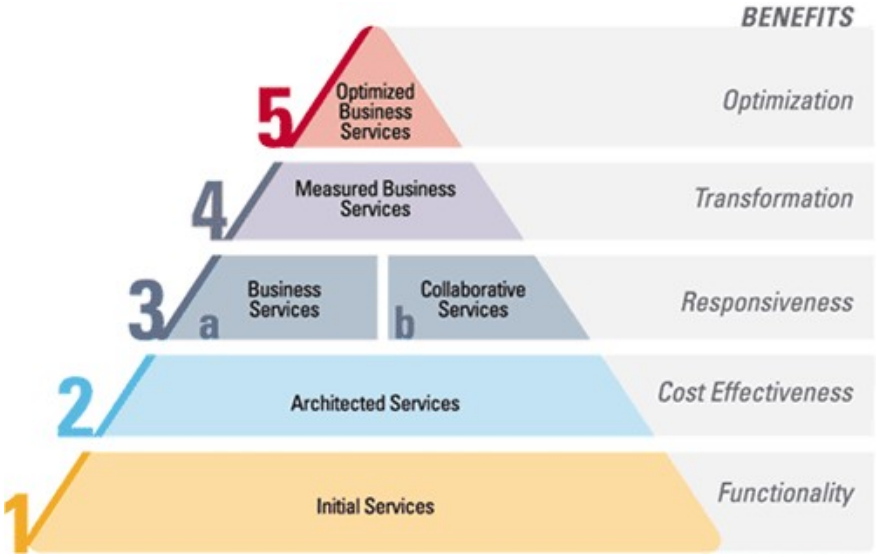

Figure 2.4: Service Oriented Architecture Maturity Model Levels with Key Business Impact (cf.[SoS06])

The *New SOA Maturity Model* (SOA MM) was published on October 27, 2005 and has emerged from the collaboration of Sonic Software[4] with its partners AmberPoint[5] and Systinet[6]. The model is designed to show the increasingly positive impact of SOA adoption from a business benefits perspective. It provides IT decision makers with simple framework for benchmarking the strategic value of their SOA implementation, and a model for visualizing future success.

The description of the 5 existing levels in the SOA MM (Figure 2.4) is presented now shortly.

SOA Maturity Model Level 1

SOA Maturity Level 1 is *Initial*. Initial Services represent the initial learning and initial project phase of SOA adoption. Projects here are typically done to simultaneously meet a specific need to implement functionality while trying out specific technologies and an approach to SOA. This maturity level also includes initial R&D activities testing the SOA technologies in a laboratory environment. Usually, the initial introduction of SOA is driven by the application development organization – often as part of an application integration project.

SOA Maturity Model Level 2

SOA Maturity Level 2 is *Architected Service*s. At this level that standards are set as to the technical governance of SOA implementation. The key business benefit of this level is development and deployment cost reductions through the use of SOA standard infrastructure and components as compared to using older technologies or costs accumulated through multiple unique one-time projects. These benefits are greater in the heterogeneous environments typical of most enterprises.

SOA Maturity Model Level 3

The focus of SOA Maturity Level 3 is on the partnership between technology and business organizations in order to assure that the use of SOA provides clear business responsiveness. Core to the value of SOA is the linkage between business process and digital processes. SOA Maturity Level 3 is defined with two complementary paths to attain the goals of *Business Services* focused on the improvement of internal business processes, and *Collaborative Services* focused on the improvement of collaborative processes with external partners.

SOA Maturity Model Level 4

While SOA Maturity Level 3 focuses on the implementation of internal and/or external business processes, SOA Maturity Level 4 focuses on measuring and presenting these processes at the business level so as to provide continuous feedback on the performance and business impact of the processes implemented at Level 3. This level includes business activity monitoring to allow business users to transform the way they respond to business events.

SOA Maturity Model Level 5

SOA Maturity Level 5, *Optimized Business Processes* SOA, adds automatic response to the measurements and displays of Level 4. In this way, the SOA information systems becomes the "enterprise nervous system" and takes action automatically according to events occurring at the business level according to the rule optimizing business goals.

The *New SOA Maturity Model* provides a framework for discussion between IT and business users about the applicability and benefits of SOA in an organization across five levels of adoption maturity. Its goal is not only to provide a means for organizations to benchmark current implementations, but also to offer a chance for IT leaders to visualize a path to successfully advance the value of SOA for their organizations.

4 www.sonicsoftware.com
5 www.amberpoint.com
6 www.systinet.com

2.2 Current SOA Implementation: Web Services

The concept of service-oriented architectures is not tied to any special technology. Nowadays, the technology most often used to implement SOA is the Web services technology. This subchapter will provide the background about the Web services, the standards concerning them and the standards bodies engaged.

Many definitions for a *Web service* are used today [Huh02]:

- a piece of business logic accessible via the Internet using open standards (Microsoft),
- encapsulated, loosely coupled, contracted software functions, offered via standard protocols over the Web (DestiCorp[7]),
- loosely coupled software components that interact with one another dynamically via standard Internet technologies (Gartner[8]),
- a software application identified by a URI, whose interface and binding are capable ob being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based protocols (W3C).

There are three well-differentiated roles in a Web services infrastructure which are shown in Figure 2.5. The three types of participants include:

- Service providers who create Web services and advertise them to potential users by registering the Web services with service brokers.
- Service brokers who maintain a registry of advertised (published) services and might introduce service providers to service requesters.
- Service requesters who search the registries of service brokers for suitable service providers, and then contact a service provider to use its service.

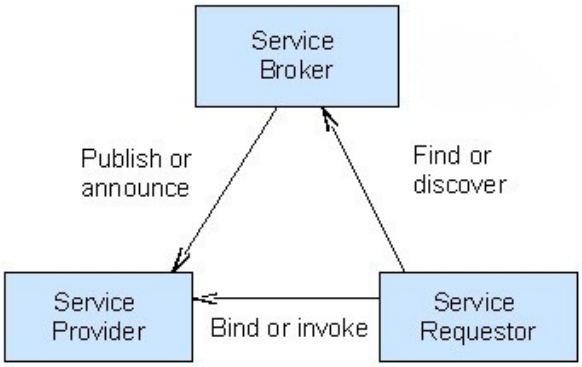

Figure 2.5: Roles in a service infrastructure

The main Web services standards are principally based on the following four components [PLL06]:

- an agreed upon transport protocol;
- a platform-independent format to describe the messages and their content;
- an interface description language which states what operations will be made available by the service with which messages;
- a common directory to publish and find the services.

7 http://www.desticorp.com/index.html
8 http://www.gartner.com/

The first component can be principally realized by any of the wide-spread transport protocols, such as SMTP or FTP. The most popular protocol in the Web services context is *HTTP*. HTTP has the advantage that, on the one hand, it allows the usage of the existing infrastructure while, on the other hand, it can be further developed basing on the available know-how of the users.

The second component is realized with Simple Object Access Protocol [Soa03], now known only by its acronym, *SOAP*. SOAP provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. Besides, SOAP specifies the binding to the HTTP as the underlying communication protocol between two addressable endpoints. The communication via SOAP over HTTP solves in some respects the existing problems in the RMI or CORBA approaches which are the reasons for tight coupling between the separate components. Besides, the latter are not suitable for open web infrastructures for, as a rule, the firewalls block the underlying proprietary protocols.

The Web Services Description Language [CCM01], *WSDL*, is used as the third component and provides the XML-based description of the user interface which makes the Web service available. WSDL permits the separate description of the abstract functionality of the service and the service access details. On the one hand, the operations of a service are described by its input and output parameters, on the other hand, an independent description of endpoint addresses to access the service and the transport protocols used is given.

Universal Description, Discovery and Integration [BCE02], *UDDI*, represents the forth component of the Web service based middleware solution and as a directory technology provides an interface to locate Web services. The UDDI interface allows to dynamically find the business partners and external services. The UDDI server is itself accessible over a Web services interface via SOAP and offers the operation to publish new services and search for the registered ones. The description in UDDI can in its turn contain a link to an existent WSDL description. Unfortunately, UDDI has not established as a widely-used technology.

There is a huge lack of moderation in the existing UDDI registries. An evaluation in [Mod02] has shown that 67% of entries in the present UDDI registries are invalid or unavailable, the reasons very probably being the complexity of the UDDI description for a Web service and the insufficient service description update mechanisms.

Besides, UDDI as a directory has only a restricted expressiveness what concerns the dynamic binding of the existing Web services to distributed applications. WSDL descriptions provide information in machine-readable form, and yet they concern only the syntax of the service interface, and not the semantics of the service. So is only the interface machine-readable, and not the functionality of the service. UDDI is restricted to the keyword search of the natural language description of the Web services. The present research is centered around the extension possibilities of the present Web services descriptions with the Semantic Web technology (Section 2.3).

Figure 2.6 shows the place of the standard protocols in the interactions between Web services infrastructure participants.

There is a whole range of further standards and recommendations for the technology summarized under the name of “Web services”. Those components described above only permit the description of services that follow simple interaction patterns as WSDL description are restricted to the small set of input/output operations. However, the business processes are often complex and need the description of more sophisticated interaction patterns.

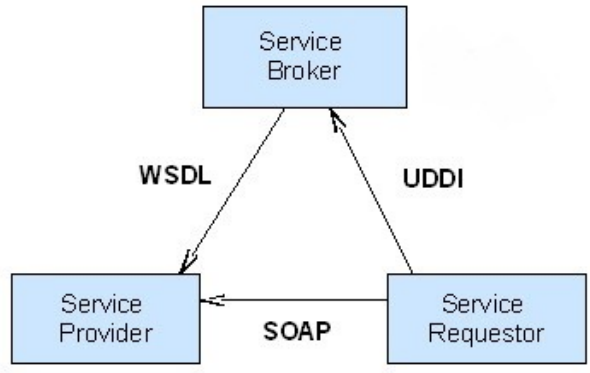

Figure 2.6: Protocols for Web services

The Business Process Execution Language for Web Services [ACD03], *BPEL4WS*, allows the description of such business processes based on WSDL and this way also the description of the executable combinations of different Web service invocations. The BPEL is an XML-based language for process description and execution which treats WSDL operations as separate activities. The overall process described can in its turn be made available as an independent Web Service.

E-Business eXtensible Markup Language [Ewe06], *ebXML*, is a set of specifications that together aim to enable a modular electronic business framework. ebXML specifications have XML messaging as a common basis. ebXML is a joint initiative of the United Nations (UN/CEFACT) and OASIS, developed with global participation for global usage.

But these approaches also make the main emphasis on the syntactic description of the Web services, and leave the semantic aspect out of consideration.

2.3 Semantic Web Technology

The Semantic Web is the vision of Tim Berners Lee, the creator of the Web, and is intended as an extension of the Web as it currently exists. Semantic Web aims to improve upon the meaning, in machine-understandable terms, of information currently available on the World Wide Web [BHL01]. This enables computers, in the form of autonomous software agents, to work with the wealth of World Wide Web information more easily. Moreover, it enhances the human-computer co-operation by bringing the concept of human understanding closer to the machine.

As with most Web-related recommendations and standards, W3C manages the development of the Semantic Web languages. Figure 2.7 displays the proposed layered architecture known as the *"Semantic Web Layer Cake"*. The W3C aims at working its way up the stack. The encoding layers and parts of the data layer are specified and working drafts exist for the ontology layer. Research groups are addressing the upper layers, however, no official W3C working group has been established for the logic and proof layers yet. The proof and trust layers as well as standard security features such as encryption, certificates, and digital signatures are long-term research goals and are moreover outside the scope of this project work.

The foundation is built by well-established and accepted Internet technologies, namely Unicode, the Uniform Resource Identification, *URI*, scheme and of course XML, XML Namespaces, and XML Schema. All layers above make heavy use of these core technologies. For instance, all mark-up languages are subsets of XML.

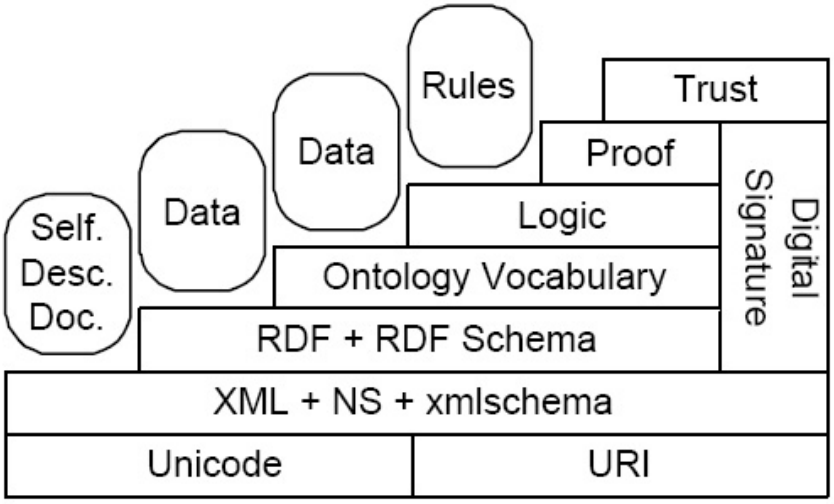

Figure 2.7: Semantic Web Layer Cake (cf. [BHL01])

The data layer allows representing information in an unambiguous way. An interconnected graph of data is established by using URIs to denote concepts and instances and arcs the relationships between them. Once captured, intelligence algorithms may be applied and deductions derived autonomously from axioms and assertions provided. Different applications can use this data or publish own information via this methodology. If such a Web enabled data representation approach is to be the basis of data integration, the meaning of globally referenced entities and concept must be specified. This is done in the ontology layer sitting on top of the data layer.

Ontologies formally represent a shared understanding about a domain. Therefore, they allow interpreting information from the data layer. One of the most cited definitions of ontology be found in [Gru93]:

> An *ontolog*y is a formal, explicit specification of a shared conceptualization.

Here, a conceptualization refers to people's conceptual understanding of a certain domain. While being very general, this definition captures the essence of what ontology means, regardless of potential application areas one might have in mind.

The logic layer contains domain knowledge in the form of rules allowing automated reasoning on available data. The idea is to be able to explicitly formalize knowledge, rather than embedding it in program code, which is hard to maintain.

