Zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften (Dr. rer. pol.) von der Fakultät für Wirtschaftswissenschaften der Universität Karlsruhe (TH) genehmigte Dissertation.

Policy-based Contracting in Semantic Web Service Markets

von

Dipl.-Inform.Wirt Steffen Lamparter

Tag der mündlichen Prüfung: 26.07.2007

Referent: Prof. Dr. Rudi Studer

Koreferent: Prof. Dr. Christof Weinhardt

Prüfer: Prof. Dr. Wolffried Stucky

2007 Karlsruhe

Abstract

Web services generalize the idea of the Web beyond the exchange of simple Web pages in order to enable the provision of a broad range of different services. By composing Web services, cross-organizational and collaborative business processes can be realized in a highly dynamic and flexible way, which is particularly important if services have to be automatically procured at runtime. However, achieving a higher degree of automation is obstructed by the informal nature of legal, contractual and organizational regulations, the numerous and complex service descriptions including manifold customization possibilities, and the open and heterogeneous nature of the Web service market.

In this thesis, semantic technologies that provide more explicit meaning of information are employed to address these problems. These technologies facilitate the exchange of information in heterogeneous systems and increase the share of machine-understandable data accessible for automated decision-making. We introduce the Core Policy Ontology in order to capture regulations as well as preferences by means of goal and utility function policies, respectively. Furthermore, we introduce the Core Ontology of Bids that facilitates customization of Web services to specific user needs by efficiently representing highly configurable Web service offers and requests. Analogously, we derive the Core Contract Ontology from the Core Policy Ontology to formally represent Web service contracts. Thereby, we provide an open, transparent and interoperable representation of contracts and enable a tight integration of contractual information with the collaborative business interactions they govern.

In order to show the applicability of the presented ontologies, we introduce an automated contracting mechanism that includes algorithms for automated matching, allocation, contract formation and contract monitoring. It exploits the semantic descriptions provided by the ontology framework and thereby enables logic-based matching between offers and requests and the specification of policies on hierarchical sets of service characteristics. Since declarative matching and allocation rules are used to define the mechanism, it can be dynamically adapted to new domains or settings. For the efficient allocation of Web services in heterogeneous environments, we present a novel approach that enables the integration of semantic matching and efficient optimization techniques such as linear programming. Moreover, the mechanism can be used to verify whether a Web service invocation adheres to the obligations stated in the contract. The contracting mechanism is prototypically implemented using WS-BPEL and the ontology reasoner KAON2. The evaluation of the prototype indicates that Web service contracting is applicable in practice and that semantic matching of requests and offers is particularly important for settings with highly customizable services.

Acknowledgements

This work would not have been possible without the support and guidance of many persons. First of all, I would like to thank my advisor Professor Dr. Rudi Studer for giving me the opportunity to do this research. Throughout my studies he granted me the freedom, the trust, and the help I needed.

In addition, I would like to thank my co-advisor Professor Dr. Christof Weinhardt for fostering my interdisciplinary research by providing new ideas and insights. Thanks also to the other committee members, Professor Dr. Wolffried Stucky and Professor Dr. Hagen Lindstädt, for their support.

I would like to thank the phenomenal team at the research group LS3WIM and the graduate school IME. They provided the inspiration and constructive criticism that helped me to continuously improve my work. In particular, I am grateful to Anupriya Ankolekar, Sudhir Agarwal, and Andreas Eberhart for guiding my research and for many fruitful discussions and debates. Special thanks also to Saartje Brockmans, Björn Schnizler, Julien Tane, and Raphael Volz who took the time to review chapters of this book and provided me with invaluable comments.

Above all, I am indebted to my family and friends. This work would not have been possible without all their support, encouragement, and confidence during my PhD study and especially during writing of this thesis.

Contents

List of Figures

List of Tables

Part I

Foundations

Chapter 1

Introduction

1.1 Motivation

Service-oriented computing is a paradigm where applications are composed by services. Services are business assets that are exposed by software components provided internally or by other businesses. Service-oriented architectures thus constitute a distributed computing infrastructure for both intra- and inter-organizational application integration and collaboration [PG03]. They abandon the prevailing software paradigm, where applications are installed and executed on local machines. Rather, applications contract other software modules to get certain subtasks completed.

In recent years, technologies for implementing service-oriented architectures have matured into productive systems. Companies are implementing SOA-based applications and hope to gain strategic benefits, such as increased application flexibility, agility and reuse. Since major functionalities of software systems are purchased from other companies, the contracting process in a service-oriented architecture can be seen as a special e-procurement process. According to [GB01], the average procurement cycle in enterprises is of the order of three months. Of this time, about 50% is spent in identifying the appropriate suppliers, about 20% of the time in handling the RFQ (request for quotes) process, and an additional 10% is spent in negotiating the terms and conditions of the contract.

The ability to increase automation in the contracting process could lead to significant time savings and therefore also to cost reductions. In addition, automation of the contracting process is indispensable in case services have to be procured "on demand", i.e. at the point of time when they are required. This is the case whenever selection of a service depends on the execution context, e.g. on the time of invocation or on the current location of the customer. In addition, dynamic contracting enables timely reactions and automatic reconfiguration of the application in case of service failures or frequent changes in the set of available providers. This can lead to more robust systems and to lower costs since erroneous and expensive services can be automatically replaced.

However, automating the contracting process requires overcoming several serious obstacles: (*i*) legal regulations, e.g., involving restrictive data protection rules, and organizational policies that regulate business transactions might have to be considered in the contracting process; (*ii*) the size of the decision problem which might involve a considerable number of providers with each of them offering differenti-

ated services and extensive customization possibilities; (*iii*) the heterogeneity and openness of the service market, which might involve many business partners dynamically joining and leaving the market, each of them with different procurement systems, data formats and information models. The latter has also been recognized by the Harvard Business Review (October 2001):

> "Trying to engage with too many partners too fast is one of the main reasons that so many online market makers have foundered. The transactions they had viewed as simple and routine actually involved many subtle distinctions in terminology and meaning."

Our work addresses these issues by combining and extending technologies and techniques from various fields of research in an original way:

- As a basic technology for implementing service-oriented architectures, *Web services* emerged as the state of the art, providing a set of standard specifications and protocols. Our platform utilizes Web service technologies to realize a Web compliant service-oriented infrastructure.

- Since the find-bind-execute-paradigm of service-oriented architectures is a special kind of electronic procurement process, we rely on concepts known from the area of *electronic markets* to design a Web service contracting process.

- The Semantic Web [BLHL01] addresses the heterogeneity of the environment by providing more explicit meaning of terms. One of the cornerstones of the Semantic Web are ontologies as formal specifications of conceptual models [Gru93]. By committing on common ontologies, different autonomous entities on the Web can interoperate and by leveraging the formal definitions of the ontology constructs, new knowledge can be inferred from existing information. We formalize Web service offers, requests and contracts with *ontologies*. This facilitates interoperability through a standardized syntax and semantics. By means of the underlying logical calculus matching in the market can be improved to handle heterogeneous offer as well as request descriptions. Augmenting electronic markets with ontologies enables the trading of complex services and realizes a degree of automation which would not be possible otherwise [MMW06].

- As decisions automatically taken within the contracting process have to adhere to legal and organizational regulations, this work takes up the idea of *policy-based computing*. In this context, regulations are declaratively captured by policies expressed via ontologies. While featuring management tasks such as consistency checking, ontology-based policy representation facilitates also the exchange of policies contained in offers or requests, which is needed in order to identify possible transactions in the market.

By grounding our work on these four pillars, we extend the state of the art in designing a semantic Web services market that supports an automated contracting process while addressing the heterogeneity that comes with open, Web-based markets. This is realized by the development of an ontology framework that enables the expression of Web service offers, requests and contracts based on a formal policy model. Policies enable us to automate tasks like finding a suitable business partner and

verifying if a transaction has been executed correctly. By extending the traditional policy view that captures only hard constraints to the concept of utility function policies, this work enables preference-based selection of business partners and allows a compact representation of requests and offers. This is particularly important in situations where services need to be customized to the needs of a requester by offering a wide range of different configurations. The formal specification of offers and requests features improved matching functionality by addressing heterogeneity issues in the Web. We are thereby able to overcome a common problem in electronic markets which is caused by different entities using different levels of abstraction for describing service functionality. By utilizing declarative matching and allocation rules, the approach facilitates a high degree of flexibility as the vocabulary can be extended during the runtime of the system and the allocation mechanism can be changed seamlessly without changing the implementation.

Efforts in developing inter-organizational service-oriented infrastructures have intensified considerably within the last years. For the area of automated Web service contracting, this is evidenced by the substantial number of submissions to the World Wide Web Consortium (W3C) proposing semantic descriptions for Web services. Prominent examples are the proposal for Semantic Annotations for WSDL (SAWSDL),[1] Semantic Markup for Web Services (OWL-S),[2] the Web Service Modeling Ontology (WSMO),[3] and the Semantic Web Services Framework (SWSF).[4] This work complements the approaches above by focusing on important aspects in Web service markets beyond pure Web service descriptions. Among others, these aspects include the modeling of customizable offers and requests as well as legally enforceable Web service contracts. We thus use an abstract service description where services are described using a set of attributes. Such a general description of Web services enables us to abstract from various existing Web service description frameworks such as WSDL, OWL-S, SAWSDL, WSMO, while simultaneously allowing us to leverage existing decision-theoretic algorithms for multi-attribute products. However, our approach allows the representation of attributes using existing service description ontologies and thereby enables the reuse of existing work.

1.2 Research Questions and Goals

Realization of an inter-organizational service-oriented computing infrastructure has to address technical issues such as implementing and describing Web services, economic questions like selecting the right service at an acceptable price, and legal problems dealing with automated contract conclusion and interpretation. One of the goals of this thesis is to provide a better insight into how an interdisciplinary approach drawing from the fields of computer science, economics and law can be used to build a service-oriented computing infrastructure. The following hypothesis captures the main research question of this thesis.

Main Hypothesis: *Contracting in Web service markets can be automated using semantic policy descriptions.*

[1] W3C Working Draft, April 2007, `http://www.w3.org/TR/sawsdl/`
[2] W3C Submission, November 2004, `http://www.w3.org/Submission/OWL-S/`
[3] W3C Submission, April 2005, `http://www.w3.org/Submission/2005/06/`
[4] W3C Submission, September 2005, `http://www.w3.org/Submission/SWSF/`

As discussed in the previous section, the major obstacles for a higher degree of automation in the contracting process between Web service providers and requesters are legal and organizational regulations that are often specified in an informal way, numerous and complex service descriptions including manifold customization possibilities which makes manual selection cumbersome, and the open and heterogeneous nature of the Web service market hinders collaboration between different parties. Semantic technologies address these problems by providing more explicit meaning of information. This facilities exchanging of information in heterogeneous systems and increases the amount of machine-understandable data required for automated decision-making.

In order to support our main hypothesis, we investigate in this thesis how ontologies can be applied in order to realize automated contracting between Web service providers and their customers. Since electronic markets generally require the design of two components – language and mechanism – we split our main hypothesis into two subordinate hypotheses for each of which an approach how to support the hypothesis is given.

Hypothesis 1: *Semantic technologies can be used to express policies such that they enable the specification of offers, requests and contracts in Web service markets.*

Approach: *Develop an expressive Web service market ontology for representing Web service offers, requests and contracts in a formal machine-interpretable way.*

The first hypothesis postulates that semantic technologies can be applied to design a communication language for the exchange of knowledge about products and prices in the market. This requires a language that allows capturing legal and organizational regulations as well as extensive customization possibilities in a machine-understandable manner. To support this hypothesis, we develop an expressive Web service market ontology and thereby show that semantic technologies are a suitable technology for this purpose.

Hypothesis 2: *The contracting process in the market can be automated based on semantic descriptions.*

Approach: *Develop algorithms that automate the contracting process based on the Web service market ontology and that provide the flexibility to cope with environmental changes.*

The second hypothesis postulates that semantic technologies can be used to automate the contracting process. We are going to support this hypothesis by designing algorithms for automated matching, allocation, contract formation and validation based on the previously defined ontology.

This means, in order to realize a semantic Web service infrastructure we first have to develop an ontology that provides the expressivity to formalize all required information in the market and second we have to design the contracting algorithms in a way that they utilize the formalized knowledge and can be executed without human intervention.

1.3 Contributions

The main contribution of this work is the design and realization of a service-oriented computing infrastructure that builds on existing Web service technology and enables the automation of the contracting process. As indicated by our research questions, realizing such an infrastructure requires two main components which are provided in this thesis:

- We present a novel ***ontology framework for Web service markets*** that enables the formal representation of market information. The framework thereby supports the interpretation of this information by machines and the exchange of information between different market participants in the Web.
- We present a ***contracting mechanism*** based on this ontology framework that automates the matching of Web service offers and requests, the determination of optimal allocations between offers and requests, and the conclusion and monitoring of Web service contracts.

The automation of these tasks allows the procurement of services "on demand" and thus addresses several major shortcomings of today's service-oriented architectures, such as the inability to consider the execution context or to react on service failures or a changing set providers. Moreover, the time for integrating a new service into the architecture can be reduced, which may lead to major cost savings for the service requester. In the following, the novel aspects of the ontology framework and the contracting mechanism are discussed in more detail.

Web Service Market Ontology

To realize the goal of automation, we have developed a set of ontology modules that enables the formal and unambiguous representation of market information. In particular, the work contributes the ***Core Policy Ontology***, the ***Core Ontology of Bids*** and the ***Core Contract Ontology***. The Core Policy Ontology is novel in that it enables not only expressing hard constraints in a formal and declarative manner, but also fine-grained preferences over alternatives captured by the concept of ***utility function policies***. By reusing these policies in the Core Ontology of Bids highly ***configurable offers and requests*** can be expressed in a compact way which allows us to communicate them efficiently within the market. The fact that services are easily customizable, e.g., by differentiation of quality of service levels, is an issue which has by far been neglected by existing approaches. Moreover, policies can be specified depending on the context in which they are applicable, which also enables automation in the presence of ***context-dependent preferences*** as observable particularly in mobile scenarios. In extending the DOLCE foundational ontology library and by utilizing ontology design patterns, the modules provide a ***high-quality ontology framework*** that circumvents typical shortcomings of naively built ontologies, such as conceptual ambiguity, poor axiomatization, narrow scope and loose design [Obe05].

Contracting Mechanism

As a further contribution to the state of the art, we present a contracting mechanism that exploits the declarative semantic descriptions provided by the ontology framework in several ways: the heterogeneity that arises from a constantly changing number of autonomous market participants is addressed by ***logic-based matching of offers and requests***. This allows us to overcome the different levels of abstractions in offer and request description that are usually observed in markets. In contrast to other logic-based matching approaches, we additionally show how such techniques enable defining ***policies on hierarchical sets of service characteristics***. This means, it is not required to specify preferences for all possible configurations, but the valuation of a market participant for a certain configuration can be inferred from general policies defined on a higher level. This simplifies the definition of policies considerably. Another drawback of existing semantic approaches is that they assume a fixed, predefined set of matching algorithms that is applied irrespective of the domain, application, or the characteristics of the underlying ontology (cf. [PKPS02, LH03, NSDM03, GMP04]). This is only sensible under the assumption that all ontologies are specified with the corresponding matching algorithm in mind. However, this contradicts the basic idea of the Semantic Web, where domain ontologies are to be reused by several applications in order to reduce the modeling efforts that have to be devoted to building ontologies. Therefore, this work relies on the idea of ***customizable matching rules*** that can be declaratively defined for each domain ontology. Based on these rules, the right matching approach is applied automatically. The declarative nature of the matching rules enables adding of new service characteristics and required domain ontologies during runtime of the system, which is essential for providing the required flexibility.

In order to allow for dynamic contracting at runtime the contracting algorithms have to be executed within a short period of time. In this work we thus present a ***computational tractable selection mechanism***, which could be used as basis for more complex allocation mechanisms such as auctions. Developing such mechanisms essentially requires an algorithm that selects the optimal configuration for a provider/requester pair. This has to involve semantic matching of services and ranking according to utility function policies. To realize this in a computationally tractable manner, we propose a novel approach that ***integrates semantic matching and efficient optimization techniques*** such as linear programming. This allows us, on the one hand, to benefit from existing efficient optimization tools developed over the last decades in the field of operations research, and on the other hand, to gain from the flexibility and expressivity provided by semantic technologies. Assuming additive utility function policies, experimental results indicate that our algorithm introduces an overhead of only around 2 sec. compared to random service selection, while giving optimal results. The overhead, as percentage of total time, decreases as the number of offers and configurations increase. Moreover, our experiments indicate that applying semantic matching in Web-based markets increases the utility of the participants.

In addition to the matching and allocation algorithm, this work presents a method for ***semi-automated contract formation, execution and monitoring***. Since full automation is not achievable based on current (German) law, we propose a semi-automated approach which originally combines a manually concluded umbrella contract with an individual contract automatically closed for each invocation. Based

on this formalization, the contract management tasks can be automated such as contract execution or monitoring. This is especially important for configurable services where each contract might be different. In order to address fuzzy interpretations of contract clauses, ***interpretation rules*** are introduced to allow the monitoring of such aspects.

As a proof of concept, we finally present a ***prototypical implementation*** that shows how the developed techniques can be applied based on current Web service infrastructure. Here we discuss how dynamic binding of Web services is realized for a WS-BPEL process using conventional WS-BPEL engines.

1.4 Reader's Guide

This thesis is structured as follows. In *Part I: Foundations* the fundamental ideas and concepts are introduced. First, in Chapter 2 the four technologies that constitute the cornerstones of our work are presented: service-oriented architectures, policy-based computing, electronic markets and ontologies. In Chapter 3, we discuss how these technologies can be integrated in terms of a design and engineering methodology. According to this methodology, the development of a semantic Web service market infrastructure comprises the following steps: requirements analysis, design, embodiment, implementation, and testing. The subsequent parts and chapters are structured according to this engineering process.

In *Part II: Designing a Semantic Web Service Market* we describe the design process of developing a semantic Web service market. As a first step, the requirements are elicited from a set of typical scenarios for service-oriented architectures in Chapter 4. The scenarios cover enterprise, mobile and grid service applications. The requirements are clustered into language and mechanism-specific requirements. In Chapter 5 the conceptual design of the market model is introduced in an abstract way without discussing concrete technologies or other implementation details. The market model addresses the language-specific requirements by introducing an appropriate conceptualization and formalization of the ontology and the mechanism-specific requirements by specifying the Web service contracting algorithms. In the subsequent embodiment phase this abstract conceptual model is explicitly specified: Chapter 6 presents an ontology framework which implements the conceptualization introduced in the conceptual design. The framework introduces the novel modules Core Policy Ontology, Core Ontology of Bids and the Core Contract Ontology. Based on this framework concrete contracting algorithms are presented in Chapter 7, which includes matching, allocation, contract formation as well as contract monitoring functionality.