2.3.1 RDF

RDF, or the Resource Description Framework [RDF06], is an XML based ontology language used for expressing semi-structured meta-data. There is no in-built restriction on semantics, but the triple-based syntactic structure of RDF allows applications to effectively extract potentially useful meta-information from a document. The core idea is that everything is treated as a URI. A triple consists of a class, property and value (or subject, property, and object which are used in this work synonymously). Each class is considered a *thing*. If Joe is Peter's brother, the following subject, predicate, object triple states this:

```
Subject:    http://www.mit.edu/~joe/
Predicate:  http://www.cogsci.princeton.edu/~wn/isBrotherOf
Object:     http://www.mit.edu/~peter/
```

In RDF, this subject, predicate, object triple is written as follows:

```
<?xml version='1.0'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:wn="http://www.cogsci.princeton.edu/~wn/">
<rdf:Description rdf:about="http://www.mit.edu/~joe/">
<wn:isBrotherOf
rdf:resource="http://www.mit.edu/~peter/" />
</rdf:Description>
</rdf:RDF>
```

RDF has no support for complex data types for properties and semantic constraints on concepts defined as a class. It is desirable, however, to express more sophisticated assertions.

2.3.2 OWL

The Web Ontology Language [OWL04], OWL, extends the RDF language-schema addressing the shortcomings outlined above. OWL was originally a part of the DARPA project as DAML+OIL [DAR06], and was renamed to OWL on submission as a standard to the W3C. An OWL knowledge base is constructed in a similar fashion to an RDF knowledge base.

```
<rdfs:Class rdf:ID="WINE">
      <rdfs:subClassOf rdf:resource="#POTABLE-LIQUID"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#MAKER"/>
            <owl:minCardinality>
               1
            </owl:minCardinality>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#MAKER"/>
            <owl:toClass rdf:resource="#WINERY"/>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#GRAPE-SLOT"/>
            <owl:minCardinality>
               1
            </owl:minCardinality>
         </owl:Restriction>
      </rdfs:subClassOf>
       <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#FLAVOR"/>
            <owl:toClass rdf:resource="#WINE-FLAVOR"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </rdfs:Class>
```

Figure 2.8: An OWL Wine Ontology excerpt

A class hierarchy is defined and properties are assigned to class concepts. The power of OWL emerges when one considers how it improves upon the RDF language. Firstly, OWL expresses complex data-types and value restrictions on those data-types. Secondly, through use of OWL keywords complex relationships between classes and types can be defined. Figure 2.8 taken from the WC3 OWL tutorial[9] shows how the concept of wine may be captured in an OWL ontology.

The ontology in Figure 2.8 defines the concept of "wine". According to the specification, a wine is a potable liquid produced by at least one maker of type winery, and is made from at least one type of grape. In addition, it can have some specific flavor.

OWL differentiates between the declaration of a concept and an instantiation of that concept. Essentially, OWL individuals are the extensional knowledge of an OWL knowledge base that serve as the application of the intentional knowledge defined by OWL structure keywords. Figure 2.9 illustrates the instantiation of an OWL individual.

```
<WhiteWine>
     <MAKER rdf:resource="#StGenevieve" />
     <GRAPE-SLOT rdf:resource="#Dry" />
     <FLAVOR rdf:resource="#Moderate" />
</WhiteWine>
```

Figure 2.9: An application of the abstract OWL Wine class

OWL provides a means by which equality and difference between semantic concepts can be expressed. `owl:equivalentClass`, `owl:equivalentProperty` and `owl:sameAs` can each be used to express equivalence between two syntactically differing concepts. For example, the concept of wine the concept of vino can be considered as the same concept defined differently. By asserting the Vino class equivalent to Wine class using `owl:equivalentClass` the concepts become equivalent in the eyes of any reasoner and deductions are made accordingly. `owl:sameAs` works in the same fashion except it is applied to individuals, not classes. The difference operators `owl:differentFrom`, `owl:AllDistinct`, and `owl:distinctMembers` apply the inverse semantics to individuals and individuals which are declared part of collections.

OWL property characteristics are used to enrich the semantics available in terms of class properties. `owl:objectProperty` and `owl:datatypeProperty` define object types and data-types respectively. Assertions such as `owl:transitiveProperty`, `owl:SymmetricProperty` further enrich the semantics with assertions on related properties of classes. A complete OWL specification is to be found under [OWL04].

Reasoning with OWL

OWL as well as RDF are closely connected to Description Logics (DL). There are several tools available for reasoning with OWL knowledge-bases. JESS [San06] rule engine, RACER [RAC06] and Pellet [MIN06] are three common rule-based tools used for querying knowledge base assertions. All of these tools have in-built support for OWL reasoning.

The W3C OWL standard is presented in three dialects and provides different computability guarantees. OWL Lite is a subset of the OWL DL language. Similarly, OWL DL is a subset of OWL Full. The first two of the above omit certain semantic restrictions, such as multiple cardinality, in order to guarantee a certain level of computability. The above mentioned reasoners are capable of reasoning over at least the OWL DL language. In fact, most support OWL-DL plus certain features of OWL Full. It is assumed, unless explicitly stated, that any references to OWL will refer to the OWL DL subset.

9 http://www.w3.org/TR/owl-features/

One of the possibilities to make inferences about the OWL ontologies is to explicitly use a query language, such as OWL-QL, that will be presented next.

2.3.3 OWL-QL

OWL Query Language [FHH03], *OWL-QL*, is a formal language and protocol for a querying agent (further referred to as a server) and an answering agent (further referred to as a client) on the Semantic Web for conducting a *query answering dialog* using knowledge represented in the Ontology Web Language. OWL-QL is an updated version of the DAML Query Language (DQL) [FHH03a] developed by the Joint United States/European Union ad hoc Agent Markup Language Committee1, and the authors of this paper, who are members of that committee, are the editors of both the DQL specification and the OWL-QL specification [FHH03b].

The design of OWL-QL is based on a number of basic assumptions about query answering dialogs on the Semantic Web, and on the intended role of OWL-QL.

First, the Semantic Web is expected to include many kinds of query answering services with access to many types of information represented in many formats. Traditional database query languages like SQL [InT92] and languages for retrieving information from the Web (e.g., XQuery [Mar03] and RQL [KaC03]) are not suitable for supporting such heterogeneity. OWL-QL supports query answering dialog in which the client may use automated reasoning methods to derive answers to queries, as well as scenarios in which the knowledge to be used in answering a query may be in multiple knowledge bases on the Semantic Web, even if those knowledge bases are not specified by the client.

Second, it is expected that some servers will have only partial information about the topic, some will have performance limitations, and some will be simply unable to handle certain kinds of queries. OWL-QL therefore provides an adaptable query answering protocol which both allows a server to return partial sets of answers as the answers are computed and allows a client to specify the maximum number of answers that it wants the server to include in the next set of answers it sends to the client.

Third, a Semantic Web query language needs to support queries that do not include a specification of the knowledge base to be used in answering the query. OWL-QL supports server selection of the knowledge base to be used in answering a query, and client requests that a server identify the knowledge base used in answering a query.

Fourth, the set of notations and surface syntactic forms used on the Web is already large, and various communities have different preferences, none of them universal. The essential aspects of the design of OWL-QL are independent of the surface syntax of the language. The OWL-QL specification is stated at an “abstract” or structural level, allowing essentially the same language to be implemented in multiple surface syntactic forms. The specification describes the types of objects (e.g., queries and answers) that are passed between server and client during a query answering dialog, the necessary and optional components of each of those object types, and the expected response of a server to each type of object sent to it by a client. The XML Schema provided in the OWL-QL specification is only one example syntax for the language.

The query syntax and the protocol structure of OWL-QL will now be presented in detail.

Query Answering Dialog

Figure 2.13 shows the schema of a OWL-QL dialog. The communication starts with a query sent by the client to the server. The server computes the necessary information and sends an answer (Answer Bundle) together with a Continuation Token that contains a Process Handle (analog to a Session-ID) with which the client can continue the communication. The dialog is over if one of the

parties sends a Termination Token. For the client, this means that it does not need any further answers, for the server – that it does not have any further information available.

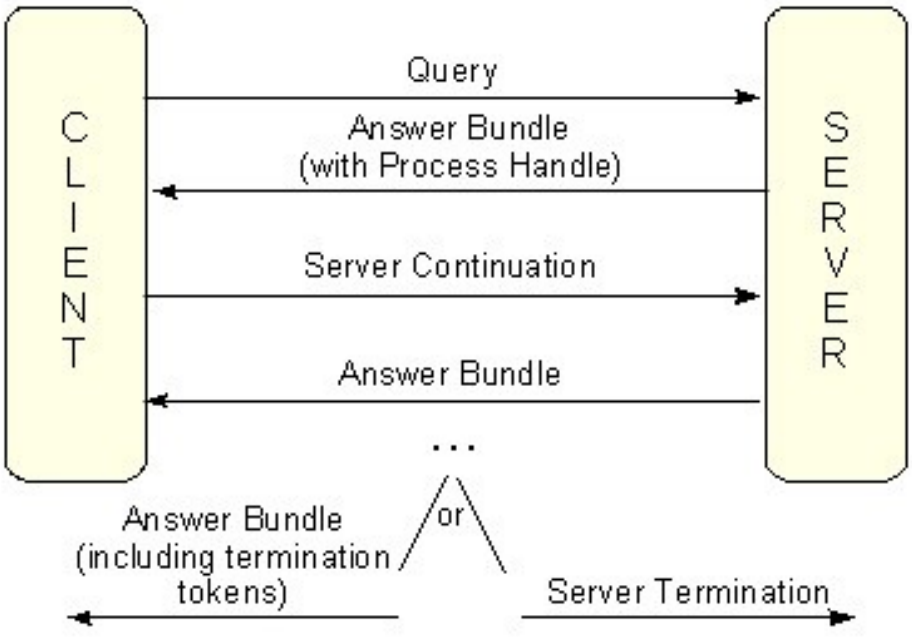

Figure 2.13: OWL-QL dialog schema

A special feature of OWL-QL is that it can support iterative queries. The dialog frame enables a client to get a restricted number of query answers without delay while after sending a Continuation Token that has a reference to a previous query further answers can be retrieved.

Query Syntax

OWL-QL defines a query document to be send to the reasoner. This document is composed of the following elements: premise, query pattern, answer pattern, must-bind variables, may-bind variables, don't-bind variables, answer knowledge base reference and answer size bound.

As already said, an OWL-QL query answering dialog is initiated by a client sending a query to an OWL-QL server. An OWL-QL query is an object necessarily containing a *query pattern* that specifies a collection of OWL sentences in which some URI references are considered to be variables. For example, a client could ask "Who owns a red car?" with a query having the following query pattern[10]:

```
Query: ("Who owns a red car?")
 Query Pattern: {(owns ?p ?c) (type ?c Car) (has-color ?c Red)}
 Must-Bind Variables List: (?p)
 May-Bind Variables List: ()
 Don't-Bind Variables List: ()
Answer Pattern: {(owns ?p "a red car")}
 Answer KB Pattern: …
 Answer: ("Joe owns a red car?")
 Answer Pattern Instance: {(owns Joe "a red car")}
```

10 Query patterns are shown here as a set of triples of the form (<property> <subject> <object>) similar to the KIF [Kno06] syntax, where any item in the triple can be a variable. Variables are represented as names beginning with the character "?".

A query may have zero or more answers, each of which provides bindings of URI references or literals to some of the variables in the query pattern such that the conjunction 3 of the answer sentences – produced by applying the bindings to the query pattern and considering the remaining variables in the query pattern to be existentially quantified – is entailed by a knowledge base (KB) called the *answer KB*. For example, the answer "Joe owns a red car." used in the previous example means the answer KB entails the following sentence, expressed here in first-order logic (using KIF syntax):

```
(exists (?c) (and (owns Joe ?c) (type ?c Car) (has-color ?c Red)))
```

Each binding in a query answer is a URI reference or a literal that either explicitly occurs as a term in the answer KB or is a term in OWL. That is, OWL-QL is suited for answering queries of the form "What URI references and literals from the answer KB and OWL denote objects that make the query pattern true?" or, when there are no variables to be bound in the query pattern, "Is the query pattern true in the answer KB?". A variable that has a binding in a query answer is *identified* in that query answer.

OWL has no suitable notion of a variable, so an OWL-QL query pattern is simply an OWL knowledge base, and a query specifies which URI references in its query pattern are to be considered to be variables. Data base query languages typically designate a subset of the variables in a query as being the variables for which bindings are to be included in a query answer.

In typical knowledge representation languages, such as OWL, a knowledge base may entail the existence of a query answer but not entail a binding for every variable in the query. For example, a knowledge base that says that every person has exactly one father (i.e., that every object of type '`Person`' has exactly one value of the property '`hasFather`') and that Joe is a person (i.e., that '`Joe`' is type '`Person`'), entails that Joe has a father but may not entail a value of property '`hasFather`' for Joe, e.g., if the knowledge base does not identify the father. OWL-QL supports existentially quantified answers by enabling the client to designate some of the query variables for which answers will be accepted with or without bindings. That is, each variable that occurs in a OWL-QL query is considered to be a *must-bind variable*, a *may-bind variable*, or a *don't-bind variable*. Answers are required to provide bindings for all the must-bind variables, may provide bindings for any of the may-bind variables, and are not to provide bindings for any of the don't-bind variables. These designations are made by inclusion of a *must-bind variables list*, a *may-bind variables list*, and a *don't-bind variable list* in an OWL-QL query. These lists contain URI references that occur in the query, and no URI reference can be an item of more than one of these lists.

Specifying a query pattern and the variables lists does not indicate how the answers – the bindings to the pattern variables – are to be returned from the server to the client. OWL-QL allows a client to specify the format in which answer bindings are returned by (optionally) including an *answer pattern* in a query that can be any list expression containing all of the query's must-bind and may-bind variables. If no answer pattern is specified, a two item list whose first item is the query's must-bind variables list and whose second item is the query's may-bind variables list is used as the answer pattern. Each query answer contains an instantiation of the answer pattern in which each variable having a binding in the answer is replaced by its binding.

Since OWL does not have an "implies" logical connective, "if then" queries such as "If Joe is a person, then does Joe have a father?" cannot be stated using only a query pattern. OWL-QL facilitates the representation of "if then" queries by enabling a query to optionally include a *query premise* that is an OWL KB or a KB reference. When a premise is included in a query, it is considered to be included in the answer KB. Omitting the query premise is equivalent to providing an empty query premise. Here is an example of a query that includes a premise:

```
Query: "If C1 is a Seafood Course and W1 is a drink of C1, then what
color is W1?"
Premise: {(type C1 Seafood-Course) (has-drink W1 C1)}
Query Pattern: {(has-color W1 ?x)}
Must-Bind Variables List: (?x)
```

The set of OWL sentences that are used by the server in answering a query is referred to as the *answer KB*. This may be one or more actual knowledge bases, or a virtual entity representing the total information available to the server at the time of answering. An OWL-QL query contains an *answer KB pattern* that is a KB, a list of KB references, or a variable. If a query's answer KB pattern is a KB or a reference to a KB, then the conjunction of the answer sentences specified by each query answer must be entailed by that KB. If a query's answer KB pattern is a list of KBs and/or KB references, then the conjunction of the answer sentences specified by each query answer must be entailed by the conjunction of the KBs in or referenced in that list. If a query's answer KB pattern is a variable, then the server is free to select or to generate an answer KB from which to answer the query, but if the variable is a must-bind variable, then the answer must provide a binding to the variable that is a reference to a resource representing the answer KB. In many cases, that URI reference will be a URL that can be used to access the KB or to communicate with the server about the KB, but the URI reference is not required to be a URL.