The implementation and testing steps of the engineering process are covered in *Part III: Realization and Evaluation*. In Chapter 8 a prototypical implementation of a semantic Web service market infrastructure is presented that features automated contracting of Web services and supports dynamic binding of services in a business process. In addition, two concrete applications of the prototype are given. In Chapter 9, the design of the market is discussed with respect to the requirements that have to be met. In this context, computational tractability and communication efficiency are evaluated by means of a simulation.

Part IV: Finale consolidates the language- as well as mechanism-specific related work in Chapter 10 by discussing how other approaches address the requirements. Finally, Chapter 11 concludes the work by recapitulating the results with respect to our research questions and gives an overview of problems that remain open and have to be addressed in future work.

Throughout the work, relevant publications are given at the beginning of the chapters. Bits and pieces of the thesis are based on conference and journal publications [LEO05, LML+05, LA05, LS06, LAO+06, OLG+06, LA07, LAGS07].

Chapter 2

Basic Concepts and Technologies

In this chapter, we introduce the fundamental definitions and technologies required throughout the thesis. We start with the notion of *service-oriented architectures* in Section 2.1, which provides a powerful paradigm for developing flexible and interoperable software systems. Due to their complexity and dynamics, the design, management and administration of service-oriented architectures is difficult and time-consuming. By introducing *policies* in Section 2.2, a higher degree of autonomy can be realized and thus reduced human interaction is required. Recently, electronic markets have been proposed as an efficient coordination mechanism between service providers and service requesters. Therefore, Section 2.3 introduces the idea of *markets*. Subsequently in Section 2.4, the concept of *ontologies* is introduced. Ontologies provide a formal vocabulary for knowledge sharing in distributed systems. Within service-oriented architectures they are typically used for service discovery and mediation between different heterogeneous services. In addition, they enable the use of market mechanisms for complex product.

2.1 Service-oriented Architectures

Service-oriented architectures (SOA) come in many different forms and are implemented by means of various technologies. In Section 2.1.1, we first introduce our notion of a service-oriented architecture independent of technologies and application areas. In Section 2.1.2, two main concepts enabled by service-oriented architectures are presented: flexible binding of services and service customization. Since *Web services* have become the predominant technology and a quasi-standard for implementing service-oriented architectures in an inter-organizational context, we introduce the technology behind Web services in Section 2.1.3.

2.1.1 Basic Principles

Service-oriented architectures have received considerable attention throughout the last years which has led to several highly diverse definitions. In our work, we follow the definitions of the OASIS Reference Model for Service Oriented Architectures [MLM+06], which is an emerging standard clarifying the significant entities and their relations in a service-oriented architecture.

Before we can define the term service-oriented architecture, the notion of *service* used intuitively up to now has to be clarified. A service is defined as an "act or a

variety of work done for others".[1] We call activities that solves someone's problem *capabilities*. According to this definition, a service might cover a wide range of capabilities ranging from constructing a house to trading stocks, each of them carried out for someone else. An entity (people or organizations) offering a service is called *provider* and an entity that initiates the service execution and profits from the service ia called *requester*. Both – providers as well as requesters– define certain conditions or regulations on the usage of the service.

In the context of service-oriented architectures, only a subset of services – called *software services* – are relevant. Software services are software components that provide certain capabilities via electronic media such as the Internet. On the one hand, these can be purely digital services where the capabilities are provided entirely in electronic form. Examples are services that provide stock quote information via e-mail or a route planning service on the Web. On the other hand, software services can also cause real world effects, such as a travel booking service that sends an acknowledgement of the booking by e-mail, but delivers the actual tickets by surface mail. In order to be usable for requesters, a software service has to adhere to a prescribed *service interface* that can be used to integrate the software service into the requester's application. For this work the following definition of software services is adopted:

Definition 2.1 (Software Service) *A software service is a mechanism to enable access to one or more capabilities provided by an encapsulated software component via an electronic medium. The provider installs, runs, maintains, and evolves hardware as well as software infrastructure and provides all physical and organizational means. The access is provided by a prescribed and well-defined programmatic service interface and is consistent with the provider's constraints and conditions.*

Note that not every service available through an electronic medium is a software service. A set of Web pages that allow, e.g., the reservation of a table in a restaurant is a service, but usually not considered as a software service, according to Definition 2.1, since it does not provide a programmatic interface that can be used to invoke the service from a software program [ACKM04].

The technique of modularization and goal-oriented composition of software services can be seen as a distinguishing aspect of service-oriented software development compared to traditional software development [PG03]. Generally, a *service composition* defines which software services are used in which order by an application. Along this line, a service-oriented architecture can be defined as follows:

Definition 2.2 (Service-oriented Architecture) *A service-oriented architecture is a software design where a* reusable *set of* interoperable *and* discoverable *software services is* loosely-coupled *in order to realize a distributed application.*

We next describe the key principles of service-oriented architectures in more detail:

Loose-coupling. Loosely-coupled relations between two services minimize the dependencies between the two ends. This can be realized by using message-oriented communication and encapsulation of implementation and business

[1] http://www.thefreedictionary.com/service

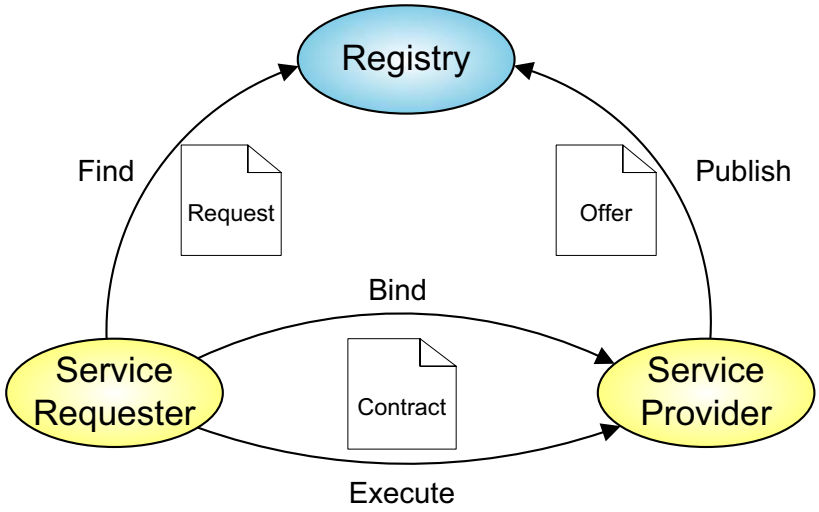

Figure 2.1: Illustration of the Publish-Find-Bind-Execute Paradigm

logic details with clear interfaces. This allows a maximal degree of autonomy for each service and leads to a separation of concerns.

Interoperability. Interoperability is the capability to communicate or transfer data among the services in the system with little or no need for manual adaption. Typically interoperability in service-oriented architectures is approached by means of standardized communication protocols and languages.

Reusability. Each service is self-contained in a sense that capabilities can be used in different business processes or for different purposes. In this situation, applications are built by composing existing modular services.

Discoverability. A prerequisite for the reuse of services is their discovery. Services have to be discoverable either manually by the application developer or automatically by the system. Discoverability is usually ensured by adding a service repository component to the service-oriented architecture.

Additional principles such as service encapsulation or abstraction can be found in the literature [Erl06]. However, these are direct consequences of the principles discussed above.

These design principles enable an easy reorganization of services and flexible implementations of business processes reflecting the fact that business processes are often much more volatile than the information they manipulate, i.e. while process typically change frequently, the service from which they are composed are rather static and can be reused several times. This idea is captured by the following statement [Bur05]:

"*Design Services to Last, Design Systems to Change.*"

To realize this flexibility, a service-oriented architecture is based on the *publish-find-bind-execute* paradigm, which is illustrated in Figure 2.1. A service provider *publishes* a *service offer* using a *registry* and makes the service discoverable and thus reusable for requesters. The offer contains information about the interface of the service and the constraints and conditions under which the service may be accessed. These

constraints and conditions are usually called *policies* [MLM+06, W3C06b]. When a requester requires a service, a *service request* is sent to the registry. This request describes the requirements in terms of service interface and policies, which have to be met by the service offers to be qualified as suitable service candidates. Depending on the concrete implementation of the registry, either the set of suitable offers or a concrete *service contract* is returned. A contract represents a concrete agreement between the requester and the provider fixing the agreed interface and service levels in a well-defined and unambiguous manner. If the registry returns suitable offers instead of contracts, a contract has to be closed in the binding process. A *binding* is a unidirectional assignment of a task in a requester's business process to a concrete service. The flexibility of the business process and the ability to adapt to changing requirements depend crucially on the binding mechanism used. Therefore, in Section 2.1.2, different binding mechanisms are discussed in more detail. Once the best binding is determined, the requester *executes* the service by sending the input data required by the service interface. Requests, offers and contracts are key concepts in our work and will be formally defined in Section 5.

2.1.2 Flexible Binding and Customization of Services

In today's business environment there is an urgent need for flexible software systems that can be easily adapted to fast changing requirements. Consider the example where a drastic increase in the number of users might require a quick increase in the scalability of an application, or where a merger requires the fast integration of two independent software systems. By leveraging the loose coupling of services in a service-oriented architecture, this flexibility can be provided in two ways: first, we can reconfigure a service and adapt it to our needs. We call this reconfiguration of services *service customization*. Alternatively, we can dynamically replace a service in the business process by another provider. We call this feature *flexible binding* of services. These two alternatives are briefly discussed in the following.

Service Customization

A key concept in economics and management is *product differentiation* or *versioning*. The idea behind product differentiation is to provide a certain product in such a way that it differs from the products of the competitors in the market with the intent to influence the demand. Thereby, suppliers can decrease the substitutability of their product which increases their monopoly power [BKK+02], and they can provide a version customized to often very heterogeneous requirements of customers which could significantly increase the revenue of the supplier [Var97]. Product differentiation is usually realized by exploiting customer *self-selection*. For example, it can be distinguished between feature-based and performance-based differentiation [Ded02], where either a product is available with different features or with the same features but different quality. All of these versions are offered to customers who select the version most suitable to them. Thereby, the utility of the supplier and customer can be increased.

The idea of product differentiation can be directly transferred to software services in a service-oriented architecture. In fact, *service differentiation* is particularly easy, since it can be realized simply by providing different quality-of-service guar-

antees, by forwarding requests to different service implementations (e.g. each with different performance characteristics), or by assigning different priority levels to requests (e.g. requesters paying a higher price are served first). Examples of service differentiation can be found in [ZWX06, CM02, WBCL02, DLP03]. In line with [Bro98, BAG03], we call these different versions of a certain service *configurations* throughout our work. Initially, we will only provide an intuitive definition of the term, which is then made concrete in Chapter 5 as part of our formal model.

Definition 2.3 (Service Configuration) *A service configuration selects a value for each attribute of a service and thereby unambiguously defines all relevant service characteristics. The choice of configuration might affect the functional as well as non-functional aspects of a service and is a major determinant of the price.*

Obviously, providers as well as requesters have certain constraints and conditions with regard to the allowed configurations. For instance, a requester might have minimum requirements regarding quality of service. We represent such constraints by means of policies, which are introduced in Section 2.2. In the following, we first discuss the case where reconfiguration of individual services is not sufficient or possible.

Flexible Binding

If adaption of the business process is not possible through reconfiguration of individual services, entire services have to be replaced. As defined in the Section 2.1.1, a process sequences certain tasks which are executed by certain software services. The assignment of tasks to the services that carry out this task is called a *binding*. Bindings can be specified in three different ways [PA05]:

Binding by Inclusion. In this case services are statically bound in the composition by inclusion. That means a binding is explicitly set to the address of an service. Such hard-wiring can be considered as the state-of-the-art in today's service oriented architectures.

Binding by Reference. In this case the composition is linked to an external service description, which then in turn refers statically to a service. This approach separates binding from composition and increases flexibility. For example, the address of a service can be changed without changing the composition.

Binding by Constraint. While in case of "binding by reference" compositions still uniquely identify services, "binding by constraint" abolish this static assignments. In fact, this approach distinguishes between the composition and the set of suitable services. The composition only defines which criteria (e.g. conditions on the interface and policies) a suitable service has to meet.

If using the "binding-by-constraint" paradigm, an *evaluation* will be required that determines the concrete binding. Based on requests and offers, the evaluation has to calculate the set S of suitable services and select one service $s \in S$. This evaluation process might also require customization of the selected service.

A major advantage of this approach is that this evaluation can be done at different stages in the software development. We distinguish between *dynamic* and *static*

binding.[2] While static binding can be used for all three binding paradigm mentioned above, dynamic binding is only possible if the "binding-by-constraint" paradigm is used. Static and dynamic binding is defined as follows:

Static Binding. In case of static binding the evaluation is done at development time. The development time comprises the composition, compilation and deployment of process. That term "static" arises from the fact that a deployed composition is executed always with the same services.

Dynamic Binding. In contrast, dynamic binding evaluates the constraints at runtime of the composition, i.e. during execution of the application. Pautasso et al. [PA05] distinguish further whether the binding is done at startup time, invocation time or failed invocation time. If binding is done at startup time, all services are bound before the composition is executed. In doing this, one can make sure that for all tasks an appropriate service is available. Evaluation at invocation time is the latest possible time before the service is invoked. Failed invocation time refers to the strategy where new bindings are only determined if the current binding fails, i.e. the binding refers to a service that does not react or is not available any more. Consequently, only in this case is an evaluation necessary. That means the dynamic binding is "dynamic" in the sense that each time the composition is executed, other services might be used.

Conceptually, using dynamic binding in service-oriented architectures provides considerable advantages: In many scenarios, the decision which service to invoke depends on runtime-specific aspects such as the current location of the requester or the time of execution. In such a context, binding at development time is simply not possible. Moreover, the set of available services may change frequently after a composition has been deployed. In this case dynamic binding is required to be able to react to these changes. However, a major challenge to realize dynamic binding remains open: modeling the constraints C that can be used to determine the most suitable configuration and binding. Before we revisit the problem of constraint representation in Section 2.2 by introducing the concept of policies, the current state-of-the-art for implementing service-oriented architectures is introduced.

2.1.3 Web Service Technology

Web service are a new form of middleware that enable the integration of computer programs across application and organization boundaries. The basic idea of conventional middleware such as Remote Procedure Calls (RPC) or Object Brokers was to reside between the applications to be integrated and to mediate their interactions [ACKM04]. While allowing distributed applications, these middleware systems were (logically) centralized and controlled by a single company. For the implementation of service-oriented architectures in an inter-organizational setting such solutions are not appropriate. They require agreements on a specific middleware platform as well as on a "global workflow" for the entire business process. This is very unlikely to happen due to the lack of trust between companies, the autonomy each company wants to preserve, and the confidentiality of the business transactions and processes.

[2] The term *early* and *late binding* are also used to refer to static and dynamic binding, respectively.

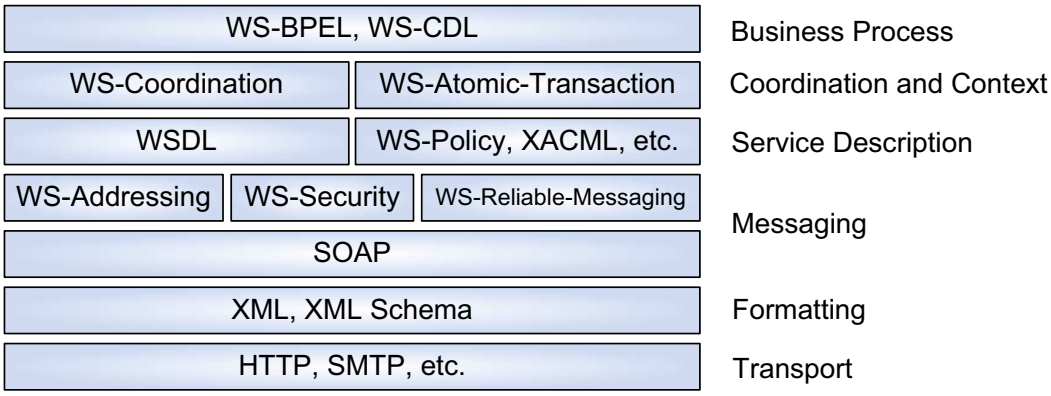

Figure 2.2: Overview of Web service technologies

When moving from intra-enterprise application integration to inter-organizational structures the *Web* aspect becomes important. Web technologies provide the basic protocols such as HTTP and information encoding mechanism like the eXtensible Markup Language (XML) [W3C04a]. Due to their standardization, they provide a key ingredient for application integration in a B2B setting. The first step towards "Web-enabled" middleware are *application servers* [Obe05]. However, they require tight integration of the distributed components and thus do not support the loose-coupling aspect of service-oriented architectures. Web services extend the concept of "Web-enabled" middleware by factorizing software functionality in loosely-coupled services that communicate via the Web and provide a well-defined programmatic interface. In line with the definition provided by the World Wide Web consortium (W3C) [W3C04d], we adopt the following notion of Web services.

Definition 2.4 (Web Service) *A Web service is a software service identified by a Uniform Resource Identifier (URI) [RFC05], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.*

This definition stresses the key aspects of implementing service-oriented architectures as manifested in Definition 2.2: for discoverability and reusability, a definition prescribing the service's interaction is required. To provide interoperability in the Web URIs for identification and the usage of standardized XML and Internet protocols is required. Loose-coupling is supported by requiring message-orientation and encapsulation of functionality behind interfaces.

As standardization plays a major role, a great number of Web service specifications provide (often alternative) proposals for enabling interaction and description of Web services. The specifications are arranged in six layers where each layer requires (at least a partial) implementation of the subjacent layers. Figure 2.2 relates the layers with the respective specification belonging to these layers. Note that for each layer further specifications exist. However, since they are not relevant for the understanding of the remainder of the thesis, we omit a detailed introduction at

this point.[3] In the following, we gradually introduce the layers starting with the Transport layer.

Transport Layer

The Transport layer defines methods that are used to transfer or convey information in the Internet. Although the Hypertext Transfer Protocol (HTTP) [RFC99] is by far the most prominent protocol for communication between two Web services, other protocols such as the Simple Mail Transfer Protocol (SMTP) are also used. Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, URLs) using the "http:" or "https:" URI schemes.

Formatting Layer

As stated in Definition 2.4, messages exchanged between Web services and the descriptions of Web services have to be encoded using XML documents, which can be validated according to a schema expressed via XML Schema (or DTD).

Messaging Layer

Messaging specifications are intended to give a framework for exchanging information in a decentralized, distributed environment. Messages that can be understood by Web services have to be organized according to the Simple Object Access Protocol (SOAP) [W3C03]. SOAP describes how documents are encoded using XML, provides conventions for the interactions between different peers, and defines how messages should be transported on top of HTTP or SMTP. There are also alternative protocols such as REST. However, they are rarely used in practise. With specifications such as WS-Security [OAS06b] additional functionality can be added to the messaging layer. WS-Security is an extension to SOAP that allows implementing integrity and confidentiality.

Service Description Layer

The focus of this layer is the definition of specifications that support the description and discovery of Web service providers, the Web services they make available, and the technical interfaces for accessing and using these services. In this context, the Web Service Description Language (WSDL) [W3C01b] plays a central role. WSDL are XML documents defined via a XML Schema consisting of an abstract and a concrete part. In the abstract part the service interface is described by means of *port type* definitions which are logical collections of related operations. For each operation the *data types* of the input and output messages are defined. Although WSDL allows the specification of arbitrary data type systems, usually the XML Schema data type system is used [W3C04e]. In the concrete part this abstract port type definition is bound to a concrete message encoding and protocol. In addition, a concrete *end point* address (specified by a URI) is attached to each port type and end points are grouped in a service element.

[3] For a comprehensive overview the interested reader is referred to `http://www-128.ibm.com/developerworks/views/webservices/libraryview.jsp?type_by=Standards`.

Since WSDL lacks expressiveness for describing requirements and conditions a Web service or a requester has to fulfill for a successful interaction, additional specifications are needed. WS-Policy [W3C06b] provides a framework through which requesters as well as providers can specify their policies. The framework is domain-independent and requires further specification for providing domain-specific vocabulary.

Coordination and Context Layer

Transactions are a fundamental concept in building reliable distributed applications. A Web service environment requires coordination behavior provided by a traditional transaction mechanism to control the operations and outcome of an application. Examples for specifications belonging to this layer are WS-Coordination and its extension WS-Transaction.

Business Process Layer

A business process specifies the potential execution order of operations from a collection of Web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, and other issues involving how multiple services and organizations participate.

The *Business Process Execution Language* (WS-BPEL) [ACD+03] is a XML-based language used to define business processes, where each task or operation is assumed to be implemented as a Web service. The key objective of WS-BPEL is to standardize the format of business process flow definitions so that companies can work together seamlessly using Web services. The WS-BPEL notation includes flow control, variables, concurrent execution, input and output, transaction scoping/-compensation, and error handling. Processes written in WS-BPEL can orchestrate interactions between Web services using XML documents in a standardized manner. These processes can be executed on any platform or product that complies with the WS-BPEL specification.

Listing 2.1 shows a simple loan approval business process specified using WS-BPEL. Loan approval is a scenario commonly used in related literature on this topic [Kha02, MM03, Act06]. In line 1-6 the business process definition specifies the name of the process and the namespaces required for the remaining document. The actual operations of the business process are defined between the *flow*-tags in line 7 and 20. The first operation called *request* is of type *receive* (lines 8-10) and initiates a new process instance whenever a message from a *customer* is received. The input of this *customer* is stored in a variable called *request*. This variable is then passed to the subsequent *invoke*-operation (lines 11-15) which uses this information as input for invoking the operation *check* of a Web service providing risk assessment functionality. The result of the service invocation is passed to the next operation (line 16-18) using the variable *riskAssessment*. This operation is of type *reply* and returns the result of the risk assessment to the customer.

BPEL supports two different types of business processes:

- Executable processes: Models the actual behavior of a participant in a business interaction. They can be executed by an BPEL engine.

```
 1 <process name="loanApprovalProcess"
 2  targetNamespace="http://ontoware.org/loanprocessing"
 3  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 4  xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 5  xmlns:lns="http://ontoware.org/wsdl/loan-approval"
 6  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 7   <flow>
 8    <receive createInstance="yes" operation="request"
 9     partnerLink="customer" portType="lns:loanServicePT"
10     variable="request">
11    </receive>
12    <invoke inputVariable="request" name="invokeAssessor"
13    operation="check" outputVariable="riskAssessment"
14    partnerLink="assessor" portType="asns:riskAssessmentPT">
15    <target linkName="receive-to-assess"/>
16    </invoke>
17    <reply operation="request" partnerLink="customer"
18     portType="lns:loanServicePT" variable="approval">
19    </reply>
20   </flow>
21 </process>
```

Listing 2.1: WS-BPEL process.

- Abstract processes: Uses process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior.

2.2 Policy-based Computing

Generally, the notion of *policy-based computing* refers to a software model that incorporates a set of decision-making technologies into its management components in order to simplify and automate the administration of computer systems" [Mur05]. This is achieved by interpreting policies at runtime to make autonomous decisions as desired by the policies' author. Policies represent the goals, constraints, requirements, or conditions of the system administrator that guide the decisions of a system. Thereby, they state the objective/desired behavior, but they do not specify the way how this should be accomplished. A major advantage of policy-based computing is the separation of the components responsible for managing the system and the guidelines defining the desired system behavior. This facilitates manageability, while providing a high degree of flexibility. In the area of computer science and artificial intelligence various definitions of the term policy can be found in literature (e.g. [MLM+06, RFC01, Slo94, KW04]). Most of them share the commonalities captured by the following definition.

Definition 2.5 (Policy) *A policy represents some constraint or condition on the use, deployment or description of an owned entity and thereby guides the behavior of an autonomous decision maker (agent). Policies are expressed with a declarative, machine-*

interpretable formalism that enables automated decisions, policy changes at runtime and communication of policies to other decision makers.

Definition 2.5 captures a crucial property of a policy formalism: they have to be automatically enforceable by the system each time a decision has to be made. Thus, decision can be transferred from a human decision maker to the system level, while ensuring conformance with the human counterpart. This is one of the core ideas in the emerging field of *autonomic computing*, which is briefly introduced in Section 2.2.1. Subsequently, we discuss how these concepts can be transferred to service-oriented architectures and then introduce the fundamental policy types that have been suggested in literature.

2.2.1 Autonomic Computing

The final goal of autonomic computing is to introduce "computing systems that can manage themselves given high-level objectives from administrators" [KC03]. By enabling *self-management* functionality policy-based computing is therefore a core technology for implementing such systems. Self-managing systems maintain and adjust their operations in the face of changing environmental states (e.g. workload changes) and in the face of hardware or software failure. Often four aspects of self-management are distinguished: self-configuration, self-healing, self-optimization and self-protection [KC03]. We shortly introduce these aspects in the following.

Self-configuration. Self-configuration is a feature that enables a software system to configure itself, e.g., to different platforms or vendors in accordance to high-level policies. The goal is that the system will adjust itself automatically if new components are incorporated or the system is transferred to another platform. Since installing and updating major applications is very time-consuming, such functionality can greatly facilitate system management.

Self-healing. Manual diagnosing and fixing of failures in large and complex computer systems is very tedious and system support is required. Self-healing addresses this problem by providing means for detecting, diagnosing and repairing failures automatically. For example, this could comprise analyzing monitoring information such as log files, recognize failures and install patches or alert the human administrators.

Self-optimization. Large computer systems usually have hundreds of parameters that have to be set correctly to enable the system to perform optimally. Therefore, a self-optimization functionality is required that automatically seeks opportunities to improve system performance and efficiency. This could be for example realized by simulating different settings and measuring their efficiency and adapting the parameter values accordingly.

Self-protection. Since companies realize more and more vital business activities with computer systems and attacks become more frequent, system security becomes increasingly important and hard to guarantee. Self-protection mechanisms can address this problem by taking automatic defense measures against malicious attacks and issue early warnings to avoid system-wide cascading errors.

In the next section, we discuss how the idea of autonomic computing carries over to complex software system implemented using service-oriented architectures. In particular, we discuss how service-orientation features the different aspects of self-management.

2.2.2 Policies in Service-oriented Architectures

When applying the concept of policy-based computing to service-oriented architectures, the management and administration of the service-oriented architecture– often referred to as *SOA Governance* – can be (at least partially) delegated to the system itself and thus reduce management effort. This requires that administrators define appropriate policies how the system should behave. Since the behavior of a SOA-based system is mainly determined by the question which task of the business process is executed by which service, the major problem in SOA Governance is the management of these bindings. In this context, self-manageability can be realized by assigning services automatically according to policies reflecting the companies' business objectives, regulative norms, such as Sarbanes-Oxley[4], or IT-Governance standards (e.g. ISO 20000[5]). An explicit specification of such policies makes sure that the overall behavior of the software system will be in line with the companies' high-level objectives and regulations, while many low-level decisions can be automatically done by the system without human intervention. Thereby, the management effort can be reduced considerably.

As introduced in Section 2.1.2, by featuring the binding-by-constraint paradigm, service-oriented architectures support flexible assignments of business process tasks to available services. This is an important property for implementing self-manageable software systems, since it enables the system to discover or replace services itself as required. To enable a flexible binding mechanism, the constraints that have to be met by all services are the set of policies defined by the company. Consequently, constraint evaluation can be directly realized by the policy enforcement mechanisms. In this sense, the creation, communication and enforcement of policies are a central part of SOA Governance.

For example, self-healing functionality can be realized by a dynamic binding mechanism, where a faulty service is replaced by an alternative service complying with the administrator's policies. In the same line, self-optimization and self-configuration can be realized by replacing a service once a better service or a better service configuration is available in the system. Thus, policy enforcement has to feature compliance checking as well as rating of services for different degrees of optimality. Not all kinds of policy languages are expressive enough to support this. In Section 2.2.3 we introduce a policy classification scheme providing a coherent framework to distinguish the different types of policies suggested in literature.

2.2.3 Policy Classification Scheme

In recent years, several policy specification languages have been proposed addressing a wide range of different purposes. In this section, we introduce a policy classi-

[4]Sarbanes-Oxley Act 2002, available at `http://www.legalarchiver.org/soa.htm`

[5]ISO 20000 IT Service Management Standards, available at `http://20000.standardsdirect.org/`

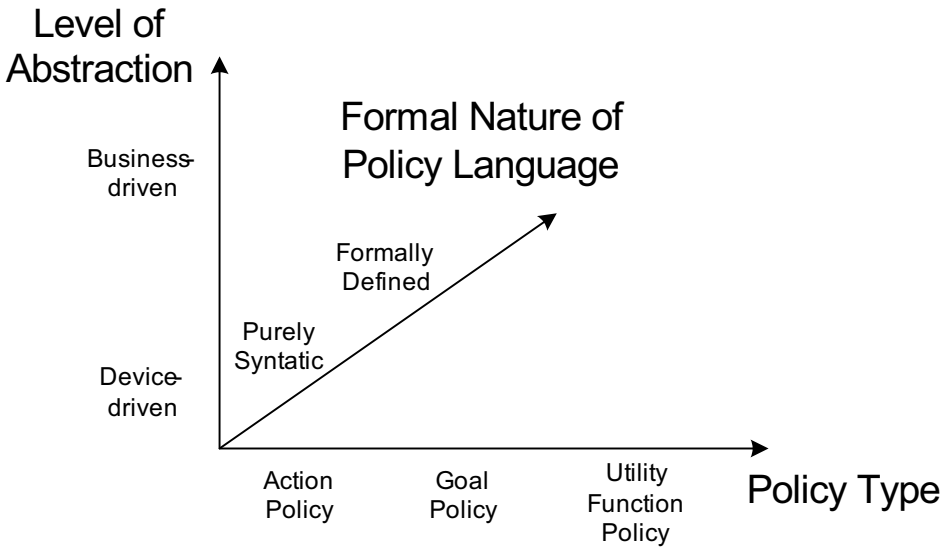

Figure 2.3: Policy classification scheme.

fication scheme that distinguishes policies on a conceptual level according to three orthogonal dimensions: horizontally we distinguish between the *type of the policy* that determines the information that is expressible; vertically we differentiate the *levels of abstraction* on which a policy is defined; finally, each policy (type) can be expressed with various languages ranging from pure syntactic specification to languages with a well-defined formal semantics. Figure 2.2.3 illustrates these different dimensions. The classification provides the basis for the design of an appropriate policy language later in this work.

Policy Type

The notion of *policy type* originates from the field of agent design. Russel and Norvig [RN03, Chapter 2.4] distinguish between *simple reflex agents*, *goal-based agents* and *utility-based agents*. To illustrate the three different approaches we use a transition system as a common framework for comparing the policies embodied in these kinds of agents. An labeled transition system is based on the notion of *states* and *actions* as specified in the following definition.

Definition 2.6 (Labeled Transition System) *A labeled transition system is a tuple* (S, A, S) *where* S *is a set of states,* A *is a set of actions and* $\rightarrow \subseteq S \times A \times S$ *is a ternary relation between states and actions called transition. If* $s, s' \in S$ *and* $a \in A$*, then* $(s, a, s') \in \rightarrow$ *represents a transition from state* s *to* s' *triggered by action* a *and is written* $s \xrightarrow{a} s'$.

A state $s \in S$ characterizes a system or system component and is usually described by a set of attributes (directly or indirectly) measured by a sensor. In a current state s a certain set of actions A can be taken which results in transitions to new states $S' \subseteq S$. For simplicity we consider only one-shot decisions and assume a deterministic environment without uncertain transitions, i.e. for a given s and a possible a there is exactly one s' such that $(s, a, s') \in \rightarrow$.

Action Policies. Simple reflex agents select actions based on the current perception of the sensors. The policies that encode the agents behavior have the form of

condition-action rules, which can be written as

$$\textbf{if} \quad \underbrace{\textit{service supports encryption}}_{\textit{condition}} \quad \textbf{then} \quad \underbrace{\textit{invoke service}}_{\textit{action}}.$$

In line with [KW04] we call these policies *action policies*. By comparing the current state S of a system to the condition (denoted by Φ) specified in the policy, possible actions $A' \subseteq A$ are determined. Thus, an action policy can be seen as a function $F_\Phi : A \times S \rightarrow \{0,1\}$ and $A' = \{a \in A | \exists s \in S : F_\Phi(a,s) = 1\}$. In order to exhibit rational behavior the action policy set must cover the entire state space and only one action should be triggered in one state. Since this can be hardly guaranteed in complex scenarios, explicit mechanisms for conflict handling between policies are required. For example, this can be realized by prioritizing policies or by explicitly introducing "meta-policies" that define which policy should be used in case of a conflict.

Goal Policies. However, action policies are often not sufficient to make a decision since they only regard the current state s when selecting an appropriate action a and do not consider information about the desired state s'. *Goal policies*, in contrast, avoid specifying what to do in a current state s, but rather specify the set of desired states S', which is called a *goal*. Goal policies define the desired state by declaratively specifying constraints Φ on its characteristics and can thus be seen as a function $G_\Phi : S \rightarrow \{0,1\}$ mapping each state to a value of 0 or 1, where 1 characterizes a desired state and 0 a not desired one (i.e. $S' = \{s \in S | G_\Phi(s) = 1\}$). With this approach rational behavior has not to be specified explicitly, but is generated by the system itself. This provides greater flexibility and frees human administrators from knowing detailed system information [KW04]. Since reaching a desired state requires knowledge about the actions to be executed to reach this state, sophisticated planning or modeling algorithms might be required. Due to the fact that actions can be derived automatically from goals, goal policies can be considered as higher level forms of policies [KW04].

Utility Function Policies. Goal policies as defined above are limited in a sense that any member of the set S' is equally desired and thus such policies cannot reflect preferences between states. That means a decision maker is *indifferent* between the different states that can be realized. Preferences can be expressed by generalizing goal policies in a sense that the function G is replaced by a function $U : S \rightarrow \mathbb{R}$ that maps each state to a real-valued number. In line with [KW04] we call declarative representations of such functions *utility function policies*. By explicitly specifying the trade-off between different states, they allow for unambiguous and rational decision making also in cases where goal policies would lead to a conflict. By means of an optimization algorithm the most desired state can be determined and goal as well as action policies can be derived from utility function policies.

Level of Abstraction

Each of the policy types introduced above can be expressed on different levels of abstraction forming a *policy continuum* [Str02]. Generally, at least two main levels can be distinguished [AAFP03]: (*i*) *Low level policies* that are defined directly based on

detailed system information. We call them *system* or *device-driven*. Since profound knowledge about the system is required such policies are typically defined by technical experts. For example, such policies might define that services supporting the AES encryption protocol with a key of 1024 bit are preferred, or that system logs may deleted after two weeks with acknowledgement of the administrator. (*ii*) *High level policies*, in contrast, are formulated from a business perspective and regulate more general aspects such as service levels an application has to meet or IT-governance regulations. These policies are relatively independent of the underlying technology. Hence, we say they are *business-driven*.

Instead of distinguishing between business- and device-driven policy definitions one can also divide the policy continuum according to roles of people defining the policies, which leads to a more fine-grain segmentation. Strassner [Str02] suggests introducing different types of *views* optimized for a certain user group. The *Business View* allows defining high-level policies using business terms and avoids technical details. The *System View* translates business policies to the technical terminology but generalizes from a specific technology. For example, a business policy specifying that only premium customers are allowed to use a certain service is translated to system policy specifying that users taking the role of premium customers can obtain a special type of access rights; others cannot. In a next step, the policies defined in the *Administrator View* map them to specific technologies, e.g. to the specific user model or system architecture. Depending on the concrete application and system implementation further views can be defined. What views are required depends on the groups of people defining policies for the system.

Formal Nature of Policy Language

The third dimension captures the language aspect. In recent years, a vast amount of policy languages have been developed for various purposes including security as well as trust aspects and business rules. Each approach comes with a policy specification language that enables expressing, storing and interpreting polices. Policy languages range from natural language descriptions (e.g. [MBG99, MOR01]) via more structured languages with a standardized syntax (e.g. [W3C06b, MAPG03, IBM03]) to formal languages based on an underlying logical calculus (e.g. [BSD+04, KPKH05, Kag04, TBJ+03]). Natural language policies are deemed to be the most intuitive approach for human policy authors. However, automatic interpretation and enforcement of policies is not completely achievable due to highly ambiguous nature of natural languages statements. Controlled vocabularies and structured policies expressions can improve the situation and enable automated processing of polices through a special interpreter. Since this interpreter implicitly defines the semantics of the language syntax, it is difficult to determine their expressivity and computational properties. Moreover, a well-defined semantics which can be realized by mapping the language constructs into a logic (e.g., some variant of first order logic) provides improved interoperability. This is particularly true in scenarios where policies have to be exchanged between different independent companies as it is typically the case in service-oriented architectures.

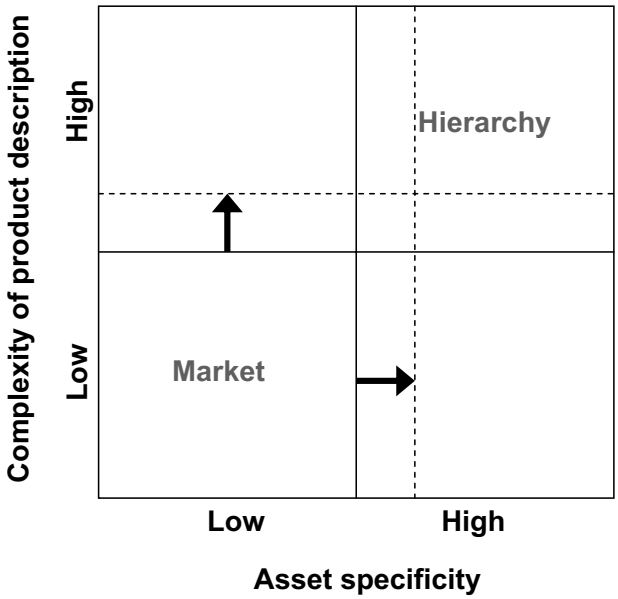

Figure 2.4: Influence of information technology on the applicability of markets [MYB87].