2.4 Semantic Web Services Description

As mentioned previously, the crucial disadvantage of the actual description standards for Web services is that they predominantly deal with the syntax – how Web service should be called – and lack semantics, i.e., the machine-understandable formalization of what the Web service is responsible for.

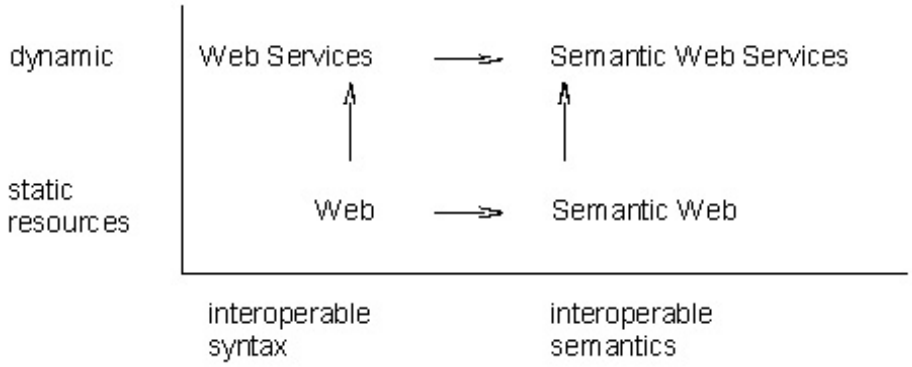

Figure 2.10: Development of the technologies on the Web

As Figure 2.10 highlights, Web service technology can sufficiently profit from the value added by the Semantic Web. The Web services technology pursues the transfer from the static to the dynamic content on the Web, whereas the Semantic Web aims at providing a shift from the interoperable syntax to the interoperable semantics. The application scenarios for the semantic descriptions of Web services technology generally correspond to the "standard" Web services technology with a difference that the use of the ontologies and the semantic descriptions of services promise a higher degree of automation (just like in the case with the semantically described static data).

The semantic description of services in such a way that their automatic use and re-use is really possible is not an easy undertaking, and the research in this field is still in its beginning phase. Presently, it is still not clear which strategy will in the end bring a breakthrough for the technology.

That is why, this section will present and compare several existing approaches. The agreement on the uniform communication infrastructure for Web services uncloses new possibilities: human developers and in the future probably also machines will find and call the suitable services to solve their current problems. The current research in this field is concentrated on the standardization of the semantic description, i.e., what the service offers to make its usage in different environments possible. The highest potential is expected from the following four aspects [PLL06]:

- *Discovery.* Before a Web service can be used in a distributed application, it should be made known for the developer or, in the automated case, for the software system. Present technologies (such as UDDI, see Section 2.2) support this design step solely by keyword search and standardized vocabulary (such as UNSPC[11]). Semantic annotation allows for the description of the services with the help of local ontologies that are connected by logic axioms with which the inference machines can compute the services set matching a certain query. Section 2.5 will consider the service discovery approaches.
- *Negotiation*. After a Web service that could be suitable for a certain problem solution is found the concrete service instance from the number of services that are available in the Web service should be determined. This means, for example, the agreement upon concrete transport and payment terms.
- *Composition.* In the case that a query cannot be processed by any of the Web services available, the semantic description provides an opportunity for the combination of several Web services.
- *Invocation.* After a service or a combination of services is found and chosen it can be executed. For this purpose, the information from the knowledge databases (for example, the input values that are contained in the semantic query description) is adapted to the formats required by the respective communication protocol.

Standards bodies have recognized the importance of the semantic annotation of the Web services for the real breakthrough of the Web services technologies. This is made clear by the the initiatives which pursue the further development and standardization of the technology. Among them are the undertakings of such experienced standards experts as the W3C (Semantic Web Services Interest Group [SWS06]) and the OASIS (Semantic Execution Environment Technical Committee [SEE06]).

Still the results of the standardization process for the corresponding technology as well as the maturity of the current standards for the use in industry is not achieved at the moment which was also confirmed by the W3C workshop “Frameworks for Semantics in Web Services“ [WWF06]. It mentioned the following reasons for the lack of a clear momentum at present towards a W3C recommendation track in this area: (1) use of these technologies is primarily in research and/or prototyping efforts at present; (2) lack of vendor commitment to provide tools and other forms of support; (3) the preference of the Web services community for a “go-slow” approach. Nevertheless, there is still big hope about the further development of the present approaches.

The rest of this section provides an overview about the three presently most important approaches to the semantic description of the Web services [PLL06]: OWL-S, WSML, and WSDL-S. It concentrates on the conceptual presentation of the description languages, the further details can be found in the respective specifications. For all the approaches, a short overviewof the support of the discovery, negotiation, composition and invocation is given.

11 United Nations Standard Products and Services Code: http://www.unspsc.org/

2.4.1 OWL-S

OWL-S [MBH04] was the first initiative to define a standard ontology to semantically annotate Web services. Since its first publication in May, 2001 under the name DAML-S up to the actual version 1.1 which was submitted in September, 2004 in W3C by Nokia, University of Maryland, the National Institute of Standards and Technology (NIST), Network Inference, SRI International, France Telecom, Stanford University, Toshiba, and the University of Southampton as a standard proposal, OWL-S has absorbed many improvements and enhancements. Still, basically all of its cornerstones have survived from the very beginning.

OWL-S is an ontology of service concepts. OWL-S organizes a service description into four conceptual areas: the process model, the profile, the grounding, and the service (Figure 2.11).

A *process model* describes how a service performs its tasks. It includes information about inputs, outputs (including a specification of the conditions under which various outputs will occur), preconditions (circumstances that must hold before a service can be used), and effects (changes brought about by a service). The process model differentiates between *composite*, *atomi*c, and *simple* processes. For a composite process, the process model shows how it breaks down into simpler component processes, and the flow of control and data between them. The subprocesses of the composite process are linked by control constructs, such as sequence, split and join, choice, iteration, and if-then-else. Atomic processes are essentially "black boxes" of functionality, and simple processes are abstract process descriptions that can relate to other composite or atomic processes.

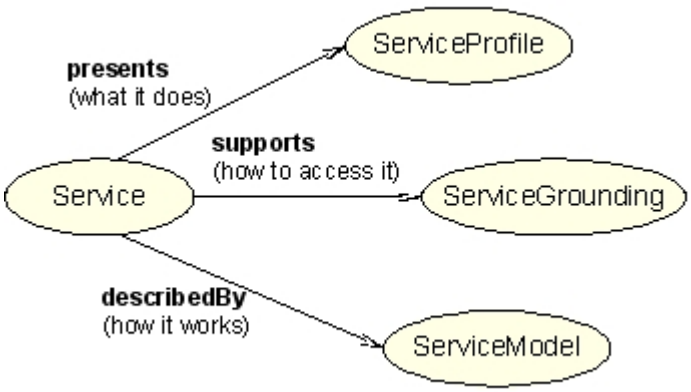

Figure 2.11: The top level of the service ontology (cf. [MBH04])

A *profil*e provides a general description of a Web service, intended to be published and shared to facilitate service discovery. Profiles can include both functional properties (inputs, outputs, preconditions, and effects) and nonfunctional properties (service name, text description, contact information, service category, and additional service parameters). The functional properties are derived from the process model, but it is not necessary to include all the functional properties from the process model in a profile. A simplified view can be provided for service discovery, on the assumption that the service consumer would eventually look at the process model to achieve a full understanding of how the service works.

A *grounding* specifies how a service is invoked, by detailing how the atomic processes in a service's process model map onto a concrete messaging protocol. OWL-S allows for different types of groundings to be used, but the only type developed to date is the WSDL grounding, which allows any Web service with a WSDL definition to be marked up as semantically using OWL-S.

A *service* simply binds the other parts together into a unit that can be published and invoked. The different parts of a service can be reused and connected in various ways. For example, a service provider may connect its process model with several profiles in order to provide customized advertisements to different communities of service consumers. A different service provider, providing a similar service, may reuse the same process model, possibly as part of a larger composite process, and connect it to a different grounding. The relationships between the service components are modeled using properties such as `presents` (Service-to-Profile), `describedBy` (Service-to-Process Model), and `supports` (Service-to-Grounding).

OWL-S is actually a metamodel for Web services and not just a description language. OWL-S is a OWL ontology, but OWL-S descriptions include much more semantic information than if the description logic of OWL alone is used. This is accepted by W3C itself and, on the one hand, there are attempts to formally describe the semantics of the OWL-S models [NaM02], on the other hand, there is research done on the proprietary abstract syntax for OWL-S services descriptions that will not be tied by the OWL restrictions.

- *Discovery.* The service profile is the element of OWL-S that is used for the description of Web services in directories and for their retrieval. There have been proposals made to embed the OWL-S profile descriptions into UDDI [SPS04]. But to use the semantic descriptions in reality, the presently very easy query interface of UDDI should be extended with an inference machine.
- *Negotiation.* To conclude the service agreement, OWL-S offers the non-functional attributes in the service profile. The combinations with such standards as WS-Policy [WSP06] are not considered in the present version.
- *Composition.* [SPW04] and [MaM03] make proposals to combine OWL with the planning problem solutions from the AI. But these planners do not use the whole potential of OWL-S, because of the extreme problem complexity. This complexity lets to assume that a long time passes before the automation of the Web services compositions will be achieved on the basis of the present technology. Therefore, it makes sense to specially consider the manually supported Web services composition.
- *Invocation.* The service grounding is the element that is responsible for the service invocation. The grounding allows for the linkage of the atomic processes from the service model to the WSDL operations whereas the separate input and output parameters are bound to the WSDL input and output messages.

There are already several implementations of OWL-S available. For example, OWL-S Matchmaker [Pao02] is a system for service discovery which was also integrated in UDDI. [SiP04] provides a Java interface for OWL-S descriptions which allows the parsing, serialization, and the execution of the OWL-S services. The OWL-S Editor [Ele05] is a plug-in for the ontology editor Protégé and permits the graphical editing of the OWL-S descriptions.

The Semantic Web Services Framework [BBB05], SWSF, an initiative that, in some respect, can be seen as a successor of OWL-S, tries to alternatively define the semantics of the OWL-S concepts in an ontology that is formalized directly in the first order predicate logic, and not in OWL. But as the SWSF has only had purely theoretical importance and there are no implementations whatsoever, it will not be considered in more detail in this work.

2.4.2 WSML/WSMO

The Web Service Modeling Ontology [BBD05], *WSMO*, proposed to the W3C in April 2005 is a conceptual model for the semantic service description. The WSMO working group was founded in

April, 2004 and is a primarily European initiative of the EU projects SEKT [SEK06], DIP [DIP06], and KnowledgeWeb [KWe06].

WSMO follows basically the same principles as OWS-S, but still with somewhat other focus. Just like in OWL_S, the ontologies are an important element, but WSMO is not formalized as an ontology itself, but in a meta datamodel, according to the MOF (Meta-Object Facility) methodology [MOF04]. This metamodel can be expressed in different knowledge representation languages.

WSMO follows the basic principal of strict separation of dimensions, i.e., service descriptions, ontologies, and user queries are independent Web services and are correlated by mediators. Thus, WSMO differentiates between the following four top-level elements for the description of Web services (Figure 2.12):

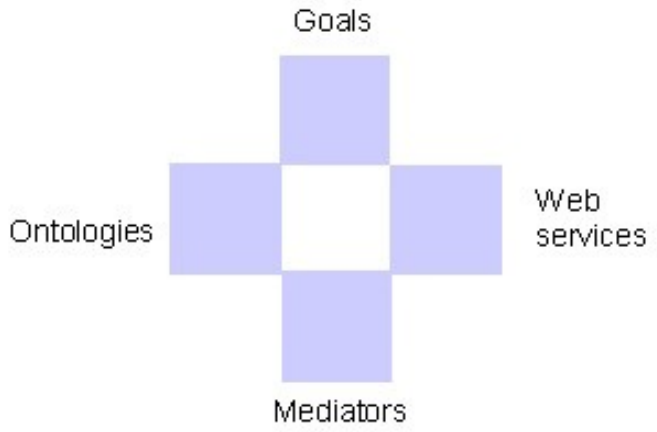

Figure 2.12: WSMO top-level elements (cf. [BBD+05])

Ontologies define the vocabulary for the description of all other elements, such as services and queries. Ontologies are formalized with the Web Service Modeling Language, WSML, a language that was specially developed for WSMO. WSML subsumes the expressiveness of OWL, but represents rather a frame-based than a keyword-based approach. The underlying logic formalisms to describe the axioms and rules in ontologies are based not on the description logic as in OWL, but on frame-logic [KLW95] and logic programming [Llo87].

Web services in WSMO are described from three different perspectives: non-functional properties, capability (functional properties), and interfaces (dynamic aspects, or the behavior of the Web services). Similar to OWL-S, the capabilities allow for the specification of the pre-conditions, assumptions, post-conditions, and effects with the help of the logical expressions. In the interface description, WSMO differentiates between the Choreography interface, i.e., the interface for the user, and the orchestration interface that shows what services and goals are invoked by the Web service. In the broad sense, choreography and orchestration interface in WSMO can be compared to the OWL-S process model. One more similar feature between the two approaches: WSMO allows for the connection to the existing WSDL descriptions over a grounding mechanism.

Goals specify the queries, i.e., the needed functionality from the user perspective. A goal represents a conceptual reflection of the service descriptions. The de-coupling of the goal and service descriptions as a separate entity (goal-driven approach) is a key difference from OWL-S. Mediators describe the elements to overcome the heterogeneity between different components. Mediators solve the incompatibilites on different levels.

- *Data level* – mediators define the rules to dissolve the terminology conflicts [ScB05] .

- *Process level* – mediators solve the conflicts basing on the different interaction patterns of the processes that are part of the Web services [CiM05] .

Mediators in WSMO can connect different other elements (ontologies, Web services, and Goals). Thus, one distinguishes between OOMediators, WWMediators, WGMediator, and GGMediator, whereas the first is a pure data mediator and the other three can also contain process mediation.