2.3 Electronic Markets

The infrastructure coordinating service supply and demand can be seen as a market platform. Economic theory distinguishes between two extreme forms of coordination that enable transactions between different parities: (*i*) *markets* and (*ii*) *hierarchies*. (*i*) Markets coordinate the transactions through supply and demand forces, which determine prices, quantities, quality, etc. Traditionally, markets have been used in scenarios with many autonomous participants, where products are simple and mostly standardized, and where the required interactions are rather simple. (*ii*) In hierarchies transactions are planned by controlling and directing at a higher position in the hierarchy. In scenarios with more complex products and interactions usually a hierarchical coordination mechanism has been used. While markets are preferable in terms of transaction costs (e.g. they provide more efficient information processing), they typically come with much higher coordination costs than hierarchical approaches (e.g. selecting suppliers, negotiating contracts, paying bills, etc.) [MS84].

However, with the availability of markets based on more sophisticated information and communication infrastructure a shift towards market-based coordination can be realized [MYB87]. Such *electronic markets* are institutions that allow the exchange of goods and services between multiple participants through global communication networks, such as the Internet. In the process, they create economic value for buyers, sellers, market intermediaries, and for the society at large [Bak98].

Electronic markets differ considerably from classical markets by being independent from time and space [Sch93]. For example, they enable world-wide access and trading all day and night which is not possible in most "off-line" markets. Therefore, more information can be gathered in a shorter time and due to the electronic nature additional market services can be provided, which reduce the transaction costs in the market. For example, they may reduce search costs for products and

information, enable companies to automate their transactions with business partners all over the world, and facilitate product customization and aggregation. In fact, improved information representation and handling within electronic markets leads a much broader applicability of markets beyond simple uniform goods and commodities. Figure 2.4 captures this idea by illustrating applicability of the coordination mechanisms depending on:

- *product complexity*, i.e. the amount of information required for describing a product in such detail that a meaningful matching and selection can be carried out.
- *asset specificity*, which refers to the fact that certain products cannot be used by other person or companies, because they are not easily transferable; for example, a huge machine or internalized knowledge.

For example, even complex products can be traded via an auction if market infrastructure provides an adequate representation formalism (i.e. bidding language) and matching algorithms. In this context, the concept of ontologies introduced in Section 2.4 plays a crucial role.

Before we come to the representational aspects, we look in more detail on the market process. Section 2.3.1 introduces the different phases the market process can be partitioned and Section 2.3.2 provides more insight into the contracting process by discussing different market mechanisms.

2.3.1 Market Phases

The exchange of products and services between customers and suppliers is carried out through *business transactions*, which can be seen as the process of initiating, arranging and completing a contractual agreement about the exchange of goods and services [LS98]. Langenohl [Lan94, pp. 18-22] identifies three main transaction phases of electronic markets – *information*, *agreement* and *settlement phase* – which are discussed in the following.

Information Phase: In the information phase market participants gather all kinds of information about the participants in the market and the products available. This could for example comprise information about the reputation or credit rating of potential business partners or the concrete technical specifications of a product. Based on this information an offer (either to sell or buy) is generated. With the submission of the offer to the market the information phase for a certain market participant ends.

Agreement Phase: Starting with receiving the offers and requests from the market participant, the agreement phase constitutes the core component of a market infrastructure. Ströbele and Weinhardt [SW03] distinguish between three steps that have to be executed to transform requests and offers to legally binding contracts:

- *Matching*: Matching (Matchmaking) is the process of comparing requests and offers with the goal of finding suitable counterparts. Matching is thus a core functionality of a market mechanism. The quality of a market

crucially depends on the quality of the matching algorithms used. In fact, the quality will be low if many ill-suited matches are realized as well as if many suitable matches are not realized. This corresponds to the concepts of *precision* and *recall* known from information retrieval [vR79].

- *Allocation*: An allocation is a function that maps the set of requests to a set of offers. Note that this mapping does not have to be bijective in sense that for all offers and requests a suitable counterpart is assigned. For example, several requests might be assigned to an extremely competitive service offer, or in case of excess demand some requests may not be assigned to any offer. Determination of an allocation can be done by means of the take-it-or-leave-it principle or it might involve negotiations or auctions mechanisms to increase the efficiency of the market.
- *Acceptance:* After the allocation is determined, for each pair of customers and providers allocated to each other a legally binding contract has to be concluded. With closing a contract an agreement between a customer and provider is reached and thus the agreement phase is completed.

Settlement Phase: Finally, in the settlement phase the transaction agreed upon is carried out, which might involve the exchange of products or the invocation of a service. With a proper execution a contract is fulfilled. The contractors might further want to verify if a certain execution complies with the terms agreed-upon. This involves *monitoring* of the execution.

In the following, we will have a closer look on the agreement phase, in which a *market mechanisms* provides matching and allocation functionality.

2.3.2 Market Mechanisms

Market mechanisms can be seen as an institution according to the Neo-classical institution theory that define the set of admissible actions (e.g. available messages in the communication protocol), and the rules that define how the outcome is determined based on these actions. According to [Par01, MMW06], an outcome $\mathcal{O}$ refers to an *allocation* of products to market participants $i \in \{1,\ldots,N\}$. A market mechanism thus consists of two set of rules: those defining the set of admissible actions (called *strategies*) which are denoted by $\Sigma_1 \ldots \Sigma_N$ and those for selecting the allocation based on the actions which is represented by a function $g : \Sigma_1 \times \cdots \times \Sigma_N \rightarrow X^N$.

In the following, we distinguish between between two basic forms of market mechanisms: rather simple mechanisms based on *fixed prices* and more complex mechanisms featuring *dynamic pricing* used in negotiations and auctions.

Fixed-price Mechanisms

In a fixed-price mechanism prices are statically defined by the market participants. In particular, the price does not react to the bids and therefore does not adapt to new information coming in the market. Such an approach is usually adopted in traditional retail markets, where the supplier dictates the price leaving no room for negotiations. A popular fixed-price mechanism is the *hit-and-take mechanism* described below.

Definition 2.7 (Hit-and-Take Mechanism) *A hit-and-take mechanism (aka take-it-or-leave-it mechanism or offer/accept mechanism) requires the provider (requester) to announce its transaction proposal including detailed product description and fixed price. This price represents the acceptable price for which the product can be sold/bought. Given this price (together with the exact product description and trading conditions) the potential trading partner either accepts this transaction proposal or declines it. Conflicts arising due to exceeding demand and supply are handled according to the first come first serve principle.*

For example, if a price fixed by the provider is lower than the corresponding reservation price of the requester, a potential transaction is found by the mechanism; if the price fixed by provider is higher then the requester's price, no transaction can take place. In this context, the problem of a fix-price mechanism becomes evident. In scenarios where prices of products are not known exactly (e.g. a product is unique or extremely volatile) it is hard to fix a price in a way that the market performs optimally, e.g. a maximum of transactions are carried out.

Dynamic-pricing Mechanisms

In order to address the problem of inefficient allocations, dynamic pricing mechanisms can be used. Dynamic pricing refers to a mechanism where prices and other transaction conditions are dynamically fixed based on the interplay between supply and demand. According to [Hur73], such a coordination mechanism can be used to allocate resources efficiently to requesters. They allow the determination of prices in cases where the true value is not known and the market participants' estimate may be imperfect. There are two main forms of dynamic-pricing mechanisms which are discussed in the following.

As a first category of mechanisms featuring dynamic pricing we consider *negotiations*. As defined by [LWJ01, BKS03], in the following we use a rather general definition that covers mechanisms ranging from highly individual bilateral negotiations to mechanisms with very structured protocols.

Definition 2.8 (Negotiation) *A negotiation is an iterative, progressive communication and decision making process by which a group of agents communicate with one another to try to reach an agreement on some matter of common interest. Usually the process starts with a rather inefficient offer and leads to a compromise (or to a disagreement). A negotiation between exactly one buyer and one seller is called a bilateral negotiation.*

As a second category of dynamic-pricing mechanisms we introduce *auctions*.[6] The most appealing properties of auctions are their process efficiency (e.g. simple communication protocol, high rates of Pareto-efficient outcomes, fast convergence to equilibrium) and ability to manage a large number of bidders. Therefore, auctions have emerged as the primary market institution for electronic commerce.

Definition 2.9 (Auction) *An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids submitted by the market participants [MM87]. Thus, auctioning is the form of negotiation with simple well-defined rules, but it naturally includes multiple parties [Kar03].*

[6] Note that auctions (and particularly on-line auctions) can be seen as a special kind of negotiation mechanism having a distributive negotiation protocol and multiple parties. For a detail discussion on the relation of negotiation and auction mechanisms the interested reader is referred to [KNT00, BKS03].

Naturally, the design of negotiation systems requires a highly interdisciplinary approach drawing from computer science and information systems, economic sciences and management, and law and social sciences [BKS03, WHN03]. In this context, law and social sciences mainly contribute with qualitative studies of the market participants' behavior and prescriptive as well as descriptive negotiation models. Computer science deals with designing electronic market platforms, decision support systems, and agent-based simulation framework for markets. Economics, finally, contribute to the field by providing techniques for constructing agent strategies and formal negotiation models that can be used for predicting market outcomes. This usually involves a game-theoretic analysis from which certain conclusions for the institutional design of market mechanisms can be drawn, i.e. how should a social choice function look like that implements $g(\cdot)$ of the mechanism? For example, in one of the first game-theoretic approaches to negotiations Nash [Nas50, Nas53] describes a two-person multi-item negotiation problem and presents a technique for determining equilibria that represent optimal strategies for the market participants. Typically, the goal is to design market mechanisms with the following characteristics (compare e.g. [Par01, DJP03, BKS03, SNVW06]):

- *Pareto-optimal outcomes*, i.e. there is no outcome, where one agent is better-off without other agents being worse-off.
- The pricing mechanism should be *incentive compatible*, i.e. each self-interested agent has an incentive to bid its true valuation of the product.
- *Allocative efficiency*, i.e. the total utility across all market participants should be maximal.
- The outcome should be *budget balanced*, which means that the sum of all payments in the market is zero. No money is removed from or injected into the system.
- *Individual rationality*, which means all participants realize a nonnegative utility in equilibrium.

A wide range of different electronic negotiation and auction mechanisms has been proposed in literature and some of them have already been successfully implemented in practice (e.g. eBay[7], onSale[8]). For a more detailed overview of negotiation and auction mechanisms refer to overview articles such as [OR05, LLSG04] for (bilateral) negotiations and [MW82, dVV03, ADR05] for auctions, respectively.

In addition, several classification schemas for dynamic-pricing mechanisms have been proposed. [SW03] provides a comprehensive classification of negotiation and auction mechanisms according to endogenous and exogenous factors. Other classifications have been presented, e.g., by [LWJ01] focusing on automated negotiations between agents and [WWW01] focusing purely on auctions.

For our work, a coarse classification along the main dimensions of a market mechanism is sufficient. We adopt the view of [BKK+02], where a market mechanism is described by three dimensions:

[7] www.ebay.com
[8] http://www.onsale.com/

- *Multi-attribute*: In order to allow negotiations not only about price, there are mechanisms that support multiple attributes, which capture additional characteristics of the product such as quality aspects.
- *Multi-unit*: Often a buyer requires several units of a product at once. In this case, often volume discounts are provided or one time costs such as registration fees have to be paid. Obviously, these aspects have to be explicitly considered during negotiation.
- *Multi-item*: In some cases not only one product is required but a bundle of products. In such cases, the value of a bundle containing both products might be valued higher by a customer than the sum of the value for the single products. We call this *superadditivity*. Superadditive prices occur in case of complementary products that are usually used together, such as desktop computers and computer monitors. Similarly, *subadditivity* describes substitutes where products suit the same purpose, e.g. a laptop and a desktop computer. Mechanisms supporting multiple items are also called *combinatorial* market mechanisms.

In the context of Web service markets, we will see later (Chapter 4) that such multi-dimensional markets are required. As already discussed above, in order to realize market mechanisms in a distributed environment with complex products like Web services an expressive knowledge representation formalism with the corresponding matching algorithms is required. Therefore, in the next chapter we introduce the concept of ontologies which provide expressive means for representing market information and an executable calculus for handling this information in an efficient way.

2.4 Semantic Technologies

In this section, we present basic technologies for the formalization of knowledge and its processing within machines. Knowledge representation and reasoning is a branch of symbolic Artificial Intelligence that aims at designing computer systems that enable reasoning about a machine-interpretable representation of domain knowledge. In this section, we show how *ontologies* as conceptual models enable formalizing the semantics of information in heterogeneous, distributed systems, such as service-oriented architecture or Web-based markets. Thereby, an ontology formally specifies the relationship between the data and its meaning, and thus provides an unambiguous language that can be interpreted by humans and machines alike.

After defining the concept of ontologies in Section 2.4.1, we discuss languages for the specification of ontologies in Section 2.4.2 focusing on the Web Ontology Language (OWL), the Semantic Web Rule Language (SWRL) and the query language SPARQL. We introduce a classification of ontologies according to their generality in Section 2.4.3 and then present the foundational ontology DOLCE as a basis for the ontologies developed throughout this work in Section 2.4.4.

2.4.1 Ontologies

While originally the term *ontology* denotes a branch of metaphysics introduced by Aristotle [Ari08] that addresses the philosophical investigation of existence, *ontologies* in computer science are computational artifacts that represents knowledge about a domain of interest. In recent years, ontologies became an important technology for knowledge sharing in distributed, heterogeneous environments, particularly in the context of the *Semantic Web* [BLHL01]. Within the Semantic Web community the following definition is predominantly used [SBF98].

Definition 2.10 (Ontology) *An ontology is a formal explicit specification of a shared conceptualization of a domain of interest.*

Requiring an ontology to be an "explicit specification of a conceptualization" was first introduced by Gruber [Gru93]. Conceptualization refers to the way knowledge is represented. It is encoded in an abstract manner using concepts and relations between concepts. Abstractness refers to the fact that ontologies try to cover as many situations as possible, instead of focusing on particular individuals [Gua98]. An "explicit specification" refers to the fact that the concepts and the constraints on their use are explicitly defined in an ontology and thus accessible for machines. This basic definition is extended by requiring a "formal specification" and a "shared conceptualization" [Bor97]. In this context, formality refers to the type of knowledge representation language used for specifying the ontology. This language has to provide formal semantics in a sense that the domain knowledge can be interpreted by machines in an unambiguous and well-defined way. In addition, the vocabulary formally defined by this language should represent a consensus between the members of a community. By committing to such a common ontology, community members (or more precisely their software agents) can make assertions or ask queries that are understood by the other members. Finally, an ontology always covers knowledge about a certain "domain of interest". Therefore, many applications use a set of ontology modules that model different aspects of the application.

There is a broad range of application areas where ontologies have been successfully used within the last years. Examples are information integration (e.g. [ABdB+05]), matching of products or user profiles (e.g. [NSDM03, CCC+04]), and the search of textual or multimedia content (e.g. [SR03, PBS+06]). For the different applications different ontology languages with a different degree of formality are required. For example, in many applications already a rather low degree of formalization can be sufficient to realize immediate benefits [Hen03].

Common to most languages are the principal constituents: concepts, relations and instances. Depending on the concrete language, they are represented differently. For instance, they map to generic nodes in a semantic network, to unary predicates in logic or to concepts in description logic [GHA07]. Instead of introducing all of these different formalisms, we refer the reader to [SS04b] for an overview of ontology languages. In the following, we introduce only formalisms that are specifically required throughout this work.

2.4.2 Ontology Formalisms

In this section, we introduce the languages that are used for representing and querying knowledge within the service-oriented architecture. We rely on the ontology

language OWL which is standardized by the World Wide Web Consortium (W3C)[9]. We first introduce the main ideas behind OWL as well as its logical foundations. Then the rule language SWRL is presented that considerably increases expressiveness of OWL by supporting additional axioms. Finally, a query language for our OWL and SWRL knowledge bases is introduced that provides means for retrieving knowledge from the ontology.

Web Ontology Language (OWL)

In order to guarantee mutual understanding in distributed environments, the underlying logic has to be standardized. The Web Ontology Language (OWL) [W3C04c] is an expressive ontology language standardized by the World Wide Web Consortium. Historically, OWL emerged from several former knowledge representation and description languages, like SHOE [Hef01] and DAML+OIL [W3C01a]. The development was mainly driven by the need to build one widely accepted and backward compatible standard for knowledge sharing in the Web. The logical foundation of OWL is a subset of first-order logic called *description logic* [BCM$^+$03]. Subsequently, we first introduce the family of description logic languages and then show how they can be used in a Web environment.

Description Logics. The development of description logic was influenced by ideas stemming from the work on frame languages, such KL-ONE (see [BS85]), which provided a logical basis for interpreting individuals, concepts (unary predicates) and roles (binary predicates) between them. Concepts can be built by concept and role constructers. In addition, terminological axioms can be used to define how concepts or roles are related and assertional facts can be used to define statements about the properties of individuals.

A major focus of research has been the trade-off between expressiveness of the knowledge representation language and the difficulty of reasoning over this language [BL84]. In practical applications it is particularly important that algorithms exist, which allow for deriving logical consequences in a sound and complete manner. If one can guarantee that these algorithms always terminate, the corresponding logic is called *decidable*.

Decidability of a description logic depends on the provided constructors from which concepts and relations can be composed. Table 2.4.2 gives a short overview of the constructs available in a particular description logic. OWL comes in three levels of expressiveness: OWL-Lite, OWL-DL and OWL-Full, reflecting different degrees of expressiveness and in turn also different degrees of scalability (compare [HPSvH03]). OWL-Lite as well as OWL-DL directly map to a corresponding description logic dialect, whereas OWL-Full departs from the description logic semantics.