WSML is based on the existing standards, similar to OWL, and offers both XML and RDF serialization. The formal semantics of the WSML has not been completely defined for all WSMO elements yet.

- *Discovery.* WSMO compares the goal and service descriptions (capabilities) to discover the Web services. It is based on the logic programming.
- *Negotiation.* Policies in WSMO can be described to some extent with the non-functional attributes which reminds of the OWL-S approach. Besides, there are also proposals to integrate the peer trust rules into WSMO [OLP+04].
- *Composition.* The element to be used for the description of complex processes is Orchestration Interface. Presently, there are still no implementations for it. WSMO does not specify which description language exactly should be used for service composition, it only defines the conceptual model basing on the abstract state machines.
- *Invocation.* Similar to OWL-S, the grounding mechanism is being developed in WSMO presently. The connection to WSDL is achieved within the choreography and orchestration of the Web services. But it is still possible that this grounding mechanism will be in the end substituted by the model that is close to that of WSDL-S (s. below).

The Web Service Modeling Ontology shares the vision with OWL-S that ontologies are essential to support automatic discovery, composition and interoperation of Web services. But despite sharing a unifying vision, OWL-S and WSMO differ greatly in the details and the approach to achieve these results. Whereas OWL-S explicitly defines a set of ontologies that support reasoning about Web services, WSMO defines a conceptual framework within which these ontologies will have to be created. Another difference between OWL-S and WSMO is that while OWL-S does not make any distinction between types of Web services, WSMO places a lot of stress in the specification of mediators: mapping programs that solve the interoperation problems between Web services.

In WSMO's vision, mediators perform tasks such as translation between ontologies, or between the messages that one Web service produces and those that another Web service expects. In the process of defining mediators, WSMO produces a taxonomy of possible mediators that helps to define and classify the different tasks that mediators are supposed to solve. However, it can be difficult to map this taxonomy onto the classical problems of Web service interoperation; i.e. discovery, composition and invocation. For example, it is unclear how mediators can help during discovery, since discovery is intrinsically a selection problem, while mediators attempt to reconcile the differences between goals of Web services.

The definition of mediators in WSMO calls attention to some very important translation tasks that Web services face. Not surprisingly, these same translation tasks are needed in support of OWL-S Web services in their interaction. However, rather than stipulating the existence of a new type of component in the Web services infrastructure, OWL-S provides to Web services and their clients the information that is needed to find existing mediators that can reconcile their mismatches, or perhaps to create mediators through the process of Web service composition.

WSMO is supported by two development environments: [WST06] and [MDC+03]. There is a Java API available, [WSM+06], that provides tools to parse, validate and serialize the WSML descriptions and connection to various inference machines.

2.4.3 WSDL-S

WSDL-S [AFM+05] was proposed by IBM and the University of Georgia to the W3C in November 2005 and represents a bottom-up approach basing on WSDL and extending the existing Web service interface description with semantic information. This „lightweight“ approach offers limited potential for automation of processes, principally for simple interaction models, but can still be interesting for manual service discovery.

The WSDL-S philosophy is to extend upon the existing and accepted Web services standards to achieve prompt results. The key points can be summarized as follows:

- WSDL-S is directly integrated into the existing standards (WSDL).
- For the annotation there is no definite language prescribed for knowledge representation. Different knowledge representation formalisms should be allowed.
- The existing WSDL service parameter typing with XML Schema as supported in WSDL should be used in integration in the semantic description.
- A mechanism should be found to convert between the XML-based syntactic typing and the ontology concepts.

To achieve all this, WSDL-S adds a small number of a further elements to the existing WSDL standard. These elements allow for the annotation of the input and output parameters as well as the WSDL operations themselves. Pre- and post-conditions for operations give a semantical description of the environment before and after the operation execution. Besides, there is an opportunity to categorize WSDL 2.0 port types according to some ontology.

The second basic principal mentioned above has made it necessary to only represent the semantical models in WSDL as a reference and deposit them outside it. The concepts of the existing ontologies are specified with their URI. But WSDL-S does not specify how the ontologies should be defined.

The integration of the XML Schema is realized with two alternatives for the annotations. Either the complex types or the separate subordinate elements of the complex type can be annotated. As a mechanism for the information conversion from ontologies (such as OWL) or their equivalent into the XML Schema, the references to the translation rules (in the XSLT) are used.

- *Discovery.* The description of the pre- and post-conditions allows for the service discovery, just like in the other approaches described. A P2P infrastructure represented in [VSS+05] used WSDL-S description formalism. Similar to OWL-S based discovery, it is also based on the subsumption between concepts in a description logic.
- *Negotiation.* As a lightweight approach, WSDL-S offers no explicit support for negotiation. But as a WSDL-S as an extension of „pure“ WSDL can be easier, in comparison to the other approaches, combined with the corresponding Web service standard recommendations, such as WS-Policy.
- *Composition.* WSDL-S does not dispose of the expressive power to describe the processes, the description of the complex interaction patterns is also not supported. But the sequences of WSDL operations can be expressed implicitly if the necessary information is provided in the pre- and post-conditions.
- *Invocation.* As an extension of WSDL, WSDL-S offers support for automatic Web service execution basing on the existing approaches to the execution with the help of WSDL described services. To integrate the additional semantical information, a mapping between the ontological concepts and the corresponding XML Schema, similar to OWL-S, is necessary.

There are several tools that support WSDL-S that come from the METEROR-S project [Met06]. There has also been a Web service discovery infrastructure developed [VSS+05] and a framework to annotate the Web services [POS+04].

While comparing WSDL-S approach with OWL-S, one notices that the semantic expressiveness is rich and flexible in OWL-S, it defines a new way to describe Web services and suffers from limitations. First, the OWL-S profile model duplicates the descriptions embodied in the rest of WSDL (namely input and outputs). This leads to the inconvenience of creating multiple definitions for describing the same service. Second, it assumes that everyone uses OWL for representing ontologies which may not always be the case. WSDL-S was created to overcome these limitations. Besides, while it is noted that the theoretical underpinnings of OWL-S in description logic makes it a richer language for representing semantics, extending the industry standards such as WSDL to include semantics may prove to be a more practical approach for adoption.

For the purposes of this work OWL-S has been chosen as the the longest available and most elaborated standard. Its support for logic reasoning was also an important criterion of choice. It is still too early to discard OWL-S despite its limitations as they have not yet been completely overcome by any other alternative.

2.5 Semantic Web Service Discovery

Service Discovery can be defined as locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria. For the users to discover a service means to get its description so as to be able to decide whether they want to use it. User-centered Web service discovery is in any case based on the technology available to automate this process.

Although heavily supported by languages such as OWL, OWL-S and RDF as well as SOAP and XML research into semantic service discovery is still maturing and, as a result, a standard means of discovery is still a way off. As a result of this non-convergence, research continues in several parallel avenues outlined below.

Semantically Enhanced UDDI

As already said, UDDI alone does provide any semantical support. Still there is a very active body of research in semantically enhancing the UDDI registry standard. Since the UDDI standard is plentiful in features and a mature standard, it seems a logical progression to attempt to build on this maturity by adding semantic annotation. In [AGD+03] the authors endeavor to provide a structure whereby semantic information may be annotated onto current UDDI elements, such as *tModel*. Similarly, [SPS04] endeavor to ”import” the semantic web into a UDDI standard implementation. Each of these works aims to introduce concept matching to the UDDI registry by incorporating reasoning and OWL-S support to current implementations. The active research in this area highlights one of UDDI’s main weaknesses, lack of service capability support and emphasizes a general consensus amongst the web service academic community that semantic support for capability matching of web services is primary the area forward.

OWL-S Matchmaking

In [PKP+02] Paolucci et al. outline a methodology and efficient algorithm for semantic service capability matching. The current body of research focuses primarily on comparing inputs and outputs of a service as semantic concepts represented in OWL. By extracting subsumption relationships between input requirements and outputs, the authors propose a way of ranking semantic matching results. This ranking can be used in conjunction with other user-defined, or plug-in, constraints to inform of an exact, or potentially useful web-service capability match.

In [SPS04] the same authors propose an efficient way to apply the matching methodologies outlined in [PKP+02] to the UDDI Registry. This basic extension adds a *capability port* to the current UDDI implementation thus making it semantically aware. An interesting contribution of [SPS04] is an evaluation of ranked matching and a resulting focus on accelerating performance by minimizing the amount of matching and, therefore reasoning, that takes place.

Ranked Matching

The work done in [JRM+00], focuses on a finer grained approach to matching than presented in [PKP+02]. By consideration of the service category and finer-grained user constraints based on concept properties as well as input and output matching the work done by Jaeger et al. [JRM+00] proposes a more accurate approach to semantic matching. The matching process is broken into four distinct phases; input matching, output matching, service category matching and user constraint matching, each of which scores a numerical ranking, also based on the subsumption relation. The semantic matcher then aggregates a ranking in each of these categories and as a result can produce an accurate match with informative matching statistics. A Java prototype has been built and is hosted by the Technische Universität Berlin.

Reasoning

Reasoning for the semantic service discovery is used in connection with such a query language as OWL-QL presented in detail in Section 2.3.3.

The OWL-QL Web site[12] provides links to the OWL-QL specification and to current OWL-QL implementations, including an OWL-QL client with a Web browser user interface suitable for use by humans for asking queries of an OWL-QL server. But the proposed syntax (the KIF similar syntax just used for the illustrations) only let experts profit from the technology. One cannot expect all the users to write such queries. This problem will be addressed in chapters 3 and 4.

2.6 Summary

The power of service-oriented architectures lies in the fact that it allows easier integration of distributed computing applications, including intra- as well as interenterprise integration. Web services that use standard protocols for service interface descriptions (WSDL) and service invocation (SOAP), coupled with a global data format (XML), were introduced to turn this vision of the service-oriented architecture into reality.

Main challenges of the further development of Web services concern automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based systems. For the solution of these problems, the description of services on the semantic level is necessary, but the present standards only take the syntactic properties of the service into account, and leave the semantics out of consideration. To support the development of semantic Web services, the Semantic Web technology proposes several concurrent specifications (OWL-S, WSMO, WSDL-S), whereby OWL-S remains the major standard available.

Semantic Web service discovery is one of the key problems that has to be solved before the Web service technology can reach its full potential. Existing approaches, such as service matchmaking or query-answering schema of OWL-QL achieve good results, but in the present state they remain useless for the general public, i.e., for the users of the computer systems and require advanced computer proficiency to be successfully used. The adaptation of the technology is needed so that it is made accessible for all.

12 http://ksl.stanford.edu/projects/owl-ql/

Chapter 3 Requirements Analysis

Reusing the conceptual and technological background presented in Chapter 2, this chapter will analyze the requirements for the conceptual architecture for the user-centered service discovery support in a digital library environment. As an example and a case study, a digital library system for teachers (the Hamburger Medienindex, *HMI*) will be used.

The analysis made and the requirements accumulated throughout this chapter will be later made use of for the conceptual design in Chapter 4.

3.1 HMI Platform

The HMI project suggests and evaluates a digital library for teachers which provides them with opportunities to gather, classify, share and re-use the teaching materials. It is the follow project of the WEL ("Warburg Electronic Library") [WEL06] and TEFIS ("Technology and Information System") [TeF06] and is accomplished in cooperation with the [LLS06].

The HMI architecture (Figure 3.1) is based on two pillars: the library and the community platform. The library model is defined using the open and dynamic asset model [Sehr04] defined at STS. Community platform supports cooperation and communication within user group: community and user blogs, events, etc. Services bridge both pillars. The configuration of services and their interdependencies is supported by the Spring Framework [SpF06] used throughout the three top-most layers. The layers underlying the services layer in the HMI architecture are responsible for modeling, and storing the data and are based on integrated heterogeneous components. As content and metadata repository, CoreMedia Content Application Engine [CoM06] is used together with an Oracle database [ORD06]. InfoAsset Broker [InA06] is the core element of the community platform. Systinet Server [Sys06] and Axis [ApA06] serve for the integration of external services. Services completely disclose the complexity and heterogeneity of the system from the users as they manage the library invoking the services through the user interface that hides in its turn the technical complexity of the services. HMI-based services can also be published as Web services.

A usage scenario for the HMI can be described as follows. The community browses plain Web- and teaching-specific sites to gather teaching material. The library only manages higher quality material collected, classified and annotated by community. The library users work in public, community-specific and private spaces. The library provides services to seamlessly use external services (e.g., the BookSearch service that is based on the Amazon Web Service [AWS06]).

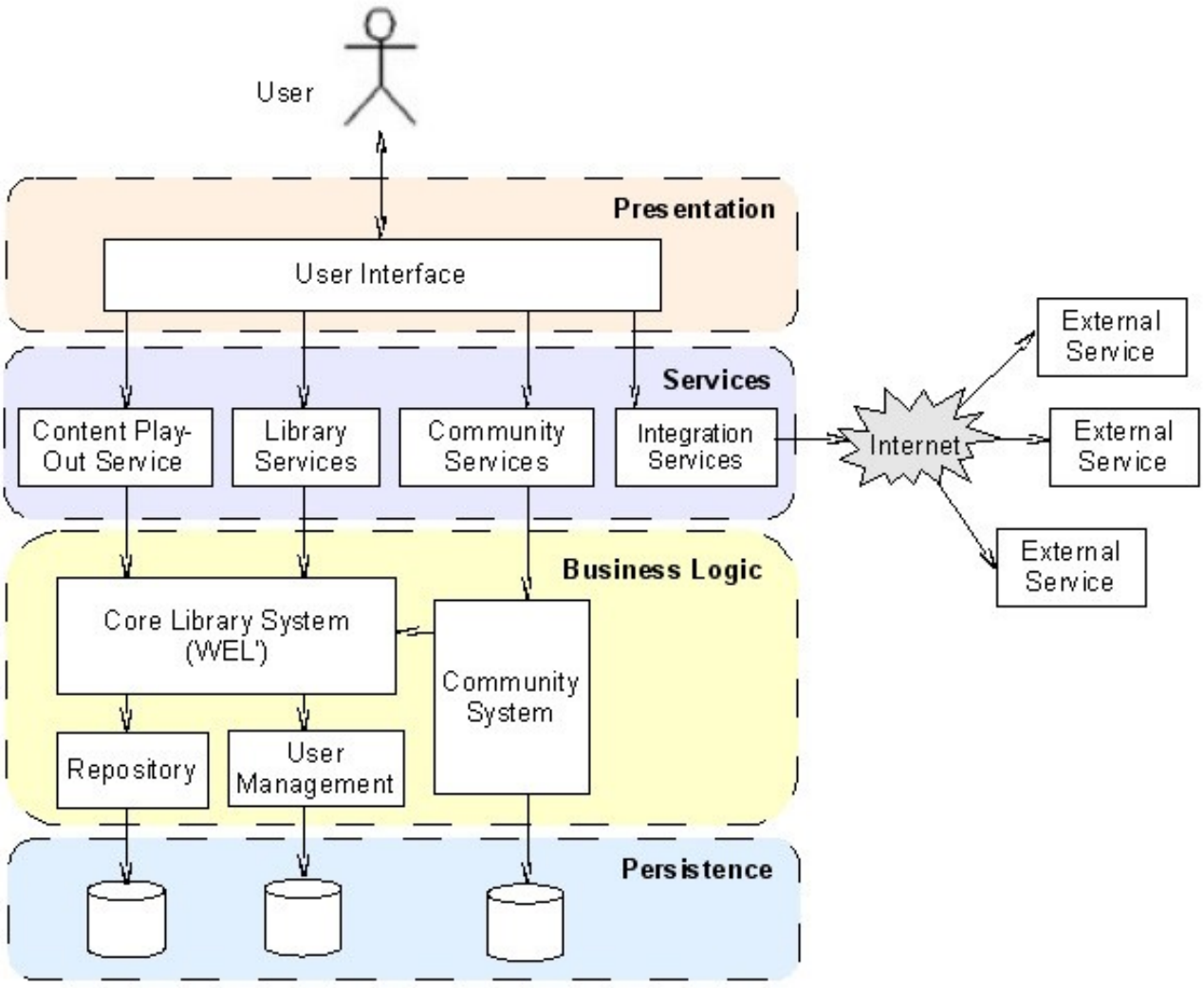

Figure 3.1: HMI system architecture

The HMI services can be classified into several groups:

- **Library-Management Services**:

  – Classifier Management Services – manage Classifier structures: AddClassifier, Remove-Classifer, EditClassifier;

  – Artefact Management Services – manage Artefacts: AddArtefact, EditArtefact, Remove-Artefact;

  – Personalization Services – add personalized adaptations of libraries;

  – Acquiring Services – material acquiring: Book search (external);

  – Classification Services – classify acquired material: ClassifyArtefact;

  – Search Services – search material or classification structures: ClassifierSuggest, Search-Artefact (with keyword search);

  – Annotation Services – Annotate, rate, recommend material;

- **User-Management Services:**

  – Group Management Services – manage user groups and single users: CreateUser, Edit-User, RemoveUser, CreateGroup, EditGroup, RemoveGroup;

– Authorization Services – manage access rights: UpdateAutorization;

- **Content Playout Services**:

  – Web-based content playout;

- **Community Services:**

  – Manage community or personal weblogs;

  – Integrate Weblog as form of annotation into HMI Core Services.

Further, the analysis will center on the library- and user-management services as they are the most important access points for an HMI user to interact with the system. For example, HMI Media Playout Services imply that the content searched for is found. They could be thought as part of some composite service from the user view, but composite services will not be considered in this work. The further discussion will also take into consideration the services that are already modeled (present in the classification), but not implemented yet[13].

The number of services in the system is constantly growing. If the user is supposed to access them only with the help of the user interface, the latter will get so complex that finding the webpage with the form that fulfills the user's needs becomes very tedious. That is why, service discovery in the HMI gets so important. As the previous chapter has shown, service discovery when made by means of the present technology is, in its turn, too complex for a user without a strong technical background. So the next sections have the purpose to analyze what adaptation of the technology can be done to make service discovery more user-friendly.

3.2 Services Discovery Analysis in HMI

For the purpose of the user-centered Web service discovery in the HMI, a User-Centered Discovery service should be added to the HMI services. It will provide the users with essential means to find the services they need. The HMI users will interact directly with the User-Centered Discovery service interface. To analyze the possible user interaction with such a service, the next section will present the major use cases.

3.2.1 Use Cases for User-Centered Service Discovery in HMI

In this section, a short list of very simple use cases will be sketched to provide a basis for the further analysis towards the definition of the requirements for the user-centered service discovery. In all use cases, the result is that the users who want to find a service that can satisfy their needs obtain information from the published service advertisements. Figure 3.2 presents the possible use cases.

Use Case 1: Browse Service Descriptions

In the case of UDDI (Section 2.2) browsing can be understood as browsing through the advertisement repository, whereas the party interested in a specific service manually finds what s/he wants by drilling down through the categories. But for users without a strong technical background such as HMI users in a normal case (if they are not computer science teachers specialized in the Web service technology), browsing through the advertisement repository does not help much as the repository information does not have a user-friendly format and is badly categorized. As already said, there are almost no well-defined taxonomies for services advertisements (Section 2.2). So the users should have an opportunity to browse through the categories they understand well. As such, the domain objects can be well-suited if they are presented as a domain-specific ontology. As all the

13 The HMI Community Services have not been modeled in detail yet in the HMI and will not be considered here.

services in a system have to do with the domain objects in one or another way (have them as functional parameters, make use of them in pre-conditions and results) then the domain objects through which the user browses can be seen as a starting point to get to the services that are relationally connected to the domain objects. The concrete case differentiation useful here will be done in the next chapter.

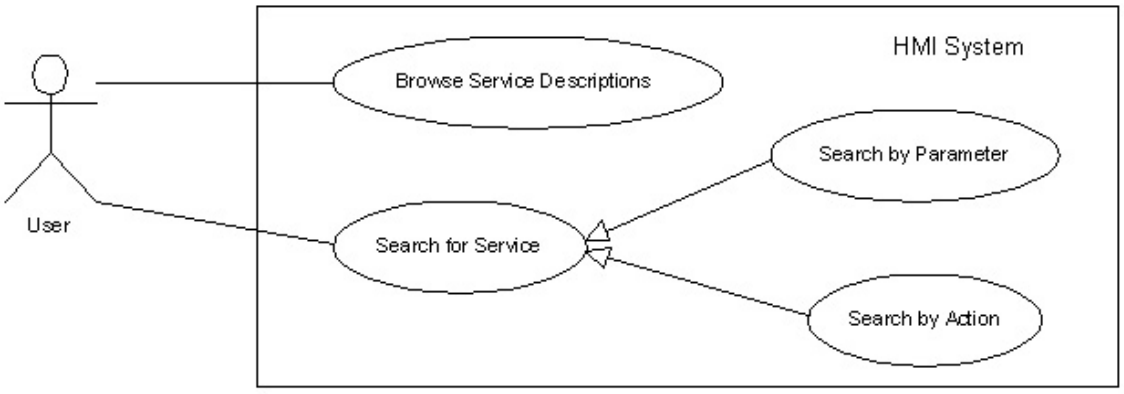

Figure 3.2: Use cases for user-centered service discovery

Use Case 2: Search for Service

Search means submitting a query to the system which describes what is searched for and getting an answer that was generated and returned back. The user-friendliness of the query language is its main criterion, others will be determined later in this chapter. It is important to define the search criteria that will be reflected in the query language. From the users' point of view a service is seen as some piece of functionality changing the domain objects like 'Classifier', 'User', or 'Image', or, on the other hand, performing some action like 'search a book', 'change user (profile)', or 'create a classifier'. That is why, the use cases “Search by Parameter” and “Search by Action” are subordinate use cases for “Search for Service”.

Use Case 3: Search by Parameter

This kind of service search may include searching for services which have a domain object as either an input or output parameter or both of them.

Use Case 4: Search by Action

A a service 'does something', or 'performs an action' as illustrated in the examples above, it will be important to see how these actions can be modeled for the domain under discussion.

For any of the use cases presented a mapping from the user formulated queries having the for the users attainable level of complexity into the full-form OWL QL queries is indispensable. Primarily, to provide this mapping, a special User-Centered Discovery Service should be added to the HMI system. Its tasks will be discussed in detail in the next section.

3.2.2 Requirements for User-Centered Discovery Service

The User-Centered Discovery service's main task consists in concealing the internal technical complexity of the service discovery from human users. It gets the queries formulated in a well-adapted human comprehensible format from the users. It generates OWL-QL messages basing on the mappings defined for the queries. And then submits them to the knowledge base query component (it can be an OWL QL query service as proposed by [RaM06]). It is important for the User-Centered Discovery service that the knowledge base itself is designed taking into

consideration the human view upon the domain under query. As the users will browse through the domain objects and use them in their queries they should very well understand the concepts they are dealing with. So designing the knowledge base for the domain by only taking the technical domain model will not be sufficient.

It is proposed to use an ontology as a well-suited abstraction for a knowledge base and at the same time a human comprehensible conceptual model of the domain. Here are the principles that are important to consider while building the domain-specific ontology for the HMI:

- the domain-specific ontology is an extended subset of the technical model;
- the concepts used are well familiar to the human users (from the domains of digital libraries, digital media, Web, time, or learning materials);
- if it is not possible to eliminate the unfamiliar or technical concepts they should be precisely commented and examples of familiar concepts describing them should be given;
- re-use of other existing ontologies is welcome.

These principles will be taken into consideration while creating a domain-specific ontology for the HMI in Section 3.3.

The other part of the already mentioned knowledge base along with the domain-specific ontology is a collection of the HMI service descriptions. The most important information for the discovery of services is contained in the Service Profile if the OWL-S upper ontology for semantic description of services is used as chosen in Section 2.4 for the semantic description of the service instances. So the HMI services will be described with OWL-S with a special emphasis on the creating Service Profiles that are well-suited for the service discovery by human users, while describing the services the domain-specific ontology built will be re-used. The possible service discovery patterns by human users and the approaches to adapt the service profiles for the better success of discovery will be presented in Chapter 4.

The core of the suggested User-Centered Discovery service is the mapping component that generates OWL QL queries from the queries specified in the suggested user-friendly domain-specific query language. The last underlies the following requirements:

- intuitive for amateur users;
- flexible inside the domain;
- extendable – for the case new services are added to the system;
- limited in the number of keywords that should be learned or presented in the UI;
- similar to the search query languages known to users (e.g., those used in search engines);
- easily visualizable (e.g. choice possibilities in each state can be easily summarized and presented)
- subset of OWL-QL so that the mapping is easy and the relevant technology can be reused.

The design of this user-friendly domain-specific query language and of its mapping to OWL QL is postponed until Chapter 4, the next step will be the analysis of the HMI domain and building an HMI domain-specific ontology.

3.3 Domain Analysis

The analysis already made has pointed out that the existing technical domain model cannot be reasonably used for the purposes of the user-centered service discovery support. This section will

analyze which parts of the existing HMI model (the part of the model relevant for services is shown in Figure 3.3 can be included into the domain model, discuss the reasons and search for sensible extensions of it so that the it gets well-comprehensible by the human users.

The HMI data model makes use of such technical concepts as 'Classifier', 'Artefact', or 'MediaContent' that do not belong to the conceptual domain shared by the HMI users who are normally teachers and not advanced computer scientists. These concepts do not speak of concrete, or at most digital, objects that the user is accustomed to face in daily life, such as a “book”, a “DVD”, an “film”, or a “notebook”. They are conceptual abstractions that are introduced by the software engineers to build a complex system upon a complex domain. They are not fine-grained either, as it is typical for the objects one encounters in life, like an “image”, or a “text”. They are specially made so abstract that embracing new fine-grained concepts to the technical model is not a pain for those who have to adapt the model.

As presented in Section 2.3.2, ontologies are the way to conceptually model any application domain in such a way that the model becomes generally comprehensible by human users and processed by machines. So building a domain-specific ontology will provide the users exactly with the taxonomy of concepts, and their relationships that are understood by them, on the one hand, and can very well be mapped to the technical model, on the other hand, – as the ontologies can be processed automatically. Besides, the automatic processing of ontologies provides opportunities to reason over them which is, as already said, important for service discovery as such. The last reason explains that if the HMI domain-specific ontology is to be built with the means of OWL than it should be the OWL DL dialect of it so that the reasoning services can be used.

But before searching for the suitable instantiations in real life for the abstract and extensible concepts from the HMI model, one can already notice another adaptation and simplification case from the user's point of view. The users will very probably not know much about the underlying WEL model (everything above the separating line in the UML class diagram in Figure 3.3) that was the basis for the HMI model (everything under the line). 'WELObject', 'WELUser', 'WELUserGroup' 'WELRelationshipAttribution', 'WELObjectAttribution' should be eliminated as giving no information to the user.

The only information explicitly given to the users about WEL is that “the HMI is a WEL library” (a known fact). It is reflected in in the ontology, along with a basic classification of HMI_Library itself which is shown in the bottom-most branch of the HMI domain-specific ontology (Figure 3.4). The concept of 'Library' includes the concepts of 'OnlineLibrary' and 'MediaLibrary'. There is no distinct boundary between them as almost every 'MediaLibrary' nowadays provides digital resources on-line, and on-line libraries can manage 'tangible' resources which one supposes to find in a 'MediaLibrary'. The classification of the 'HMI_Library' is not needed by the HMI services alone, but this is one of the points where the flexibility and extensibility of the ontological definition is assured. Figure 3.4 shows the taxonomy of concepts proposed for domain-specific ontology for the HMI ('is_a' relationship). The top-most concepts in the HMI library are 'HMI_User' and 'HMI_Item' which are both classified as 'WEL_Items'. All the proposed 'Items' in the ontology, such as 'WEL_Item', 'HMI_Item', 'Web_Item', or 'MediaLibrary_Item' are suggested in analogy to the 'owl:Thing' with the meaning that we have to do with “Things from the WEL”, “Things from the HMI”, or “Things from the Web”. The word “thing” is considered to be too general in this case and is replaced by the word “item”.