According to [HPSvH03], OWL-Lite is equivalent to the $\mathcal{SHIF}(\mathbf{D})$ description logic and it is in worst case decidable within deterministic exponential time (EXPTIME complexity). OWL-DL extends $\mathcal{SHIF}(\mathbf{D})$ with nominals and allows unqualified number restrictions. Hence it is equivalent to the $\mathcal{SHOIN}(\mathbf{D})$ description logic which is in worst case decidable in non-deterministic exponential time (NEXPTIME complexity) and for which no nearly optimal and complete inference

[9] http://w3c.org

Symbol	Available Constructs
$\mathcal{AL}$	conjunction, universal value restriction and limited existential quantification
$\mathcal{C}$	disjunct and full existential quantification with full negation
$\mathcal{R}+$	transitive role (owl:TransitiveProperty)
$\mathcal{S}$	shortcut for ALCR+
$\mathcal{H}$	role hierarchy (rdfs:subPropertyOf)
$\mathcal{I}$	inverse role (owl:inverseOf)
$\mathcal{F}$	functional role (owl:FunctionalProperty)
$\mathcal{O}$	nominals, i.e. enumeration of classes or data values (owl:oneOf and owl:hasValue)
$\mathcal{Q}$	qualified number restrictions
$\mathcal{N}$	unqualified number restrictions
$\mathbf{D}$	concrete domains

Table 2.1: Description logic variants [BCM+03].

algorithm exists. In general, OWL-Full is shown to be undecidable [Mot05]. In the following, we define the syntax and semantic of the most expressive decidable language OWL-DL, which fully subsumes the OWL-Lite fragment and is used throughout the work.

The syntax of $\mathcal{SHOIN}(\mathbf{D})$ is given by concept as well as role descriptions and constructers for transforming them into complex concept and role descriptions. In addition, individuals and datatypes are supported. The meaning of these modeling constructs is formally defined via a *model theoretic semantics*, i.e. it is defined by relating the language syntax to a model consisting of a non-empty set of objects $\Delta^{\mathcal{I}}$, denoted by a domain, and an interpretation function $\mathcal{I}$, which maps entities of the ontology (e.g. an atomic concept ϕ) to concrete entities in the domain (e.g. the set $\mathbf{C}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$) [HPSvH03]. Thereby, *axioms* define certain constraints on these interpretations. Based on [BCM+03], the following list defines certain concept descriptions and constructors with their abstract syntax and model theoretic semantic. Let $\mathbf{A}, \mathbf{R}$, and $\mathbf{I}$ be pairwise disjoint finite non-empty sets of atomic concepts, roles and individuals, respectively.

- *BOTTOM*: the concept $\bot$ represents a shortcut for $\phi \sqcap \neg\phi$ with $\phi \in \mathbf{A}$ and $\bot^{\mathcal{I}} = \emptyset$.
- *TOP*: the concept $\top$ represents a shortcut for $\phi \sqcup \neg\phi$ with $\phi \in \mathbf{A}$ and $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$.
- *Conjunction*: the conjunction of concepts $\phi \sqcap \psi$ (with $\phi, \psi \in \mathbf{A}$) refers to the set of individuals belonging to both concepts, i.e. $(\phi \sqcap \psi)^{\mathcal{I}} = \phi^{\mathcal{I}} \cap \psi^{\mathcal{I}}$.
- *Disjunction*: the disjunction of concepts $\phi \sqcup \psi$ (with $\phi, \psi \in \mathbf{A}$) refers to the set of individuals belonging to either ϕ or ψ, i.e. $(\phi \sqcup \psi)^{\mathcal{I}} = \phi^{\mathcal{I}} \cup \psi^{\mathcal{I}}$.
- *Negation*: the complement $\neg\phi$ with $\phi \in \mathbf{A}$ contains all individuals not contained in ϕ, i.e. $\neg\phi^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash \phi^{\mathcal{I}}$
- *Existential restriction*: an existential restriction $\exists R.\phi$ denotes that only individuals with a relation R belong to the concept ϕ, where $R \in \mathbf{R}$ and $\phi \in \mathbf{A}$, i.e. $(\exists R.\phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in \phi^{\mathcal{I}}\}$.

- *Universal restriction*: an universal restriction $\forall R.\phi$ denotes individuals, for which all roles R point to the concept ϕ, where $R \in \mathbf{R}$ and $\phi \in \mathbf{A}$. The interpretation is given by $(\forall R.\phi)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in \phi^{\mathcal{I}}\}$.

- *Unqualified number restrictions*: an unqualified number restriction $\geq_n R$, $\leq_n R$, or $= nR$ defines concepts of individuals having at least, at most, or exactly n relations $R \in \mathbf{R}$, respectively. The semantics for $\geq_n R$ is given by $(\geq_n R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} || \{b \in \Delta^{\mathcal{I}}\}|(a,b) \in R^{\mathcal{I}}\}| \geq_n\}$. The definitions for $\leq_n R$ and $=_n R$ are analogous.

- *Nominals*: nominals I are individuals used in concept expressions and they are interpreted as singleton sets that consist exactly of one element of the domain, i.e. $I^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and $|I^{\mathcal{I}}| = 1$.

In the following, we introduce an additional concept constructor originally not contained in OWL-DL, but planned for the next version of OWL (Version 1.1). Namely, this constructor is a qualified number restriction, which extents unqualified number restrictions in that a range concept can be defined. Qualified number restriction are already supported by most OWL-DL reasoners. The semantics of qualified number restrictions is given as follows:

- *Qualified number restrictions*: an qualified number restriction $\geq nR.\phi$, $\leq nR.\phi$, or $= nR.\phi$ defines concepts of individuals having at least, at most, or exactly n relations $R \in \mathbf{R}$ to a concept ϕ. The semantics for $\geq nR$ is given by $(\geq nR)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} || \{b \in \Delta^{\mathcal{I}}\}|(a,b) \in R^{\mathcal{I}} \wedge b \in \phi^{\mathcal{I}}\}| \geq n\}$. The definitions for $\leq nR.\phi$ and $= nR.\phi$ are analogous.

The description logic concept constructers are augmented by role constructers that combine role and/or concept descriptions to more complex role descriptions. $\mathcal{SHOIN}(\mathbf{D})$ supports transitive closure as well as inverse roles:

- *Transitive closure*: The transitive closure R^+ with $R \in \mathbf{R}$ allows modeling the transitive characteristic of roles, i.e. $(R^+)^{\mathcal{I}}$ is the transitive closure of $R^{\mathcal{I}}$.

- *Inverse roles*: An inverse role R^- with $R \in \mathbf{R}$ has inverted domain and range descriptions, i.e. $(R^-)^{\mathcal{I}} = \{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\}$.

Based on the concept and role definitions introduced above, terminological and assertional axioms can be defined, which constrain the allowed interpretations. These axioms cover concept (role) inclusion, equality and assertion are defined as follows.

- *Inclusion*: If a concept $\phi \in \mathbf{A}$ is a subconcept of another concept $\psi \in \mathbf{A}$, we write $\phi \sqsubseteq \psi$. In this case $\phi^{\mathcal{I}} \subseteq \psi^{\mathcal{I}}$ holds. The same is true for roles and is defined analogously for the inverse inclusion "$\sqsupseteq$". A set of concept inclusions is called *concept hierarchy*, a set of role inclusions *role hierarchy*.

- *Equality*: Concept equality $\phi \equiv \psi$ is given if the two concepts classify the same individuals $\phi^{\mathcal{I}} = \psi^{\mathcal{I}}$ (role equality is defined analogously).

- *Assertion*: A concept assertion defines that an individual a belongs to a concept ϕ. Concept assertions are denoted by $\phi(a)$ and the semantics is defined by $a^{\mathcal{I}} \in \phi^{\mathcal{I}}$. Similarly, role assertions $R(a,b)$ are defined as $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

By means of the underlying model theoretic semantics that describes models that are valid according to a certain description logic theory, *logical consequences* can be drawn from the set of axioms defined in the ontology – a process usually referred to as *deduction* or *inferencing*. Thereby, knowledge can be derived that is not explicitly stated but that is implicitly given by the logical theory [GHA07]. When an axiom F *satisfies* an interpretation $\mathcal{I}$ (or if $\mathcal{I}$ is a model of F), we write $\mathcal{I} \models F$. In case $\mathcal{I} \models F$ holds for all axioms F in the knowledge base KB, we say the KB is *satisfiable*. A logical consequence is denoted by $KB \models F$. To illustrate this consider the following example.

Example 2.1 *Assume a service registry where a description logic knowledge base is used to describe, classify and store Web service offers. For example, consider the description of a route planning Web service. We define that (A1) a* Service[10] *has at least one input (indicated by relation* hasInput*) and output (indicated by relation* hasOutput*), (A3) a* RoutePlanningService *is a* Service *has exactly one role* hasStart *as well as* hasDestination *and at least one role* hasRoute*, which (A4/A5) are specializations of* hasInput *and* hasOutput*, respectively. In addition, (A6) assume a concrete* Service *that (A7) provide a route A8) between two places within Germany.*

(A1) $Service \sqsubseteq \exists hasInput.\top \sqcap \exists hasOutput.\top$

(A2) $RoutePlanningService \sqsubseteq Service \sqcap =_1 hasStart \sqcap =_1 hasDestination \sqcap$

(A3) $\exists hasRoute.\top$

(A4) $hasStart \sqsubseteq hasInput;\ hasDestination \sqsubseteq hasInput$

(A5) $hasRoute \sqsubseteq hasOutput$

(A6) $Service(ServiceCompA)$

(A7) $hasRoute(ServiceCompA, calculatedRoute)$

(A8) $hasStart(ServiceCompA, Germany)$,
$hasDestination(ServiceCompA, Germany)$

From a set of such axioms conclusions can be derived that are not explicitly stated in the ontology, e.g. a subsumption hierarchy between concepts in the ontology can be constructed. For example, consider the case where a requester is looking for a RoutePlanningService *in the knowledge base described above* (A1)-(A8)*. In this case the instance* ServiceCompA *is also returned as a result. Although it is not explicitly state that* ServiceCompA *is an instance of* RoutePlanningService*, we can infer this from the knowledge base, viz.,* $KB \models RoutePlanningService(ServiceCompA)$.

This is particularly important for matchmaking in heterogeneous markets, where offers and requests are usually described on different levels of abstraction, e.g. when looking for a route planning service for Germany also route planning service for entire Europe are relevant.

An Ontology Language for the Web. In order to represent ontologies in a compact and convenient way, we have described ontologies up to now using the abstract description logic syntax presented in [BCM+03]. However, in order to make

[10] Throughout the work entities from an ontology are formatted using the *slanted* style. Sometimes concept names in the text are used in plural to improve the readability.

```
 1 <rdf:RDF xmlns="http://www.ontoware.org/service#"
 2       xml:base="http://www.ontoware.org/service"
 3       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 4       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 5       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 6       xmlns:owl="http://www.w3.org/2002/07/owl#">
 7     <owl:Ontology rdf:about=""/>
 8     <owl:Class rdf:ID="RoutePlanningService">
 9         <rdfs:subClassOf rdf:resource="#Service"/>
10         <rdfs:subClassOf>
11             <owl:Restriction>
12                 <owl:onProperty rdf:resource="#supports"/>
13                 <owl:someValuesFrom rdf:resource="#Navigation"/>
14             </owl:Restriction>
15         </rdfs:subClassOf>
16         <rdfs:subClassOf>
17             <owl:Restriction>
18                 <owl:onProperty rdf:resource="#start"/>
19                 <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
20             </owl:Restriction>
21         </rdfs:subClassOf>
22         <rdfs:subClassOf>
23             <owl:Restriction>
24                 <owl:onProperty rdf:resource="#destination"/>
25                 <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
26             </owl:Restriction>
27         </rdfs:subClassOf>
28          ...
29     </owl:Class>
30     <owl:Class rdf:ID="Service"/>
31  ...
32 </rdf:RDF>
```

Listing 2.2: OWL ontology in XML/RDF serialization.

the description logic applicable for a heterogeneous, distributed environment such as the Web, resources (e.g. ontology elements) have to be uniquely identifiable in the system, support for extensive modularization has to be provided and compliance with existing Web languages and protocols has to be guaranteed. Thus, OWL uses Uniform Resource Identifiers (URI) for defining the vocabulary and provides a serialization using XML [W3C04a], which comes with support for datatypes and data values [W3C04e]. In addition, OWL provides an import mechanism for reusing other ontologies and therefore enables modularization, where different ontologies can be created and updated by different parties and still be shared within a distributed environment. Listing 2.2 shows an excerpt from the XML/RDF serialization of Example 2.1.

Built-in Name	Functionality
swrlb:equal(x,y)	true iff $x = y$
swrlb:notEqual(x,y)	true iff $x \neq y$
swrlb:lessThanOrEqual(x,y)	true iff $x \leq y$
swrlb:greaterThanOrEqual(x,y)	true iff $x \geq y$
swrlb:add(z,x,y)	true iff $z = x + y$
swrlb:subtract(z,x,y)	true iff $z = x - y$
swrlb:multiply(z,x,y)	true iff $z = x * y$
swrlb:divide(z,x,y)	true iff $z = x/y$
swrlb:max(z,x,y)	true iff $z = \max(x,y)$

Table 2.2: Subset of SWRL built-ins. A full list is available in [HPSB$^+$04].

Semantic Web Rule Language (SWRL)

In order to define our ontology, we require additional modeling primitives not provided by OWL-DL. For example, due to the restriction to tree structures [GHVD03] OWL-DL does not support triangle relations between concepts, such as the role *suitableFor* saying that a *Service* is suitable for a certain *Country* if the starting point of a route is in this *Country*. Obviously, this rule leads to non-tree models and the description logic becomes undecidable. In contrast to description logics, rule languages can be used to express such triangle relation. Intuitively, the head (consequent) of rule holds if the condition specified in the body (antecedent) holds. Thus, the example above can be formalized as follows:[11]

(R1) $\quad suitableFor(x,y) \leftarrow Country(y), start(x,z), locatedIn(z,y)$

However, compared to description logics, rule-based approaches have also drawbacks, e.g. they are restricted to universal quantification. The Semantic Web Rule Language (SWRL) [HPS04, HPSB$^+$04] allows us to extend OWL with Horn-like rules that are interpreted according to first-order semantics. In addition, SWRL provides a XML-based syntax for encoding rules within an ontology, an extension to the OWL semantics which provides formal meaning for SWRL constructs, and an extensible set of built-in predicates[12] that can be used for implementing operations such as arithmetic calculation, string comparisons or manipulations, etc. The SWRL built-ins used in this work are introduced in Table 2.2.

Unfortunately, reasoning with knowledge bases that contain arbitrary SWRL expression usually becomes undecidable [HPS04]. Thus, we restrict ourself to *DL-safe* rules [MSS05]. DL-safe rules keep the reasoning decidable by placing constraints on the format of the rule, namely each variable occurring in the rule must also occur in a non-DL-atom in the body of the rule. This means that the identity of all objects referred to in the rule has to be known explicitly (i.e. they have to be explicitly named in the knowledge base). For example, Rule R1 is not DL-safe, since x, y, and z occur

[11] For the notation of rules we rely on the standard first-order implication syntax. In the following, rules are labeled by $R1,\ldots,Rn$.

[12] Whenever built-ins are used within a rule, they are identified by the prefix "swrlb".

only in DL-atoms. However, the rule can be made DL-safe by adding the non-DL-atoms $\mathcal{O}(x)$,$\mathcal{O}(y)$, and $\mathcal{O}(z)$ to the body, which ensure that the variables refer only to *known* objects, i.e. individuals in the knowledge base. Since we deal only with known instances in our application and the terms $\mathcal{O}(x)$ are automatically added by the reasoner, we do not explicitly mention the non-DL-atoms $\mathcal{O}(x)$ in the following.

SPARQL

In order to access information stored in a knowledge base, a query language is required. *Queries* can be seen as "intentional" denotations [LL87] of individuals in the knowledge base representing required characteristics without referring to an concrete individual. The emerging standard for querying RDF and OWL ontologies is the query language SPARQL [W3C06a] currently being standardized by the W3C. In addition to a SQL-like query syntax, SPARQL provides a data access protocol based upon HTTP as well as SOAP, and a XML format in which query result are returned. Although originally a query language for RDF graphs or triples, SPARQL can be also used to express *conjunctive queries* over description logic knowledge bases. A detailed discussion how the SPARQL syntax can be used to encode conjunctive queries is presented in [Haa06]. Listing 2.3 shows the language syntax of SPARQL.

```
1 PREFIX ns:<uri of namespace>
2 SELECT [DISTINCT] <projection>
3 FROM <uri of dataset>
4 WHERE{<graph pattern> [FILTER <expression>]}
5 [ORDER BY <attribute> [ASC|DESC]|LIMIT <n> | OFFSET <m>]
```

Listing 2.3: SPARQL syntax.

Keywords like `SELECT`, `FROM`, `WHERE`, `FILTER`, etc. are interpreted in line with their meaning in SQL. The meaning of `<projection>`, `<graph pattern>` and `<expressions>` is given as follows [W3C06a]:

- `<projection>`: A Projection $p(QS,VS)$ is the solution $\{(v,QS(v))|vinVS\}$ over a query solution (QS) and a set of wildcard variables (VS). In other words it is a selected subset of the variables defined in the `<graph pattern>` section.
- `<graph pattern>`: There are four OWL relevant types of graph patterns that can be used in a query:
  - basic: A subject-predicate-object pattern binding OWL Classes and (or) properties to variables. For example, `?x <http://example.de/a.owl#hasValue> ?z`. Here `?x` is an variable representing an OWL individual whereas `?z` is an variable representing either an OWL individual or an data literal depending on the range of property `<http://example.de/a.owl#hasValue>`.
  - group: One graph pattern containing a conjunction of other graph patterns that must all match.
  - optional: A graph pattern that may fail to match but the query is still executed against the data without failing entirely.

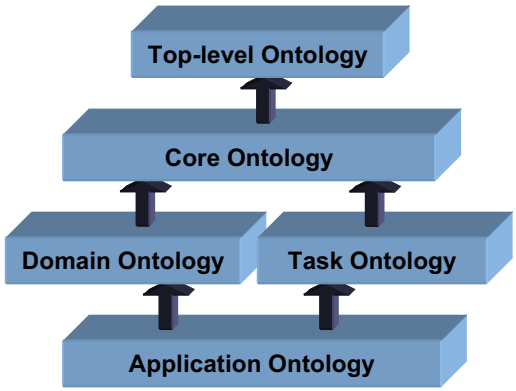

Figure 2.5: Categorization of Ontologies. Arrows represent specialization relationships.

  - union: A combination of two graph patterns that bind the same variables and may match.

- `<expression>`: A regular expression that is supplemented by a subset of the built-in functions and operators defined by XQuery [W3C07b]. To get an overview about the applicable unary, binary and trinary operators as well as the regular expression grammar the interested reader is referred to the [W3C06a].

After having defined the language for expressing knowledge bases including rules and queries for accessing the data stored in this knowledge bases, we discuss the different types of ontologies that can be built based on this languages in the next section.

2.4.3 Categorization of Ontologies

One of the central ideas behind ontologies is the possibility of reusing existing ontologies and thus reducing the modeling effort. However, it has turned out that different types of ontologies are more suitable for reuse than others. In this context, the generality of the ontologies is important: general ontologies can be reused in many different contexts, whereas very specific ontologies are rarely reused. Thus, the following categorization of ontologies can be applied [Gua97, Obe05]:[13]

Top-level Ontology: *Top-level ontologies* describe general concepts, such as object, event, action, etc. that may be present or occur in many (or even all) different domains and applications. Therefore, these concepts are independent from a concrete usage scenario and can be shared by a large community of users. Top-level ontologies are also often called *foundational*, *generic* or *upper level ontologies*. Since they are easily reused, it is worth devoting effort into building philosophically sound and highly axiomatized top-level ontologies, which unambiguously describe the vocabulary. Prominent examples of top-level ontologies are DOLCE [MBG+02b] and SUMO [NP01].

[13] Of course, also other categorization dimensions have been proposed in literature. For example, ontologies can be classified according to the level of formality or with respect to the ontology language used.

Core Ontology: *Core ontologies* are situated between the two extremes of top-level and domain/task/application ontologies. They are still application independent and generic over a set of domains, but serve a specific purpose required in different domains. For example, Oberle [Obe05] presents a Core Ontology of Software Components and Services which describe certain aspects of computer systems independent from a certain application. Core ontologies reuse the vocabulary defined in the top-level ontology.

Domain or Task Ontology: These ontologies describe vocabulary specific to a certain domain, such as financial instruments or location information, or specific to a certain task such as selling or diagnosing. Much work has already been devoted to domain ontologies in the area of medicine, genetics, geography, etc. and to task ontologies focusing on scheduling and planning tasks, intelligent tutoring, etc.

Application Ontology: An *application ontology*, finally, introduces vocabulary to adapt the ontologies above to a concrete application. Usually they can be rarely reused for other application contexts.

In this work, we mainly focus on developing appropriate core ontologies, e.g., for expressing service offers, requests and contracts in a Web service market. Since core ontologies are grounded in a top-level ontology, we introduce the foundational ontology DOLCE that provides the modeling basis for our work in the next section. The entire ontology framework for electronic markets is presented in chapter 6.

2.4.4 The Foundational Ontology DOLCE

While for some applications low quality ontologies with ambiguous vocabulary definitions might be sufficient (e.g. for ontology-based text classification [BCHS05] or information retrieval [Sto04]), exact definitions of the term are required for establishing consensus in a community (especially for persons joining the community). In order to explicitly capture the ontological commitment in a community, a rich axiomatization that eliminates terminological and conceptual ambiguities is required. However, building common ontologies in a bottom-up manner, where different ontologies of participants are integrated, might not be possible, since the intended models of the ontologies do not overlap [BGG+02]. By grounding the different domain ontologies on a common basis, a certain overlap can be ensured and reaching consensus becomes considerably easier. As introduced in Section 2.4.3, we call such a common basis foundational (or top-level) ontology.

Foundational ontologies are high-quality formalizations of domain independent concepts and associations that contain a rich axiomatization of their vocabulary. Such formal principles are required to allow a comparison and integration of different conceptualizations [GM03]. To enable high axiomatization, foundational ontologies are usually formalized with a rather expressive logic, e.g. full first order logic or modal logic. While this expressivity enables exact definitions of the vocabulary, it also leads to a high reasoning complexity and undecidability, which obstructs the direct practical applicability of foundational ontologies. Therefore, most foundational ontologies come also in a lightweight version that enables reasoning at runtime. Nevertheless, the heavyweight version is still useful, since it can be used

as reference, e.g., for determining a consensus and meaning negotiation. Another major advantage of foundational ontologies is the fact that by reusing generic concepts, relations and larger ontology structures the modeling effort can be reduced. We call such reusable structures *ontology design patterns* [Gan04]. Typical design patterns are location in space and time, which can be used in many different domains and applications.

In literature, several foundational ontologies have been presented over the last years, including the Basic Formal Ontology (BFO) [MBG+03], the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [MBG+02b], the Object-Centered High-level Reference Ontology (OCHRE) [Sch03], OpenCyc[14], and the Suggested Upper Merged Ontology (SUMO) [NP01]. Oberle [Obe05] compares these ontologies with respect to a set of requirements stemming from the design and management of middleware. These requirements are thus also applicable for this work. He concludes that only DOLCE supports all required features. In particular, only DOLCE provides a theory of contextualization and a theory of information objects which will be crucial for representing policies, bids and contracts in an electronic market. Moreover, DOLCE comes in a heavyweight as well as lightweight version and the modular structure of DOLCE reduces the risk of over-commitment, i.e. agents have to reach consensus only about certain domains (individual modules), not about their entire conceptualization. We, therefore, select DOLCE as modeling basis for our work. In the following, the part of DOLCE relevant for the subsequent chapters is introduces in more detail.

DOLCE

The foundational ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) provides a philosophically well-founded basis for developing core, domain, task and application ontologies. It has already been successfully applied in different domains, such as software engineering [Obe05], law [GST05], and biomedicine [GCB04]. Central to the structure of DOLCE is the distinction between *Endurants* (i.e. objects or substances), *Perdurants* (i.e. events or processes), *Qualities* and *Abstracts*. According to [MBG+02a], *Endurants exist in time* without having temporal parts, whereas *Perdurants happen in time* and may have temporal parts. That means, while *Endurants* may change in time, *Perdurants* cannot change, since they have no unique identity in time. The ability to model 4D entities, such as *Perdurants*, is a major advantage of DOLCE. *Qualities* inhere in *Perdurants* as well as in *Endurants* and represented entities that can be perceived or measured. *Qualities* are located in abstract entities called *Regions*. They encode *Qualities* in some metric or conceptual space, e.g. a color space or speed range. Each of the presented DOLCE entities features a hierarchy of specializations. A detailed description of DOLCE can be found in [MBG+02a].

Based on this backbone, further ontology modules are defined. These include ontological theories about contextualization, information objects and plans. They are provided by the DOLCE modules Descriptions & Situations (DnS), Ontology of Information Objects (OIO) and Ontology of Plans (OoP). The concepts of DOLCE and its modules required in our work are briefly described in Table 2.3.

[14] www.opencyc.org

Module	Concept label	Usage
DOLCE	*Endurant*	Static entities such as objects or substances
	Perdurant	Dynamic entities such as events or processes
	NonAgentive-SocialObject	Non-physical *Endurant* that does not actively participate in *Perdurants*
	NonAgentive-PhysicalObject	Physical *Endurant* that does not actively participate in *Perdurants*
	Quality	Basic entities that can be perceived or measured
	Region	Quality space such as colors, speed ranges, etc.
DnS	*SituationDescription*	Non-physical objects like plans, regulations, defining *Roles*, *Courses* and *Parameters*
	Role	Descriptive entities that are played by *Endurants* (e.g. a customer that is played by a certain person)
	Course	Descriptive entities that sequence *Perdurants* (e.g. a service invocation which sequences concrete communication activities)
	Parameter	Descriptive entities that are valued by *Regions* like the age of customer
	Situation	Concrete real world state of affaires using ground entities from DOLCE
OoP	*Task*	*Course* that sequences *Activities*
	Activity	*Perdurant* that represents a complex action
	Plan	describes a *SituationDescription* that sequences *Activities*
OIO	*InformationObject*	Entities of abstract information like the content of a book or a story

Table 2.3: Upper level concepts from DOLCE, Descriptions and Situations (DnS), Ontology of Plans (OoP) and Ontology of Information Objects (OIO) that are used as modeling basis.

Descriptions & Situations

The intent of Descriptions & Situations is the representation of non-physical objects, such as social institutions, regulations, plans or mental contents [GM03]. Therefore, Descriptions & Situations introduces the distinction between a *Situation* and a *SituationDescription*. A *Situation* is constituted by entities of the ground ontology (in our case DOLCE) and defines a state of affair (e.g. real settings in the world such as facts or legal cases). A *SituationDescription* (or in short *Description*) is a conceptualization, which encompasses non-physical social objects such as laws, plans, policies, etc. and it (partly) represents a theory that is perceived by an *Agent*. The fact that a *Situation* is a model of this theory is reflected by the *satisfies* relation between a *SituationDescription* and *Situation* which reifies the satisfiability relation, $\models$, of the underlying logic. A *SituationDescription* contains descriptive entities, such as *Roles*, *Course of Events*, and *Parameters*. By means of the *satisfies* relation they allow the definition of views on concrete *Situations*, i.e. depending on the con-

straints specified in the *SituationDescription* a concrete *Situations* satisfies or does not satisfy the *SituationDescription*.

The ground entities in Descriptions & Situations are derived from DOLCE: *Functional Roles* are *played-by DOLCE:Endurants*[15], *Courses of Events sequences DOLCE:Perdurants*, *Parameters* are *valued-by DOLCE:Regions*.

Ontology of Plans

The Ontology of Plans (OoP) uses the design pattern Descriptions & Situations and formalizes a theory of plans in a generic way, i.e. independent from concrete logical calculi. It specializes DOLCE and DnS by adding concepts for modeling planning concepts, such as *Tasks* and *Goals*, and by concretizing the *DnS:satisfies* relation.

A *Plan* represents a *DnS:SituationDescription* that *DnS:defines* a *DnS:FunctionalRole* and a *Task*. A *Task DnS:sequences Activities*, which specialize *DOLCE:Perdurants*. By refining the *DnS:satisfies* relation the Ontology of Plans can be used to decide whether a certain *Plan* will be (has been) fulfilled by a certain plan execution (e.g. workflow). For a detailed description of the Ontology of Plans the interested reader is referred to [GST04].

Ontology of Information Objects

The notion of information is a crucial concept in the area of computer science. However, it is often hard to grasp in a conceptual model. For example, one might distinguish between the content of information, the physical representation in a computer system, and encoding used for representation. Disregarding this distinction may easily lead to a conceptual ambiguity. This problem is, e.g., discussed in [MOGS04] for the service description ontology OWL-S. In order to avoid such problems, the Ontology of Information Objects (OIO) provides a design pattern for modeling abstract *Information Objects*, the encoding of these objects and the conceptualization expressed by an *Information Object*. A detailed description of the Ontology of Information Objects can be found in [GST04].

2.5 Conclusion

In this chapter, the basic technologies for realizing a semantic Web service market infrastructure have been presented. Therefore, we have first discussed the basic principles of service-oriented architectures and the technologies for implementing such architectures in a heterogeneous environment. In this context, mechanisms for dynamic binding of services at deployment or runtime are required. These mechanisms have to support the fact that Web services can be easily configured to customers' needs. To express and enforce such requirements autonomously at runtime, we have presented the idea of policy-based computing, where policies capture guidelines how the system should behave. Since the find-bind-execute-paradigm of a service-oriented architecture can be seen as a specialization of market process,

[15]For all concepts that are not contained in the ontology discussed in the current section, the namespace of the ontology module is added where the concept is derived from.

electronic markets have been introduced as a third technology. Finally, we have presented ontologies as a means for representing information that provide a high degree of interoperability and enable powerful matching algorithms. This is required to facilitate the use of market mechanisms for complex goods and services. In chapter 3, we show from a methodological as well as from a conceptual point of view, how these technologies can be seamlessly combined in order to realize a semantic Web service market infrastructure.

Chapter 3

Towards a Semantic Web Service Market

We approach the goal of this thesis systematically by first defining a methodology which is used for deriving requirements from typical scenarios, for coming up with a conceptual design, for implementing this design, and for evaluating whether the realized infrastructure meets the postulated requirements. Since engineering a service-oriented architecture that enables the automation of the contracting process requires the integration of various technologies, we propose an integrated methodology in this chapter, which is used to structure the remaining thesis. In addition, we define in this chapter the expected results of the engineering process which is a *Web service market infrastructure* that enables providers as well as customers to automate the contracting process.

In many aspects service-oriented architectures are fundamentally different from traditional distributed component- or object-based frameworks. Most notably they introduce the roles of providers and requesters as first class citizens in the architecture. This is an important step towards truly inter-organizational business processes since it allows us to directly apply concepts and ideas traditionally developed for electronic commerce. Moreover, the concept of dynamic service binding is introduced that requires automated discovery, selection and invocation of new services. In contrast to an object in object-oriented systems, which represents a (physical or non-physical) real world "thing", a service instead represents a business activity and thus is directly usable within a business process. These fundamental distinctions lead to three types of components required for implementing a service-oriented architecture [GB01]:

1. A *hosting platform*. This is where service providers can deploy and operate their service. The hosting platform also has to provide means for requesters to invoke the service.

2. A *hub* that connects service providers and requesters. This hub should enable dynamic discovery of suitable services. Therefore, means for describing services hosted on different platforms in an interoperable way have to be provided.

3. *Standard conventions* that ensure that services can interoperate with each other irrespective of their implementations. This involves communication and interaction protocol aspects.

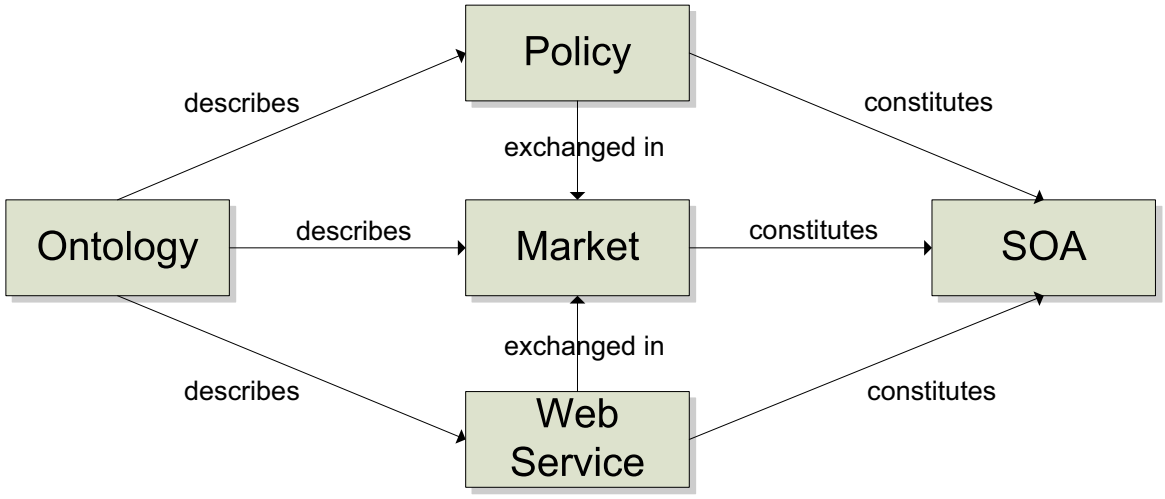

Figure 3.1: The semantic Web service market diamond illustrating the relation between the involved technologies that constitute a SOA.

In order to realize these components, we propose to combine different technologies captured by the diamond structure in Figure 3.1: (*i*) As introduced in Section 2.1.3, *Web service technologies* provide a hosting platform and a set of standardized protocols, formats and language specifications that enable interoperation between services hosted on different platforms. Therefore, they partly cover component No. 1 and 2 introduced above. (*ii*) In this context, it is important that each party is able to exactly define the capabilities and characteristics of the services that are provided or required. For example, a provider might state that her service is only accessible after a certain authentication method has been carried out. Such constraints and conditions are defined using *policies*. This is important to find out which providers and requesters can be connected by the hub. Thus, policies partly cover component No. 2. (*iii*) *Markets* – as the place where service providers, requesters and intermediaries come together to advertise their services and requests – are the cornerstone of service-oriented architectures [EL04]. Thus, results from the area of market theory should be considered when designing a hub that provides technologies for mechanism to dynamically determine suitable bindings and methods for closing legally enforceable and manageable contracts. Determining suitable bindings may involve locales for negotiating with and selecting among many potential providers. Since markets bring together requesters and providers they are required to realize component No. 2. (*iv*) In order to enable automated discovery, selection and negotiations, metadata about services has to be specified in a formal, machine-understandable way. Our technology of choice for formally describing services and their policies are *ontologies*. They come with a standardized logical foundation providing well-defined semantics that is required for matching of descriptions and for realizing a high degree of interoperability. In addition, ontologies can be used to describe market mechanisms allowing a flexible and adaptive market implementation. Therefore, they are important for component No. 2 and 3.

Realizing a seamless integration of the different technologies is essential for designing a Web service market. For this purpose we introduce the methodologies required for designing service-oriented architectures, electronic markets and ontologies and identify relations between them (Section 3.1). Subsequently in Section 3.2,

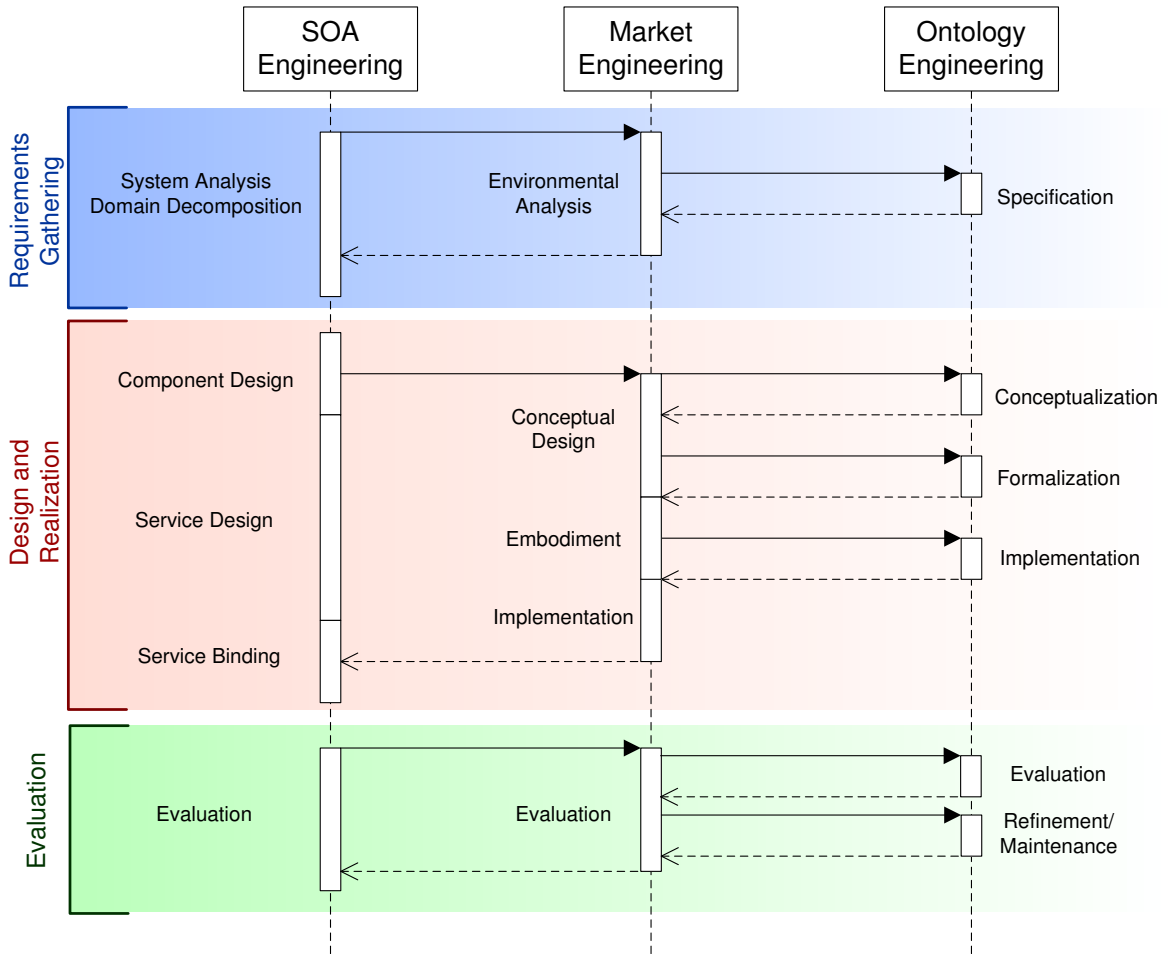

Figure 3.2: Integration of service, market and ontology engineering for developing semantic Web service markets.

we discuss the desired results of the methodology. In this context, we identify the main building blocks of a Web service market including a set of ontologies providing the communication primitives in the market and a market process bringing together the different phases that can be identified in the market with the publish-find-bind-execute paradigm of service-oriented architectures.