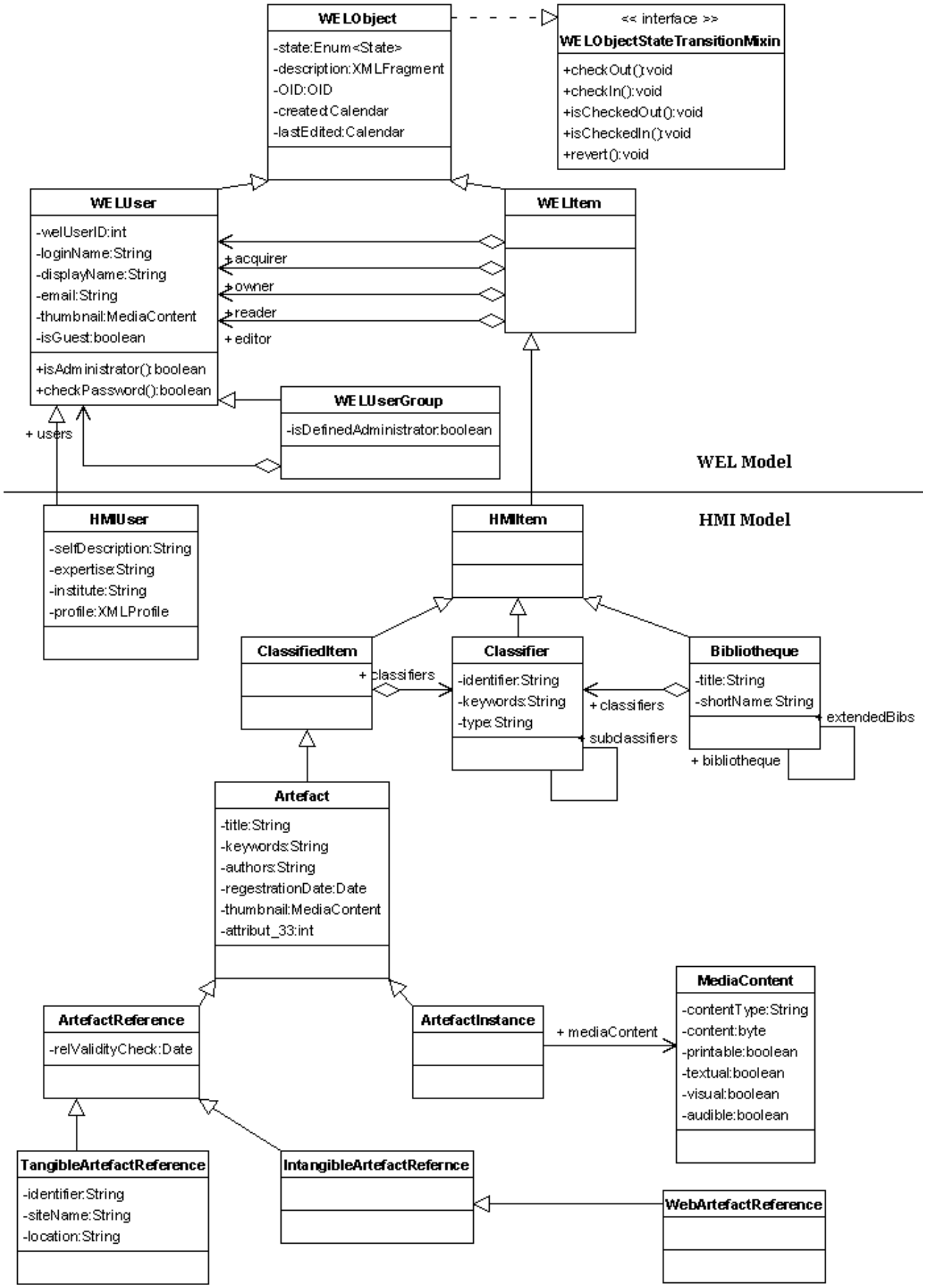

Figure 3.3: HMI Domain Model

The complex data structure and the component architectural pattern (simplified) one finds in the technical model of the HMI to express the “user” and “user groups” notions can be mapped to the three ontological concepts: 'SingleUser', 'UserGroup' and 'HMI_User' comprising both first.

Almost the complete hierarchical structure underlying 'HMI_Item' is mapped from the HMI model into the domain-specific ontology including the 'Classifier', 'Bibliotheque' and 'Artefact' subtrees. The 'ClassifiedItem' class is considered to be too abstract and adding no useful information and the 'Artefact' is thereby a direct subclass of 'HMI_Item'. These classes from the HMI model as well as such subclasses of 'Artefact' as 'ArtefactReference' and 'ArtefactInstance', 'TangibleArtefactReference' and 'IntangibleArtefactInstance', and 'WebArtefactInstance' are important concepts that the HMI domain is impossible to think of. Although they are rather abstract they must be present in the ontology as well. For better understanding of these concepts the ontology provides sufficient comments (with rdfs:comment) about such concepts. For instance, 'Artefact' is commented with “Any digital content stored in the library or references to the external content.”

To provide more concrete concepts as illustration of the existing abstract concepts outlined above, inside the 'Web_Item' hierarchical subtree under 'DigitalContent', the same classes are found as those that conceptual designer of the technical HMI model thought of when including 'ArtefactInstance' into the model. Examples of them are 'Text', 'Image', 'Video', or 'Animation'. As for 'Webpages', they can, in general, also be stored in the library as such completely. But there is a different idea available only to store a reference to the webpage or site. 'WebArtefactReference' has thus 'URL' as its subclass, it is the other subclass of 'Web_Item' along with the 'DigitalContent'. 'TangibleArtefactReference' is aimed to contain references to real-world objects that do not have a digital form, like a “book” or “newspaper”, as well as those digitized that are sometimes called “hard-media” like “DVD” or “CD”. These are the items that a 'MediaLibrary' contains. On this way, 'MediaLibrary_Item' becomes a subclass of 'TangibleArtefactReference'. Other classes of objects can be thought of that will also be subclasses of 'TangibleArtefactReference' like 'Notebook', e.g., if someone has collected his or her own teaching notes.

Until now the disjointness of the introduced classes was not discussed. The mutually among themselves disjoint classes are as follows:

- 'DigitalContent', 'URL';
- 'Animation', 'Application', 'Image', 'Text', 'Audio', 'Video';
- 'HMI_Item', 'HMI_User';
- 'Artefact', 'Bibliotheque', 'Classifer';
- 'ArtefactInstance', 'ArtefactReference';
- 'TangibleArtefactReference', 'IntangibleArtefactReference'.

Besides, 'Library' is mutually disjoint with 'WEL_Item' and 'Web_Item' which are not disjoint as they have some common subclasses. 'Site' and 'Webpage' can be hardly differentiated from other digital content. Thus, the disjointness between them is not defined. The class 'MediaContent' from the HMI model is only the encapsulation of some important properties, such as 'isPrintable', 'isTextual', 'isVisual', and 'isAudible' having a boolean value (datatype properties in OWL). One can easily think of services examples that make use of these properties, e.g., 'KeywordSearchService' would need something that 'isTextual' to be able to process it. All the four properties are defined for 'Artefact'. For 'Classifier' the property 'isTextual' is set to “true”.

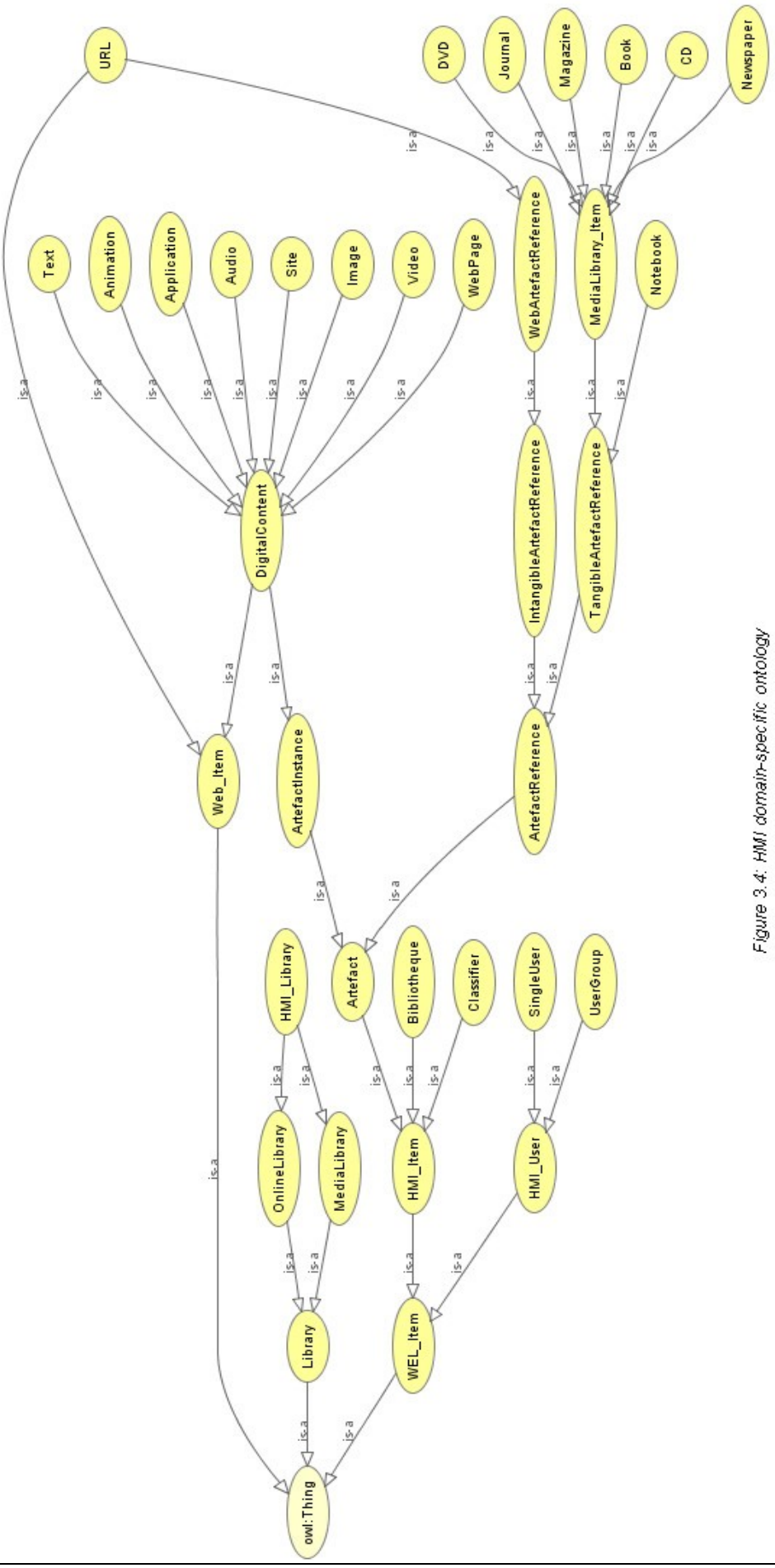

Figure 3.4: HMI domain-specific ontology

Other datatype properties follow immediately from instance attributes of the HMI model classes. 'isGuest' is a boolean property of 'HMI_User', and 'hasKeywords' is a string property of 'Classifer'.

More important for the service description later are object properties to be defined. All the properties in the domain-ontology fall into two groups: roles and inclusions (Table 3.1). Roles almost always correspond to the roles on the associations found in the HMI model, such as 'creator', 'editor', or 'owner'. The reasons to call the corresponding properties in the ontology with the action verbs in the imperative 'create', 'acquire', or 'search' and not, e.g., 'creates' or 'acquirer' will become clear in the next chapter.

Roles in the HMI model	***Actions as OWL Properties***	***Subject → Object Relashionships in OWL***	***Inverse Property***
'editor'	'create'	'HMI_User' → 'HMI_Item' 'HMI_User' → 'HMI_User'	'isCreatedBy'
'acquirer'	'acquire'	'HMI_User' → 'Artefact'	'isAcquiredBy'
'searcher'	'search'	'HMI_User' → 'HMI_Item'	'isSearchedBy'
'reader'	'read'	'HMI_User' → 'HMI_User' 'HMI_User' → 'HMI_Item'	'isReadBy'
'editor'	'update'	'HMI_User' → 'HMI_User' 'HMI_User' → 'HMI_Item'	'isUpdatedBy'
'editor'	'delete'	'HMI_User' → 'HMI_User' 'HMI_User' → 'HMI_Item'	'isDeletedBy'
'annotator'	'annotate'	'HMI_User' → 'HMI_User' 'HMI_User' → 'HMI_Item'	'isAnnotatedBy'
'owner'	'own'	'HMI_User' → 'HMI_Item'	'isOwnedBy'

Table 3.1: Actions as properties in the HMI ontology

Many properties reflect the composition of related objects. Very often this relationship is an additional means to express UML class hierarchies, especially for classes that were not taken into the ontology. For example, a hierarchical relationship exists between 'WELUser' and 'HMIUser' is not expressed in the domain-ontology as 'WELUser' was not added to it. Thus, 'hasMember' property (inverse of 'isMemberOf') is defined between 'HMI_UserGroup' and 'HMI_User' as well as between HMI_UserGroups. Other examples (inverse properties are given in the brackets):

- 'hasClassifier': 'Bibliotheque' → 'Classifier' ('hasParentBibl');
- 'hasSubClassifier': 'Classifier' → 'Classifier' ('hasParentClassifier');
- 'contains': 'WEL_Library' → 'WEL_Item'; 'HMI_Library' → 'HMI_Item', 'MediaLibrary' → 'MediaLibrary_Item', 'OnlineLibrary' → 'Web_Item' ('isContainedBy');
- 'hasExtendedBibl': 'Bibliotheque' → 'Bibliotheque'.

The 'hasCreationTime' property has instances of 'time-entry: CalendarClock-Description' from the well-known time-ontology [PaH04] as its values.

The following definitions could be built for the considered ontology:

- $\text{HMI\_Library} \equiv \text{WEL\_Library} \cap \forall(\textit{contain}.\,\text{HMI\_Item} \cup \textit{contain}.\,\text{HMI\_User})$
- $\text{Artefact} \equiv \text{HMI\_Item} \cap (\text{ArtefactInstance} \cup \text{ArtefactReference})$
- $\text{HMI\_Item} \equiv \text{WEL\_Item} \cap (\text{Bibliotheque} \cup \text{Classifier} \cup \text{Artefact})$
- $\text{HMI\_User} \equiv \text{HMI\_Item} \cap (\text{SingleUser} \cup \text{UserGroup})$

The open-world assumption of OWL is very helpful for creating a domain-specific ontology as it is really not possible to describe all of the classes in the domain. It is possible that more classes exist then described, but adding them to the ontology will, first of all, not break the technical HMI model with all the tedious work connected to it, and besides, be available at once for the reasoning services.

3.4 Summary

In such a heterogeneous system as the HMI that includes a wide variety of services, users need a strong support to find the services that fulfill their needs. Providing the service access points through the user interface alone that is done in such cases in general cannot be seen as a good solution any more as the complexity of the user interface grows in parallel to the number of services provided. The semantic service discovery technology that aims to solve the problem of finding the right service cannot be applied by the users alone, as it is too complex. This makes it obvious that an adaptation of this technology is needed so that human users can benefit from it, too.

These considerations and the analysis of the service discovery use cases have made it obvious that the mapping mechanism is needed from a user-friendly domain-specific query language to the technical realization query language such as OWL-QL. The requirements for such a simplified query language were discussed in this chapter as well as the guidelines according to which the User-Centered Discovery service should be designed and the services – described on the semantic level.

This chapter also analyzed the HMI model and as a result a domain-specific ontology of the HMI was created. The next chapter will make use of this ontology while defining the proposed domain-specific query language and its mappings to OWL-QL.