3.1 Methodology

In order to obtain a system that seamlessly integrates different technologies, the integration has usually to happen already at design time. As discussed in the previous section, developing a SOA infrastructure also requires developing an electronic market bringing together requesters and providers. Each market requires a communication language for exchanging offers, requests and contracts. In our case, this is realized by introducing appropriate ontologies. Although each of these components comes with an individual development methodology, the corresponding development processes cannot be executed independently, since there are several interdependencies in terms of time and required information. These interdependencies are captured by Figure 3.2. For example, one cannot finish the environmental analysis in the market engineering process, before the specification phase of the ontology en-

gineering has not been concluded. In fact, the requirements gathering phases have to be finished, before the design and realization phases can be approached, and the design and realization phases have to be finished, before the evaluation phases can be started. However, note that each engineering process itself does not have to be executed sequentially.

In order to identify and cope with the interdependencies that occur in the Web service market development process, we introduce the design methodologies for service-oriented architectures (Section 3.1.1), markets (Section 3.1.2) and ontologies (Section 3.1.3) and identify overlapping areas where the different processes have to be aligned and synchronized. We aim thereby at a coherent development process for *semantic Web service markets* as sketched in Figure 3.2.

3.1.1 SOA and Web Service Engineering

First, we address the question of how a Web service and service-oriented architectures as a whole are built. Generally, there is an enormous amount of literature dealing with engineering software systems ranging from sequential methods such as the influential waterfall model [Roy70] to iterative models which combine top down and bottom up approaches such as the spiral model [Boe88] or the Rational Unified Process (RUP) [Kru03]. However, although service engineering can be seen as a special case of software engineering[1], when moving to a service-oriented architecture these methodologies have several shortcomings neither addressed by object-oriented analysis and design nor by business process management techniques:

- First, the question of how services can be identified has to be answered. In order to identify the different aspects required for developing a service-oriented architecture, business process aspects as well as the enterprise-scale application architecture have to be taken into account. Obviously, object-oriented analysis is a good starting point, but it does not address how to discover the functional units of work from a business perspective required for identifying a reusable set of services.

- Second, a paradigm shift towards explicitly appreciating the key roles found in service-oriented systems is required. As shown in Figure 2.1 on page 13, the key roles are service provider, service requester and service broker (a passive broker is called a registry).

- Third, the methodology should reflect the fact that services are not built for one single business line or company, but are potentially exposed to other departments or companies.

With these ideas in mind, several methodologies have been defined that are explicitly tailored towards Web service engineering. In the following, we look more closely at the service-oriented analysis and design methodology (SOAD) propagated by IBM [ZKG04] and the service-oriented design and development methodology proposed in [PH06]. While the overall process strongly conforms with the

[1] Note that our focus is engineering of software services. For the broader field of service engineering beyond software services several mature approaches exist, which are beyond our discussion. For an overview refer to [SDBW03].

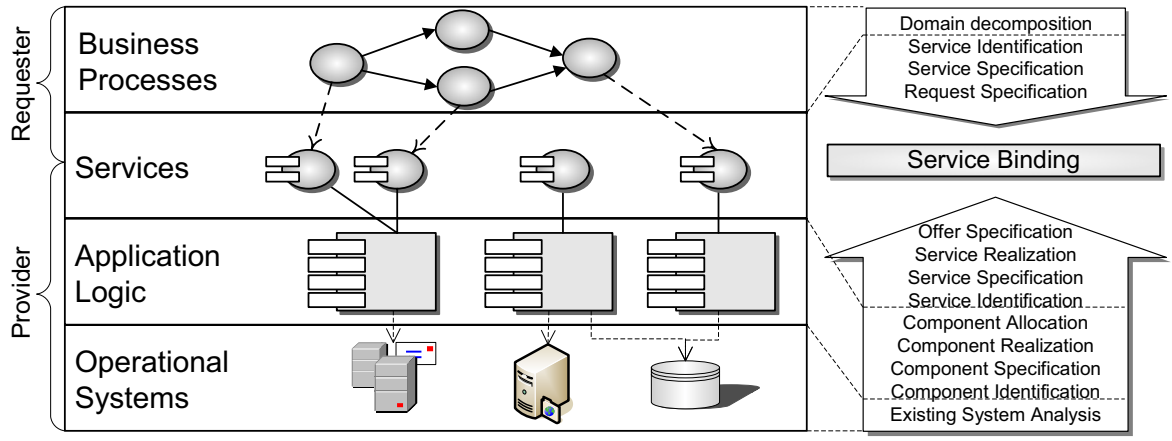

Figure 3.3: SOA layers and Web service engineering process

standard software engineering process [AMBD04] comprising the steps *requirement analysis*, *design*, *construction*, *testing and maintenance*, the design phase in particular has been refined.

Figure 3.3 relates the steps that have to be conducted in the design phase with the layers of a typical service-oriented architecture. From a software engineering perspective a service-oriented architecture can be broken down into four layers:

1. The operational system providing the basic hardware and software for running an application, such as servers, operating systems, Java virtual machines, etc.

2. The application logic within software components implementing the actual functionality, e.g., by means of an object-oriented programming language.

3. Services encapsulating functionality in self-contained activities and providing a well-defined interface to internal or external requesters.

4. Business processes combining the activities accessible via services. Often, the functionality of a business process is exposed again by a service to customers. Therefore, further business process and service layers can be added on top (Figure 3.3 abstracts from this fact).

The design process (represented by the arrows in Figure 3.3) combines a business-driven top-down approach with a bottom-up approach leveraging legacy assets, while considering the different roles in the system. The service infrastructure (layers 1 and 2) is set up by the provider, whereas layer 4 resides on the requester's side. In layer 3 the interaction between the provider's and the customer's system takes place. In the following, the design process is described in more detail:

System Analysis

The bottom-up technique starts with an *analysis of the existing system*. The goal is to find resources that could serve as a basis for providing service functionality. System analysis is typically executed by the provider.

Domain Decomposition

The top-down approach starts with the *domain decomposition* step. Here the business process is decomposed into its subprocesses using high-level business use cases that enable identifying the functionality required as service. Domain decomposition therefore requires extensive knowledge about the requester's business models and use cases.

Component Design

In the next step, viable *components* that implement the application logic required for a service are *identified*. In the *component specification*, the messaging and event specification, the internal flow and structure of the identified components and other component dependencies are described. Missing components have to be custom built in the *component realization* step. After realizing the application logic, its functionality can be exposed by providing an appropriate service. Defining the component that implements a certain service is called *component allocation*. In Figure 3.2, the component identification, specification, realization and allocation is captured by the term *component design*.

Service Design

Similarly, *service design* can be broken down to service identification, specification and realization. The *service identification* step deals with deciding which operations of the component should be accessible via a Web service. Since this is difficult without knowledge of the business domain, a top-down approach is needed, in which the results of the domain decomposition step is utilized. After identification of the services their characteristics have to be documented to enable their implementation and later reuse. This is done in the *service specification* step. In order to make services discoverable in a service-oriented architecture (e.g. required for dynamic binding), the service specification has to be machine-interpretable. For this purpose WSDL and ontology-based specifications are used that allow, e.g., defining quality of service policies or information about the business process implemented by the service. Typically domain-specific ontologies are required in this step. If such ontologies are not available for a certain service they have to be built. Thus, an ontology engineering phase has to be initiated, which is discussed in Section 3.1.3.

Service Binding

Once services are realized and exposed by the provider on a service market (*offer specification*), requesters can integrate them into their business process. Therefore, they have to decide in a top-down manner by means of domain decomposition, which services are required and which task of the process should be done by which service (*request specification*). In Section 2.1.2, we introduced the term binding for the assignment of service requests to offers. As discussed, these bindings can be specified explicitly by the developer or determined dynamically by the system using the request and offer specification. In either case, the mechanisms bringing together service demand and supply have to be carefully designed, which is the main purpose of the field market engineering discussed in Section 3.1.2.

Evaluation

Finally, the constructed architecture is *evaluated* in terms of functionality, robustness, efficiency, etc. While these step mainly corresponds to the traditional software engineering methods, some additional criteria are important, such as reusability of services, efficiency of provider selection, etc.

Since service-oriented architectures require mechanisms that bring together service requesters and providers, we discuss in the next section, how such mechanisms can be developed in a structured manner.

3.1.2 Market Engineering

Building an electronic market that meets certain requirements, such as welfare optimization or maximizing the transaction volume in the market, is a complex process. Therefore, the *market engineering process* breaks down this complex process into less complex sub-phases very much as software engineering does it for the implementation of complex software systems. Although structured according to similar phases as software engineering, market engineering focuses on different aspects and goals. In this section, we briefly introduce the different phases. A more fine-grained process accompanied with a detailed discussion for each phase can be found in [Neu04].

Environmental Analysis

The *environmental analysis* deals with gathering information about the concrete setting for which the market is designed, including information about the participants, about the products to be traded, about possible intermediaries, etc. This first step is called *environment definition* and directly makes use of information derived from domain decomposition of the service engineering process (Figure 3.2). In Section 4.1, we perform this step for Web service markets by analyzing different concrete scenarios. Based on the environmental definition, requirements can be derived that should be met by the market. For example, due to the easy differentiability of Web services, we will need multi-attribute product descriptions as part of offers and requests. Such language-specific requirements in the market engineering process are also direct requirements for the expressivity and the vocabulary of the ontology as shown in Figure 3.2.

Design and Implementation

After identifying the requirements, the market algorithms and infrastructure can be defined and implemented. First, in the *conceptual design phase*, the market is set up in an abstract way, i.e. the institutional rules are defined without specifying the way they are implemented. In our case, this mainly requires the design of a bidding language and market mechanism that provide matching, allocation and contract formation functionality. We will introduce the conceptual design for Web service markets in Chapter 5 using an abstract, implementation-independent mathematical notation. As depicted in Figure 3.2, the design and implementation phase can be done in parallel to the component and service design. However, since at the end of these phases, service offers and requests have to be specified and the market bidding

language (e.g. including the formal ontology model) has to be present before the service design phase can be finished.

Based on this conceptual design, the market can be implemented. In this context, one can distinguish between the *embodiment phase* and the actual *implementation phase*. During the embodiment phase, the abstract conceptual design is concretized, but still remains platform independent. For example, the bidding language is concretized by formalizing the appropriate core, domain and application ontologies using a concrete ontology language. Thus, the market design and implementation phase has to be accompanied by the conceptualization, formalization and implementation of the appropriate ontologies (Figure 3.2). As mentioned before, ontologies are independent of a concrete implementation platform and therefore they can be implemented in the embodiment phase. In this work, the embodiment phase is realized in Chapters 6 and 7.

Finally, in the implementation phase the market platform is realized. This involves, for instance, the implementation of the required matching and allocation algorithms, integration of an appropriate ontology reasoner, the installation of a Web server for deploying the market, etc. The implementation of our Web service market is presented mainly in Chapter 8.

Evaluation

Once the market infrastructure is set up, it can be *evaluated* whether the desired market outcome can be realized. Usually these evaluations are done with respect to the requirements specified in the environmental analysis and include technical aspects (e.g. performance, system reliability) and economic aspects (e.g. efficiency). Of course, evaluation of the market as well as of the service-oriented architecture may reveal problems which have to be corrected by going back to the corresponding engineering phase. In this context, also major revision of the ontologies could be necessary. For the Web service market presented in this thesis evaluation is performed in Chapter 9.

Since the expression of offers, requests and contracts in a market typically includes complex description of goods or services that may involve broad domain knowledge as well as a wide-range of different parties, defining an appropriate market language easily becomes a cumbersome task. Therefore, ontology engineering provides structured means that support this task. The ontology engineering process in sketched in the following section.

3.1.3 Ontology Engineering

Several ontology engineering methodologies have been proposed in literature serving different purposes or addressing different domains [CFLGP03, PM04]. For example, for ontology building from scratch TOVE [GF95], ENTERPRISE [UK95], METHONTOLOGY [LGPJ97, LGPSS99], the OTK-methodology [SSA+01] and DILIGENT [TPS06] have been proposed. Pinto and Martins [PM04] compare ontology engineering methodologies using a general process containing the stages *specification*, *conceptualization*, *formalization*, *implementation*, and *maintenance*. Although the tasks classified within a certain stage differ slightly from one methodology to the

next, these stages are most suitable as a brief introduction to ontology engineering, since they abstract from a specific methodology.

Specification

The objective of the specification stage is to identify the scope of the ontology. In this context, the domain that has to be captured and the intended users have to be specified. This also involves to determine requirements regarding the expressivity of the ontology language. For application and task ontologies also application-specific and task-specific requirements have to be considered.

Conceptualization

In a second step, the identified specification is described with a conceptual model. As shown in Figure 3.2, ontology conceptualization is part of the conceptual design phase of the market engineering process and done during component and service design. Depending on the concrete methodology used, different conceptualization models ranging from informal models, such as mind maps^{TM}, to semi-formal models, like binary relations diagrams, might be used. These conceptualizations describe the basic concepts and relations relevant in a domain. Moreover, vocabulary is clustered into groups for modularization purposes. This is essential for later reuse of ontologies and for avoiding the problem of over-commitment [Obe05].

Formalization

After describing the required vocabulary and the relations between the vocabulary terms in a conceptualization, the conceptualization has to be formalized in order to get an unambiguous definition of the terms. The formalization stage involves defining concepts by restricting their interpretation to certain individuals in the domain (see Section 2.4.2). Thus, concepts and relations are mathematically well-defined, but are not yet serialized in a computer-interpretable format. Since the result of the conceptual market design should be a formal model, ontology formalization is also a part of the conceptual design phase of the market engineering processes (shown in Figure 3.2).

Implementation

In the implementation stage, the formalized and semantically well-defined model of the ontology is represented by means of a machine-interpretable syntax, as provided by OWL, for instance. Since the ontology is still platform- and implementation-independent, this implementation of the ontology is usually part of the embodiment phase of the market engineering process. Furthermore, a fully implemented ontology is already required when entering the implementation phase of the market engineering process.

Evaluation

In this stage, the quality of the ontology is technically judged by a knowledge engineer. According to [PM04], this includes verification (i.e. is the ontology correct

according to the accepted understanding of the domain), validation (i.e. does the ontology meet the specified requirements), and user assessment (i.e. judging the usability and usefulness of the ontology and its documentation). Since up to now there are no mature ontology evaluation methods available, this area requires major research efforts. For a detailed discussion of ontology evaluation methodologies refer to [GP04, VSFGS06, OCM+07].

Maintenance

During testing of the service-oriented architecture as well as of the market mechanism, updating and correcting of ontology modules might be required. Each update or correction should be verified carefully and the implemented ontology should be checked for consistency.

After introducing the process for development of semantic Web service markets that integrates service, market and ontology engineering into one coherent process, in the next section we discuss the results of this design process in more detail. In particular, we answer the question: what are the artifacts that should be constructed and how can these artifacts be used to realize dynamic service contracting and monitoring in service-oriented architectures? Afterwards in Part II and III of this thesis, we then apply the methodology introduced above to design, implement and evaluate a semantic Web service market infrastructure.

3.2 Core Building Blocks of Web Service Markets

In this section, we view semantic Web service markets from two orthogonal perspectives reflecting the two main building blocks of electronic markets [Neu04]: first, we discuss the communication primitives that constitute the market information model in Section 3.2.1. Essentially this breaks down to a discussion about the role ontologies can play in such markets. Second, in Section 3.2.2 we consider the market mechanisms required for contracting and monitoring of Web services. This involves an exact definition of a Web service market process implementing this functionality. We realize this by specializing the general market process (e.g. defined in [TBP03, SW03]) to a process capturing the find-bind-execute paradigm of a service-oriented architecture. To put it simply, in the following we introduce the result of the engineering process distinguishing between the static aspects of a market that is modeled using ontologies and the dynamic aspects implementing the market process.

3.2.1 The Role of Ontologies

As defined in Section 2.1.1, the three main communication primitives required in a service-oriented architecture are offers, requests and contracts. Since a detailed requirements analysis regarding the desired expressiveness and properties of these primitives is done within the market engineering process in Part II of this work (Chapter 4), this section is limited to a brief discussion of the benefits that can be realized using ontologies for representing the primitives. This discussion is accompanied by a short introduction to the ontologies available in this context.

In Section 2.4, we introduced the concept of ontologies as a means for achieving interoperability through the specification of standard syntax and semantics, and through their well-defined grounding in logics which enables improved matchmaking of offers and requests in the market. Several proposals for using ontologies in electronic markets have been put forward that exploiting these features. The goal is to reach a degree of *openness*, *flexibility* and *dynamism* not achievable with traditional technologies for B2B integration such as RosettaNet[2], UNSPSC[3], etc. [DFK+04].

One major aspect in this context is the unambiguous description of the products exchanged in the market. Product descriptions have to be part of offers, requests and contracts. Such product information is subject to continuous changes due to the introduction of new products, evolution of products, changes in the organization due to internal reorganization or fusion-acquisition (e.g. fusion of product lines, reorganization of hierarchies or policies) and therefore product information is often difficult to capture in traditional relational databases [BMW+07]. In this context, OWL ontologies provide important advantages, since they allow class inheritances that feature product categories and logical classes that enable automatic classification of products according to OWL restrictions. In [BMW+07], this is illustrated using the following examples:

- The class "Outdated Products" can be introduced that dynamically classifies all products that are replaced by at least one other product, i.e. $\mathit{OutdatedProducts} \sqsubseteq \mathit{Products} \sqcap \exists \mathit{replacedBy}.\top$.
- The class "Metallic Products" capture all products the are made purely using metal, i.e. $\mathit{MetallicProducts} \sqsubseteq \mathit{Products} \sqcap \forall \mathit{madeBy.Metal}$.