Chapter 4 Design of User-Centered Discovery Service for HMI

With the help of the analysis done in the previous chapter, a User-Centered Discovery Service will be conceptually designed.

To achieve this goal the HMI services should be first semantically described and then according to the requirements defined in Section 3.2.2 OWL-QL will be mapped to the user-friendly query language.

4.1 Semantic Description of the HMI Services

All HMI services are instances of the HMI_Service that is a subclass of OWL-S Service. Still additional properties are needed for the HMI_Service. One of them is the property '`hasAccessURL`' which binds the Service instance to the URL instance. This solution is needed for the general OWL-S grounding approach (see Section 2.4.1) is too technical for human users and the HMI services are supposed to be accessed over the user interface available at the URL referenced by '`hasAccessURL`'.

As already underlined, the most important information for service discovery is provided in OWL-S Profile which defined the service capabilities, i.e. the functional and non-functional properties. The functional properties that are mostly taken into account here are '`hasInput`', '`hasOutput`" (subclasses of '`hasParameter`'), '`hasPrecondition`' and '`hasResult`'. For example, the `CreateClassifier` service has, speaking formally, in terms of the technical model of the HMI User (who performs the creation) and ClassifierCreateData as input parameters, and `Classifier` as an output parameter. As the technical model of the HMI has already been mapped into the domain-specific ontology the corresponding concepts can be easily used while describing the services semantically. But as for `ClassifierCreateData`, it does not provide the relevant information that the users need to know before invoking the service. Such technical concepts – they were not included into the ontology either – will not be part of the semantic service description. Other examples include `ClassifierRemoveData`, `ArtefactUpdateData`, or `UserProfileData`.

The pre-condition of the `CreateClassifier` service can be defined as:

```
loggedIn(?User, ?SessionID),
```

whereas the effect, or post-condition, can be described by (in a pseudo-language):

```
IF
     isAuthorized(?User, ?Classifier) AND isValid(ClassifierCreateData)
THEN
     hasParentBibl(?Classifier, ?Bibliotheque) AND
     create(?User, ?Classifier).
```

The same pre-condition is true for the BookSearch service, the post-condition of which will look like:

```
IF
     (exists hasTitle(?Book, ?inputParameter1)) OR
     (exists hasAuthor(?Book, ?inputParameter2))
THEN RETURN.
```

These conditional expressions are too complex and are not the way the users formulate their wishes. The user's view on the result achieved by the service is other than the formally defined result in the system. For example, in the first case, the result perceived by the user will be Classifier created and can be expressed by the relationship `(User create Classifier)` in RDF triple syntax (`<subject> <property> <object>`), in the second – the '`Book`' found, and the relationship coming into the foreground is `(User search Book)`.

Let's consider again the properties defined in the HMI ontology (Table 3.1). Exactly the properties falling into the category 'roles' or 'actions' are the properties utilized by the HMI services from the user's view:

```
(User edit HMI_Item),
(User acquire Artefacts),
(User delete HMI_Item).
```

For such an action as “User search Book by Title” that is what can be done with the BookSearch service and is described with additional specialization expression (“by Title”), some additional factors should be considered. First of all, it can be separated into two logically conjuncted triples `(User search Book) AND (search hasCriterion Title)`. Unfortunately, OWL DL does not allow defining properties of properties. Then, all types of search will have to be modeled as additional subproperties of the property search which is allowed. Thus, the domain ontology will be extended with an axiom `(searchByTitle rdfs:subPropertyOf search)` and the result of service description will have to be specified by `(User searchByTitle Book)`. Analogous:

```
(searchByAuthor rdfs:subPropertyOf search)
(User searchByAuthor Book).
```

Thus, it is reasonable to define a new property 'action' and make all the action properties its subproperty.

The property '`hasResult`' mentioned above provides a relationship between a Profile instance and an instance of 'Result' which is part of the process model ontology. OWL-S provides the following relationships:

```
(profile:Profile hasResult process:Result)
(process:Result hasEffect expr:Expression)
(expr:Expression expressionLanguage LogicLanguage).
```

Whereby, KIF, SWRL, DRS can be chosen as a logic language. The '`Expression`' itself is then added as a string inside the XML document. The sense of such an inclusion becomes clear if one thinks of the impossibility to characterize properties through properties in OWL DL.

Thus, the resulting expression for the `BookSearch` service, for example, with the KIF language chosen, looks like:

```
(or(search HMI_User (or Book DVD CD))
   (searchByTitle HMI_User (or Book DVD CD))
   (searchByAuthor HMI_User (or Book DVD CD))).
```

And the corresponding expression for the `CreateClassifier` service is given by:

```
(create HMI_User Classifier).
```

After preparing the knowledge base of service descriptions with the adaptations proposed, the possible query types can be discussed in detail. The next section will have a look at the query patterns taking into consideration the use cases described in Section 3.2.1.

4.2 Query Patterns

This section presents possible user query types and how they can be mapped into OWL QL. The user queries can be first grouped according to the use cases described in Section 3.2.1.

Browse Service Descriptions

To move through the taxonomy of the domain classes, the user needs a possibility to find the subclasses and ancestor classes of the given class. The **"`subclassOf <domain class name>`"** query maps into OWL QL query:

```
(rdfs:subClassOf hmi:<domian class name> ?subclass) must-bind ?subclass
```

and, respectively, the "subsumes <domain class name>" query:

```
(rdfs:subClassOf ?subsumes hmi:<domian class name>) must-bind ?subsumes
```

But more important for the users is the ability to get to know the most general subclass or a most specific one. Unfortunately, the ability to ask such so called "structural queries" [FHH03] is limited to concepts which can be expressed using OWL. There is no way in OWL to express the concept of a most general subclass or a most specific type. The OWL-QL query pattern language was not extended beyond the expressive capabilities of the content language used in the knowledge bases being queried (i.e., OWL) so as not to impose greater computational burdens on a server than are defined by the specification of the language it uses [FHH03].

So to obtain answers to queries involving concepts not expressible in OWL such as "most general subclass" or "most specific type", and also to optimize any variable with respect to any given transitive property, by using an iterative optimization technique described in [FHH03].

To optimize the value of a must-bind variable `V` in a query `Q` with respect to a transitive property `P` and a server `S`, send `Q` to `S` asking for at most one answer. If `S` provides an answer to `Q` with a binding of B_i for `V`, then send `S` a query `Q'` consisting of `Q` with the additional premise `(P` B_i `V)` and ask for at most one answer. If `S` does not provide an answer to `Q'`, then B_i is the optimal binding that `S` can provide for `V`. If `S` provides an answer to `Q'` with a binding of B_j for `V`, then send `S` a new query `Q'` consisting of `Q` with the additional premise `(P` B_j `V)`. Continue this iterative querying

until S does not provide an answer. The last binding produced for V is the optimal binding that S can provide for V.

For example, a client could use iterative optimization to find the most general subclass of C by asking for at most one answer to a query with query pattern (subclassOf ?x C) and must-bind variable ?x, and then successively asking for at most one answer to the same query with the addition of premise (subclassOf C_i ?x), where C_i is the most recently returned binding for the variable ?x.

Thus, if the automatic exchange and processing of queries is automatized then the users can just use additional keywords, such as "last" and "first" to express their wish to get the the direct parent or subclass in the answer respectively. These words would formally mean starting the query exchange according to the an iterative optimization method described.

Search by Parameter

The services description allow for differentiating between input and output parameters of which the users are also well-aware. Thus, if the users search for a service which takes a domain object as, e.g., an input parameter they are provided with a simple query type **"input <domain class name>"** which maps to the OWL-QL query:

```
Query:  (rdf:type ?service hmi:HMI_Service)
        (service:presents ?service ?profile)
        (rdf:type ?profile profile:Profile)
        (profile:hasInput ?profile ?input)
        (rdf:type ?input process:Input)
        (process:parameterType ?input ?parameter)
        (rdf:type ?parameter hmi:<domain class name>)
        (hmi:hasAccessPoint ?service ?URL)
        (rdf:type ?URL hmi:URL)
        (rdfs:comment ?service ?commentary)
             must-bind ?service don't bind ?URL ?commentary
Answer: (Service ?service is described as ?commentary can be accessed at
?URL).
```

The answer to this query will provide the URL and the commentary of the service. Both are needed so that the users get to know what the service does through its human-understandable description one finds in the commentary as well as the URL of where the service can be accessed.

A similar query will demand all the services that get a specific domain-object as an output parameter: "output <domain class name>". The difference from the OWL-QL query it is mapped to, is that 'profile:hasInput' in the forth line will have to be changed to 'profile:hasOutput'.

If the user is not going to specify whether the domain-specific object appears as an input or output parameter of the service, the query will be like **"parameter <domain class name>"** and will be mapped to another similar query with 'profile:hasParameter' in line four.

Search by Action

This time the query is aimed at finding the service by actions separated from other properties in Sections 3.3 and 4.1. The first type is finding the list of all the actions available with the keyword **"action"**:

```
(rdfs:subPropertyOf hmi:action ?subproperty) must-bind ?subproperty.
```

The second query pattern – searching for a service itself providing the action chosen – will be composed of the action name with the name of the domain class which can be seen as the object of the action performed: **"<action name> <domain class name>"**. This pattern maps into an OWL-QL query of type:

```
(rdf:type ?service hmi:HMI_Service)
(service:presents ?service ?profile)
(rdf:type ?profile profile:Profile)
(profile:hasResult ?profile ?result)
(rdf:type ?result process:Result)
(process:hasEffect ?result ?effect)
(rdf:type ?effect expr:Expression)
(expr:expressionLanguage ?effect expr:KIF)
("<action> User <domain class name>")
(hmi:hasAccessPoint ?service ?URL)
(rdf:type ?URL hmi:URL)
(rdfs:comment ?service ?commentary)
    must-bind ?service don't bind ?URL ?commentary.
```

As already said, the '`hasEffect`'-property must have a string inside according to OWL-S, so, in this case, the expression in the string has to be made part of the analysis. To achieve this, another expression is written into the query in the string form. As all the services in the library are user-centered, i.e., the '`User`' is the one who performs the action, from the users' point of view, the actions are performed as such upon their command. This is the point where one can profit from the infinitive form of the verbs in the action properties. Such a query is perceived by the user just as a command to the system to 'do something'.

To process such a query the sting will have to be extracted from it, and after services satisfying the query are found their action expression from the query and the service description will have to be compared. As the action expressions are nothing else than OWL property restrictions, to compare them matchmaking of OWL property restrictions through concept subsumption can be applied. It will guarantee that the chosen service will perform the action demanded by the user.

4.3 Evaluation

The experiences made with the semantic description of Web services have shown that there are conceptual drawbacks in OWL-S that make service modeling tedious. Especially, modeling service parameters in the Service Profile is connected with defining many cross references in the Process Model. Besides, other languages embedded into OWL-S make the design hard.

Another important point that has to be made is that presently the support of an ontology designer or a semantic service annotator through reasoning services and tools is not sufficient. The boundaries

of what features, properties, or predicates, can be described by Description Logics and are supported by the concrete reasoning engine, are not clear yet. Reasoning engines provide new optimizations and open the area possible for description. The available tools for ontology development, like Protégé [Ele+05] are not on the up-to-date level and may cause confusion declaring that the expression used is not part of DL, although it is probably just the absence of an optimization that leads to such assumptions. The boundaries of the modeling opportunities offered by Description Logics itself have still to be researched.

The Semantic Web is not a ready-to-use off-the-shelf technology. Although it impresses with a great number of proposed standards, concepts and design abstraction, they are still in development, often not compatible to each other. Besides, the boundaries are still being widened making some practices possible today that were not possible yesterday. But there is no guarantee that the reasoning service you are looking for is already available.

Another difficulty encountered is the lack of ready-to-go ontologies connected with the HMI domain (libraries, multimedia, or teaching materials) or their immaturity. The only ontology that could be utilized during the creation of the domain-specific ontology for the HMI is the time ontology.

4.4 Summary

This chapter has presented a semantic description of the HMI services with OWL-S so as to reflect the users' view on their functional and non-functional properties. Thereby, some adaptations had to be made to the OWL-S ontology, e.g., the technical binding was extended by a a '`hasAccessURL`' property to give users a chance to access an UI-enabled service found. Further analysis has shown that another adaptation of OWL-S Profile is necessary so that the goal of the service the users understand well can be described formally. Such a goal maps, on the one hand, to the OWL-S Profile's '`Result`' and, on the other hand, to the action properties of the HMI domain-specific ontology. In praxis, they are described as class restrictions inside strings.

Besides, various query types for the user-centered query language and their mappings to OWL-QL have been classified according to the use cases defined in Section 3.2.1. The users are provided with a few intuitively clear keywords and structures to formulate the queries. The greatest problem outlined is the matching of the user defined goals in the query with the goals defined by the service. Service matchmaking is proposed to solve this problem.

Chapter 5 Concluding Remarks

This chapter will provide the summary of the work and its results and outlook about the possible perspectives for future work.

5.1 Summary

The intersection of such research areas as service-oriented architectures and the Semantic Web provides the background for the semantic Web service discovery. Chapter 2 has stated that although both technologies have not yet reached the necessary level of maturity they have very high potential when applied to current complex heterogeneous computer systems to solve the problem of interoperation on different levels. The principles and goals of service-oriented architectures and their implementation with the Web service technology have been analyzed in Sections 2.1 and 2.2. The current state of research in the Semantic Web along with its main concepts (especially, that of ontology) and standards (such as RDF, OWL, OWL-QL) have been presented in Section 2.3.

It has been shown that the main drawback of the Web service technology relevant for service discovery is that the current standards only deal with syntax and ignore semantics. The Semantic Web standards (such as OWL-S, WSMO, WSDL-S) enhance the Web service technology and add semantics to the service descriptions. These standards have been analyzed and compared in Section 2.4. The possible approaches to solve the semantic Web service discovery problem were introduced in Section 2.5. They include, e.g., semantically enhancement for UDDI, service matchmaking, or reasoning with the help of a query language such as OWL-QL.

Although many approaches exist for the semantic Web service discovery, being very technical, they generally address computer experts or automatic machine interactions and ignore the fact that the search for services can also be pursued by human users without advanced technical background.

The rest of this work has been an attempt to design a conceptual architecture that supports the users in discovery of services in heterogeneous library environments such as the HMI media library. Chapter 3 has analyzed the requirements for service discovery, it has presented the HMI system and attempted to classify the services found in it (Section 3.1). The analysis of the use cases (Section 3.2.1) has provided the guidelines for the conceptual architecture (Section 3.2.2). The domain-analysis (Section 3.3) has resulted in building a domain-specific ontology for the HMI.

For the design of the conceptual architecture, the choice has been made in favor of the OWL-S standard for the semantic Web service description (Section 4.1) as well as OWL-QL for the

semantic Web service discovery. The patterns of the OWL-QL queries have been modeled and mapped to a user-centered discovery language in Section 4.2. The evaluation of the designed architecture, the experience with the used technologies and enhancement possibilities (Section 4.3) have finished Chapter 4.

As a result of the design phase a User-Centered Discovery service has been proposed in this work to support the users while searching for services in the HMI system. It hides the technology complexity from the users with the help of the user-centered query language which is intuitive, domain-specific, has a restricted vocabulary and is mapped to a subset of OWL-QL and gives the users a chance to search for the services with simple queries. Thereby, the domain-specific ontology of the HMI and the semantically described service instances serve as a knowledge base for OWL-QL queries into which the user queries are translated.

5.2 Outlook

The practical realization of the proposed architecture, could be done under support of RacerManager [RaM06]. It is an open-source semantic web infrastructure that serves as a scalable front-end for applications to efficiently query OWL ontologies and implements OWL-QL. RacerManager makes use of the OWL reasoning capabilities of the DL Reasoner RACER [RAC06]. RacerManager serves as an OWL-QL application server for the DL Reasoner RacerPro. To support the OWL-QL communication scheme (s. Section 2.3.3) RacerManager has a service-oriented architecture. It offers OWL-QL support as a webservice. Clients such as software agents or web applications can reference and query any OWL ontology by calling the web service using standards such as SOAP and WSDL.

In the implementatio, RacerManager has to be informed about the knowledge base including the HMI domain-specific ontology and semantic service descriptions. The OWL-QL queries generated by the User-Centered Discovery service have to be sent to RacerManager and the responses redirected back to the service so that the results can be presented to the users.

The experiences gathered during this work (s. Section 4.3) have extended the list of known limitations of OWL-S [BLW04]. The absence of the user goal concept was a big design obstacle. Opposite to OWL-S the goal concept is available in WSMO (Section 2.4.3) as one of its a central concepts. This work showed how important it is for the semantic discovery of services, especially if the clients searching for services are human users. It was stated that the user goal and the service capability are separate in practice and should be described separately. Thus, an interesting research topic were to compare WSMO and OWL-S as to how much they are suitable for user-centered service discovery in library environments.

Besides, after the ontologies in the related domain ([BOE06], [Mar06]) reach a better stage of maturity, an integration of them with the HMI ontology can be thought of as an important task.

Bibliography

[AGD+03] Akiragiu, R., Goodwin, R., Doshi, P. and Roeder, S. A Method for Semantically Enhancing the Service Capabilities of IDDI. ACM, 2003.

[BBB+05] Battle, S., Bernstein, A., Boley, H., et al. Semantic Web Services Framework (SWSF). W3C Member Submission, May 2005;
available at http://www.w3.org/Submission/2005/07/ (2005).

[BGO+04] Baida, Z., Gordijn, J., Omelayenko, B., and Akkermans H. A Shared Service Terminology for Online Service Provisioning. In ICEC04, Delft, Netherlands, 2004.

[BHL01] Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. Scientific American, May 2001.

[BLW04] Balzer, S., Liebig, T., Wagner, M. Pitfalls of OWLS: A Practical Semantic Web Use Case. ACM Press New York, NY, USA, 2004.

[Boo97] Booch, G. Object-Oriented Analysis and Design: With Applications. Addison-Wesley, 1997.

[CiM05] Cimpian, E., and Mocan, A. WSMX Process Mediation Based on Choreographies. In: *1st Int'l Workshop on Web Service Choreography and Orchestration for Business Process Management (BPM 2005)*, 2005.

[DHK+05] Deb, M., Helbig, J., Kroll, M., Scherdin, A. Bringing SOA to Life: The Art and Science of Service Discovery and Design. In *SOA Web Services Journal*, 27 Dec. 2005;
available at http://webservices.sys-con.com/read/164560.htm (2005).

[Ele+05] Elenius, D. et al. The OWL-S Editor – a Development Tool for Semantic Web Services. In *Proc. of the 2nd European Semantic Web Conference (ESWC2005)*, Heraklion, Crete, May 2005.

[FHH03] Files, R., Hayes, P., and Horrocks, I. OWL-QL – a Language for Deductive Query Answering on the Semantic Web. Stanford University, 2003.

[FHH03b] Fikes, R., Hayes, P., and Horrocks, I. OWL Query Language (OWL-QL) Abstract Specification; DARPA Agent Markup Language (DAML) Program; October 13, 2003.

[FHH03a] Fikes, R., Hayes, P., and Horrocks, I. DAML Query Language (DQL) Abstract Specification; Joint United States/European Union ad hoc Agent Markup Language Committee; DARPA Agent Markup Language (DAML) Program; April 1, 2003;
available at http://www.daml.org/2003/04/dql/ (2005).

[Gru93] Gruber, T. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: N. Guarino and R. Poli. Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, 1993.

[Hay05] Hayward, S. Service-Oriented Architecture Adds Flexibility to Business Processes, In: *Positions 2005*, 2005.

[Huh02] Huhns, M.N. Software agents: The Future of Web Services. Technical report, University of South Carolina, Department of Computer Science and Engineering, 2002.

[InT92] Information Technology – Database Languages – SQL". ISO/IEC 9075:1992, American National Standards Institute, New York, N.Y., 1992.

[Ire03] Irek, C. Realizing a Service-Oriented Architecture with .NET. 15 Seconds, 2003; available at http://www.15seconds.com/issue/031215.htm (2005).

[JRM+00] Jaeger, M. C., Rojec-Goldmann, G., Muhl, et al. Ranked Matching for Service Descriptions Using OWL-S. Technical Report, TU Berlin, Institute of Telecommunication Systems.

[KaC03] Karvounarakis, G. and Christophides, V. The RDF Query Language (RQL). Institute of Computer Science, Foundation of Research Technology, Hellas, Greece, July 18, 2003;
available at http://139.91.183.30:9090/RDF/RQL/ (2005).

[Kay03] Kaye, D. Loosely Coupled: The Missing Pieces for Web Services. RDS Associates Inc. 2003.

[KLW95] Kifer, M., Lausen, G., and Wu, J. Logical Foundations of Object-Oriented and Frame-Based Languages. JACM, 42(4), 1995.

[Llo87] Lloyd, J.W. Foundations of Logic Programming. Springer, 1987.

[MaM03] Mandell, D. J., and McIlraith, S. A. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In: *Proc. of the 2nd Int'l Semantic Web Conference (ISWC2003)*, 2003.

[MaM04] Malveau, R., Mowbray, Th. J. Software Architecture: Basic Training. In: *Prentice Hall PTR*, 16 Apr. 2004.

[Mar03] Marchiori M. XML Query (XQuery). World Wide Web Consortium, September 23, 2003;
available at http://www.w3.org/XML/Query (2005).

[MDC+03] Motta, E., Domingue, J., Cabral, L. and Gaspari, M. IRS-II: A Framework and Infrastructure for Semantic Web Services. In: *2nd Int'l Semantic Web Conference (ISWC2003)*. Springer Verlag, October 2003.

[MSZ01] McIlraith, S., Song, T., Zeng, H.: SemanticWeb services. IEEE Intelligent Systems, Special Issue on the Semantic Web 16, 2001.

[NaM02] Narayanan, S., McIlraith, S. Simulation, Verification and Automated Composition of Web Services. In: *Proc. of the 11th Int'l World Wide Web Conference (WWW2002)*, Honolulu, Hawaii, May 2002.

[OLP+04] Olmedilla, D., Lara, R., Polleres, A., and Lausen, H. Trust Negotiation for Semantic Web Services. In: *1st Int'l Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004.

[PaH04] Pan, F., Hobbs, J.R. Time in OWL-S. University of Southern California. 2004.

[Pao02] Paolucci, et al. Semantic Matching of Web Services Capabilities. In: *1st Int. Semantic Web Conference (ISWC)*, Springer, 2002.

[PKP+02] Paolucci, M., Kawamura, T., Payne, R. R., and Sycara, K. Semantic Matching of Web Services Capabilities. ISWC2002, 2002.

[POS+04] Patil, A., Oundhakar, S., Sheth, A., and Verma, K. METEOR-S Web Service Annotation Framework. In: *Proc. of the 13th Int'l World Wide Web Conference (WWW2004)*, 2004.

[ScB05] Scharffe, F., and Bruijn, J. de. A Language to Specify Mappings between Ontologies. In *IEEE SITIS'05*, Yaound´e, Cameroon, Nov. 2005.

[SiH05] Singh M.P., Huhns M.N. Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, 2005.

[SiP04] Sirin, E., and Parsia, B. The OWL-S Java API. Nov. 2004.

[SPS04] Srinivasan, N., Paolucci, M., and Sycara, K. Adding OWL-S to UDDI, Implementation and Throughput. In: *1st Int'l Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, July 2004.

[SPW+04] Sirin, E., Parsia, B., Wu, D., et al. HTN Planning for Web Service Composition Using SHOP2. 1(4), 2004.

[Szy02] Szyperski, C. Component Software: Beyond Object-Oriented Programming. Addison Wesley, 2002.

[VSS+05] Verma, K., Sivashanmugam, K., Sheth, A., et al. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. In: *Journal of Information Technology and Management*, Special Issue on Universal Global Integration, 6(1), Kluwer Academic Publishers, 2005.

Web References

[ACD+03] Andrews, T., Curbera, F., Dholakia, H. et al. Business Process Execution Language for Web Services version 1.1. Specification, May 2003;
available at http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (2006).

[AFM+05] Akkiraju, R., Farrell, J., Miller, J., et al. Web Service Semantics – WSDL-S. W3C Member Submission, Nov. 2005;
available at http://www.w3.org/Submission/2005/10/ (2006).

[ApA06] Apache Axis;
available at http://ws.apache.org/axis/ (2006).

[AWS06] Amazon Web Services;
available at http://www.amazon.de/exec/obidos/tg/browse/-/3516041/302-5490053-8135212 (2006).

[BBD+05] Bruijn, J. de, Bussler, C., Domingue, J., et al. Web Service Modeling Ontology (WSMO). W3C Member Submission, April 2005;
available at http://www.w3.org/Submission/2005/06/ (2006).

[BCE+02] Bellwood, T. et al. UDDI Version 3.0, October 2004;
available at http://uddi.org/pubs/uddi_v3.htm (2006).

[BOE06] Bootstrapping Ontology Evolution with Multimedia Information Extraction;
available at http://www.boemie.org/ (2006).

[CCM+01] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Description Language (WSDL) 1.1., March 2001;
available at http://www.w3.org/TR/wsdl (2006).

[CoM06] CoreMedia AG;
available at http://www.coremedia.com/ (2006).

[DAR06] The DARPA Agent Markup Language Programme;
available at http://www.daml.org (2006).

[DIP06] Data, Information, and Process Integration with Semantic Web Services;
available at http://dip.semanticweb.org/ (2006).

[Ewe06] Electronic Business Using XML (ebXML);
available at http://www.ebxml.org/ (2006).

[InA06] InfoAsset AG;
available at http://www.infoasset.de/ (2006).

[Kno06] Knowledge Interchange Format;
available at http://logic.stanford.edu/kif/kif.html (2006).

[KWe06] Knowledge Web FP6-507482;
available at http://knowledgeweb.semanticweb.org/ (2006).

[LLS06] Landesinstitut für Lehrerbildung und Schulentwicklung, Hamburg;
available at http://www.li-hamburg.de/ (2006).

[MBH+04] Martin, D., Burstein, M., Hobbs, J. et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission, Nov. 2004;
available at http://www.w3.org/Submission/2004/07/ (2006).

[Met06] METEOR-S: Semantic Web Services and Processes;
available at http://lsdis.cs.uga.edu/projects/meteor-s/ (2006).

[MIN06] MINDSWAP: Pellet;
available at http://www.mindswap.org/2003/pellet/ (2006).

[Mod02] Modi, T. WSIL: Do we Need Another Web Services Specification? In: *Web Services Architect*, 16 January 2002;
available at http://www.webservicesarchitect.com/content/articles/modi01.asp (2006).

[MOF04] Meta-Object Facility. Technical report, the Object Management Group, 2004;
available at http://www.omg.org/technology/documents/formal/mof.htm (2006).

[OAS05] The OASIS SOA Reference Model Technical Committee;
available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm (2005).

[ORD06] Oracle Relational Database;
available at http://www.oracle.com/index.html (2006).

[OWL04] OWL Technical Committeee. OWL Web Ontology Language;
available at http://www.w3.org/2004/OWL/ (2006).

[RAC06] RACER Systems. RACER An Inference Engine for the Semantic Web;
available at http://www.racer-systems.com/de/index.phtml?lang (2006).

[RaM06] RacerManager;
available at http://racerproject.sourceforge.net/ (2006).

[RDF06] Resource Description Framework (RDF);
available at http://www.w3.org/RDF/ (2006).

[PLL06] Polleres, A., Lausen, H., Lara, R. Semantische Beschreibung von Web Services;
available at http://members.deri.at/~axelp/publications/poll-etal-2006.pdf (2006).

[San06] Sandia National Laboratories. JESS – The Rule Engine for Java;
available at http://www.jessrules.com/ (2006).

[SEE06] Semantic Execution Environment Technical Committee;
available at http://xml.coverpages.org/OASIS-SemanticEx-CFP.html (2006).

[SEK06] Semantically-Enabled Knowledge Technologies;
available at http://www.sekt-project.com/ (2006).

[Soa03] SOAP Version 1.2 Part 0: Primer. W3C, June 2003;
available at http://www.w3.org/TR/2003/REC-soap12-part0-20030624/ (2006).

[SoS06] Sonic Software;
available at http://www.sonicsoftware.com/index.ssp (2006).

[SpF06] Spring Framework;
available at http://www.springframework.org/ (2006).

[Sys06] Systinet;
available at http://www.systinet.com/ (2006).

[SWS06] Semantic Web Services Interest Group;
available at http://www.w3.org/2002/ws/swsig/ (2006).

[WSM+06] WSMOJ;
available at http://wsmo4j.sourceforge.net/ (2006).

[WSP06] Web Services Policy Framework;
available at http://www-128.ibm.com/developerworks/library/specification/ws-polfram/ (2006)

[WST06] Web Service Modeling Toolkit (WSMT);
available at http://sourceforge.net/projects/wsmt (2006).

[WWF06] W3C Workshop on Frameworks for Semantics in Web Services Summary Report;
available at http://www.w3.org/2005/04/FSWS/workshop-report.html (2006).