As we will see later, these are advantages that carry over to descriptions of Web services, which naturally are the products dealt with in a Web service market. The term *Semantic Web Services* captures the idea of describing Web services using ontologies for improving discoverability, composition, mediation, etc.

Semantic Web Services

The goal of semantic Web services research is the automation of certain management tasks within a service-oriented architecture, such as discovery of suitable Web services, the composition, interoperation and execution of Web services [MSZ01]. In [SGA07] the following definition is given:

> "The Semantic Web Services vision is to semantically annotate Web Services with machine interpretable meta data, such that computer programs are enabled to reason about their functionality. In this way, various kinds of services, such as book selling, shipment of goods or provision of stock market information, can be advertised and discovered on the Internet in an automated way, and their functionalities can be combined in composite services at run-time in order to achieve higher level goals. Semantic Web Services particularly aim at realizing smooth information integration through flexible architectures within and across organization boundaries."

[2] http://www.rosettanet.org/
[3] http://www.unspsc.org/

For semantic annotation of Web services, ontologies are the technology of choice and several ontologies providing appropriate vocabularies have been proposed. The most prominent approaches are OWL-S [SPAS03], SAWSDL [W3C07a] and WSMO [DKL+05]. More specific ontologies (or extensions to the previous) are provided for topics such as quality of service modeling [TFJ+06] or specification of the temporal behavior of a Web service [AS06, BFM06].[4]

As discussed in Section 2.2, policies are required within communication primitives to specify constraints and preferences on service properties. However, since policies are also relevant in other domains they are usually modeled in their own ontology module.

Policy Ontology

In recent years, a set of policy ontologies have been proposed that enable the representation of constraints. The most influential approaches are KAoS [UBJ+04], REI [Kag04] and an ontology that formalizes WS-Policy [KPKH05]. The advantage of using ontologies is that they provide a high degree of formalization with respect to the classification introduced in Section 2.2.3. This provides features, such as automated consistency checking as well as conflict handling, and improves interoperability in heterogeneous systems. As we will see later on, by means of policies, highly configurable service offers or requests can be efficiently represented and exchanged.

Market Ontology

Market mechanisms represent a set of rules that determine how the market process (e.g. service contracting) has to be carried out. Ontologies can be used to declaratively specify these market rules and thus enable a high degree of flexibility. For example, rules can be easily added and changed at runtime of the market, the market behavior can be adapted to different contexts, and the market participants can download and understand these rules, since they have a standardized syntax and well-defined formal semantics. Several approaches that declaratively represent market mechanisms can be found in literature [WWW01, RWG01, ETJ04, TPDW05]. However, only [RWG01, TPDW05] make direct use of ontologies.

After having introduced the building blocks that constitute the information model of a Web service market, the following section focuses on the other main design object: the Web service market process which comprises algorithms for contracting and monitoring of Web services.

3.2.2 The Contracting Process

To develop a mechanism for contracting in Web service markets, the find-bind-execute-paradigm of service-oriented architectures (see Section 2.1.1) has to be aligned with the general market process introduced in Section 3.2.2. In fact, the find-bind-execute-paradigm on which service-oriented architectures are based can be seen as a specialization of the general market process introduced in Section 2.3.1.

[4] A more thorough discussion of the ontologies discussed in this section can be found in Chapter 10.

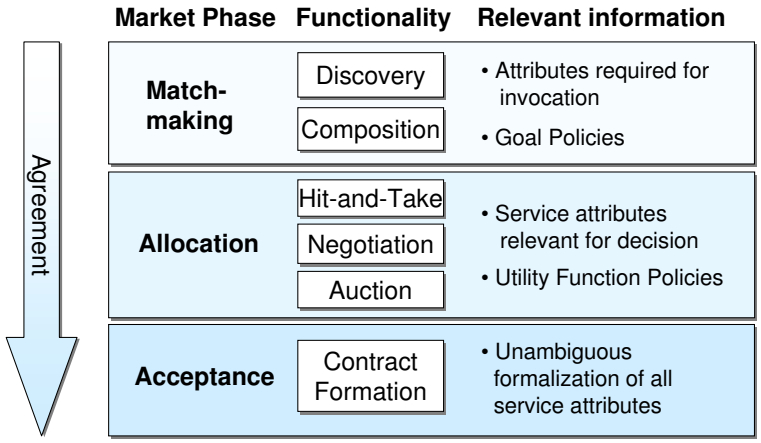

Figure 3.4: Market phases and the Web Service usage process

This means, when moving from systems operating within one company to systems that involve different, independent companies, the find-bind-execute-paradigm describes nothing other than a B2B procurement process, where digital services such as information delivery or execution of calculations are purchased. Thus, service-oriented architecture requires an infrastructure that provides an institution for coordinating between service requestors and providers. This coordination mechanism has to provide a platform where potential business partners can be discovered, prices can be ascertained, and contracts can be closed. As discussed in Section 2.3, a market, where prices are determined by the interplay between supply and demand, can be regarded as a coordination mechanism that efficiently provides these functionalities [Hur73].

Figure 3.4 brings together the agreement phases that can be identified in an electronic market (see Section 2.3) and the typical Web service usage process which comprises the steps *discovery*, *composition*, *negotiation*, and finally *contract formation*. In the *Matching Phase* suitable services are discovered. For discovery of Web services, we consider only attributes that are mandatory for invocation of a service and for integrating the results. This usually includes attributes such as the input and output of a service or attributes describing the behavioral characteristics of the service process.[5] Typically these attributes are constrained by goal policies, which are evaluated within the matchmaking process. Whether a certain attribute has to be considered within matchmaking process depends on the concrete domain and offer-/request description. Since a certain goal can not be accomplished only by a single service but also by a combination of services, this phase also includes composition.

After having determined services that are able to achieve a certain goal, an optimal assignment of service requests and offers with respect to the individual utility of the participants or to the overall welfare has to be found in the *Allocation Phase*. To achieve this, an allocation mechanism is required that determines the concrete terms of the transaction. Mechanisms featuring a fixed-price mechanism, such as a Hit-and-Take, or dynamic pricing mechanisms that involve negotiations or auctions

[5] In literature, such attributes are often called *functional properties* of a Web service. Correspondingly, attributes not required in the matching process are denoted by *non-functional properties*. However, since there is no clear definition (depending on the domain attributes might be functional or non-functional), we refrain from adopting this nomenclature.

could be used. In this phase, attributes are considered that represent decisive factors for service selection and price determination. This could also include attributes already used in the matching phase. Typical attributes are payment methods, security as well as trust characteristics, and most notably quality of service attributes.

After determining the allocation, legally binding contracts are closed between the corresponding business partners in the *Contract Formation Phase*. Especially dynamic contract formation at runtime is a big issue, since in this case the contract formation has to be done by agents without human intervention. Contracts have to be formalized in a machine-understandable way in order to enable automated contract management in the settlement phase (e.g. automated execution and monitoring of contracts).

3.3 Conclusion

In this chapter, we have aligned different technologies, namely service-oriented architectures, electronic markets and ontologies, in order to develop a coherent methodology and architectural view for establishing a semantic Web service market. In Part II and III of this work, the methodology introduced in Section 3.1 is applied: in Chapter 4, the requirements for the market are elicited using concrete scenarios. Based on the derived requirements, in Chapter 5 the conceptual design of the market is presented, which involves defining algorithms for all market phases as well as the conceptualization and formalization of the required ontologies. Chapter 6 and 7 covers the embodiment phase where the ontologies are serialized using a concrete ontology language. The market is then implemented in Chapter 8 and finally tested in Chapter 9. Applying this methodology consequently results in an ontology framework for expressing Web service offers, requests and contracts, and a set of market mechanisms for trading Web services in an open, Web-based market.

Part II

Designing a Semantic Web Service Market

Chapter 4

Scenarios and Requirements

This chapter presents scenarios for service markets and derives requirements from these scenarios. This represents the first stage of the market engineering process. We analyze three scenarios, which represent typical use cases for applying service-oriented architectures. According to our research statement presented in Section 1.2, the requirements have to cover two main aspects of a market: language-specific and mechanism-specific aspects.

This chapter is structured as follows: Section 4.1 introduces three scenarios representing the major use cases of service-oriented architectures in current businesses. Based on these scenarios Section 4.2 addresses the requirements a market for services has to meet. In Section 4.3, we discuss the importance of the requirements for the different scenarios and outline our approach to meet the identified requirements.

Parts of this chapter are published in conference proceedings. The scenario and the corresponding requirements for enterprise services are discussed in [LEO05, LAO+06], for grid services in [LS06] and for mobile services in [LAGS07].

4.1 Scenarios

The scenarios introduced is this section are meant as a starting point for determining the requirements for a semantic Web service market. The scenarios capture the main application areas of service-oriented architectures and Web service technologies, namely enterprise service architectures, mobile services, and grid/utility computing [SH05, Pap03]. In order to show their broad applicability, we first describe them in a domain-independent way and later augment them with a concrete example. The scenarios differ in the type of services they provide as well as in the environment of the architecture. However, they also exhibit important commonalities:

- a set of providers is available offering functionally equivalent services;
- services are configurable (e.g. they can provide different quality of service levels);
- different participants in the market have different policies;
- these policies could depend on the execution context;
- automation of the market process is at least partially required;

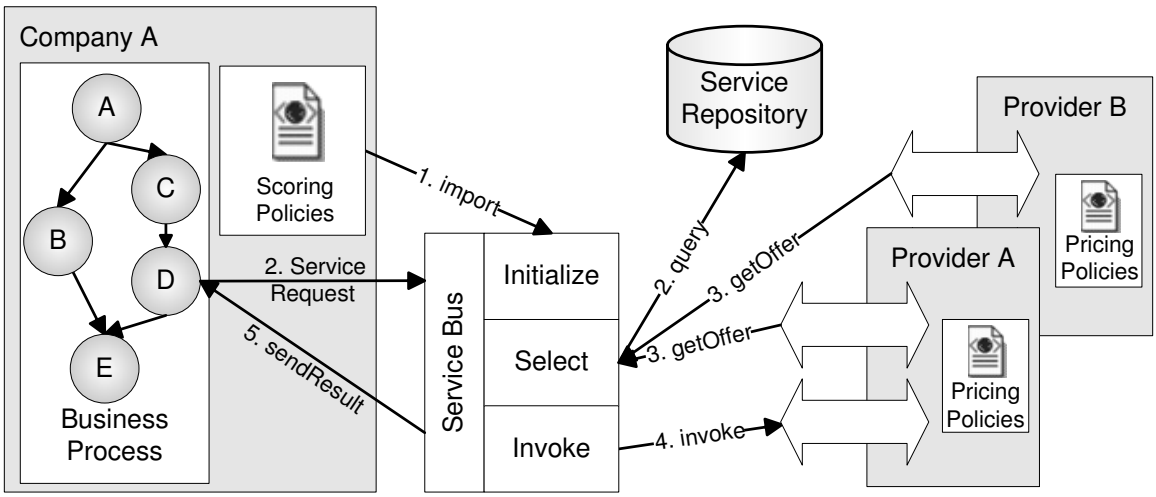

Figure 4.1: Service Bus Architecture [LEO05, LAO+06].

- and the environment is open, which refers to information systems that involve components from different organizations that are *autonomic*, typically highly *heterogeneous* and may *change dynamically* [SH05].

4.1.1 Enterprise Services

One of the driving forces behind the development of service-oriented architectures has been the need for adaptive software solutions that enable a seamless integration of software components from different vendors. The envisioned benefits are a faster redesign of a company's business processes and applications, facilitated outsourcing of functionality, and smooth application integration. The different building blocks that constitute a company's application are called *enterprise services*. Examples for enterprise services are billing services, stock quote services or order processing services. Such enterprise services are provided by companies such as SAP, Google or Amazon. For example, consider the supply chain management application *mySAP*™ SCM, which is composed of different enterprise services such as Order Processing, Transportation Planning, Billing and Invoicing, and Parts Monitoring.

The infrastructure that enables the combination of different enterprise services for an application is called an *enterprise service bus* [Sch02, Rob04, Cha04, Ley05]. Depending on the concrete definition, an enterprise services bus comprises functionality such as (dynamic) service discovery, selection, composition and execution, data integration and mediation, monitoring of executions, and wrapping of legacy software components and protocols.

Figure 4.1 sketches a simplified architecture of a service bus architecture. On the left side, a company's business process is visualized as a workflow of tasks that have to be accomplished by an enterprise service. The company further defines general policies about how the business process should behave. For example, these policies directly influence the selection of an appropriate service. They might include the company's preferences about Web service characteristics. Figure 4.1 exemplifies the automated integration of a Web service at runtime. This means, in order to enable

such dynamic binding of services using the enterprise service bus, the following steps have to be carried out, once a Web service is required within the business process (step D in Figure 4.1):

1. In order to automate the selection process, relevant policies are sent with the first service request to the service bus and are stored there locally. Note that this initialization step only has to happen once for the initialization of the service bus. Based on the policies, the service bus is able to take over the responsibility of selecting between potential providers, such as A and B. In this sense the service bus can be seen as a simple Hit-and-Take market mechanism (see Definition 2.7). Since the decision for a certain service might depend on runtime information, different policies might have to be specified for different contexts (current location, time, etc.).

2. Once a request from an application arrives, the service bus first queries a service repository (such as a UDDI repository) for suitable providers. Here only a very simple matching of the service functionality is carried out. This means only the addresses of services are returned that provide the required functionality.

3. In a next step, offers from the providers are collected in parallel. These offers contain a list of provided configurations together with the service policies of the corresponding provider. These are also stored in the knowledge base of the service bus.

4. Finally, the service bus queries the knowledge base for all service offers and configurations that fulfill the required functionality. A list of services ranked according to the difference between score and price is returned. Based on the ranking, the best provider is selected and the respective service invoked. In case this invocation fails, the second best service is chosen. This is repeated until the required task is accomplished or no acceptable service remains.

The service bus scenario described above provides the benefits that come with dynamic binding mechanisms (see Section 2.1.2). To realize these benefits, reliable automated contracting of enterprise services is required, since many enterprise applications are business-critical and thus the contracted services have to be chosen carefully. In addition, legally enforceable contracts between service requesters and providers are required to provide security in case of problems during service execution (e.g. contracts have to regulate what happens if the service is not provided as agreed). In this context, it is essential that the contracts capture all important, price-relevant attributes of a Web service. A list of such Web service attributes that are independent across domains are presented in [OEH02, Ran03, CSM+04]. Thereby, service differentiation as described in Section 2.1.2 is enabled. Especially for service providers such differentiation is essential in order to compete with other providers. For example, this could be realized by providing better quality of service guarantees for the same price or services that more closely meet the requirements of the customer.

In the following, we exemplify the usage of enterprise services using an example from the financial domain.

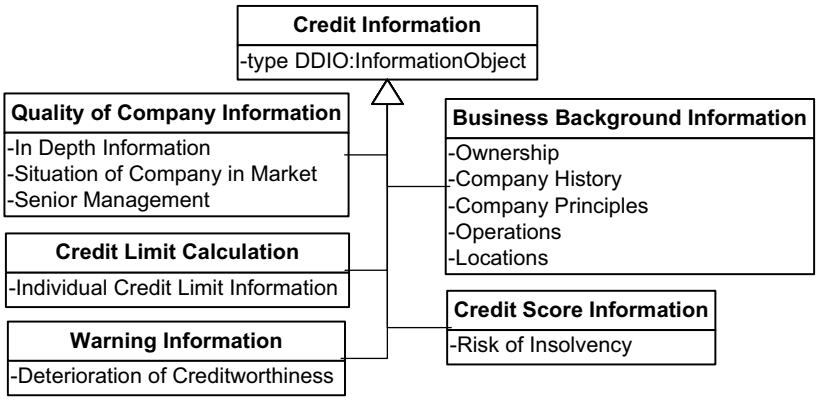

Figure 4.2: Hierarchy of financial information [LML+05].

Example 4.1 *In order to reduce credit risk and to select profitable customers, companies rely upon credit information. The latest legal developments around risk management such as Sarbanes Oxley have forced companies to have a closer look at the management of financial risk. Financial information relating to the creditworthiness of companies, the profitability of their business or the quality of their senior management helps companies to assess the risk of doing business with each other and respond to increased or decreased risk. Such financial information is sold by companies such as Dun & Bradstreet or Creditreform. Based on credit information, companies will decide whether to start business with another company or determine and adapt lines of credit. In the past, such lines of credit have often been considered too late as the buying of credit information was done manually and not always on a continuous basis. By providing this critical credit information via Web services the credit information can be integrated into existing enterprise applications facilitating risk decisions based on externally provided and permanently updated data. Often the integration of the services has to be done dynamically based on transaction specific information, such as the origin of the business partner, type of transaction, etc.*

From a technical point of view, such financial services are functions that typically take the name of a company as input and return certain financial information that can be classified according to the hierarchy shown in Figure 4.2. In addition, service levels guaranteed by the Web service providers are an essential decision criteria for the provider selection. Wrong financial information could lead to a wrong decision and thus to a huge financial loss. In the case of financial information services, the typical quality of service attributes are update time of the delivered information, delivery time (response time), warranties about the accuracy of the information, etc. In addition, other relevant attributes have to be considered during service selection such as the payment terms. A customer's IT management infrastructure has to include execution monitoring of the service usage and checking for conformance with the contract [CSDS03].

4.1.2 Grid Computing

As defined by [Fos02], a grid is a distributed computing infrastructure that coordinates resources without central control, that is based on open, general-purpose, standard protocols and interfaces, and delivers services under various, nontrivial quality of service guarantees. With the Open Grid Services Architecture (OGSA) there is an increasing trend towards providing grid resources via Web services tech-

nologies. This is realized by enabling stateful Web services through the WS Resource Framework [FMS06], which enables a service manage numerous parallel session with requesters. Although from a technical point of view, they are very similar to enterprise services (e.g. based on Web service protocols and languages, open environment, distributed control), in contrast to enterprise services, grid services are directly connected to the underlying resources. While the provision of enterprise services takes a working computing infrastructure for granted, grid services deal with providing this computing infrastructure in a flexible way. Similar to enterprise services they can be combined on demand in order to react on changing application needs (e.g. to deal with peak demand). This also requires a dynamic contracting process as is the case for enterprise services. Typical examples for grid services are services which provide CPU processing time to applications or where data can be stored for a certain time period. One of the first major commercial providers of such services via a Web service interface are Amazon, SUN and Google.

As already mentioned in the definition above, an important aspect of grid systems are the quality of service attributes of the capabilities they provide. This usually includes response time, throughput, availability, and security, and/or coallocation of multiple resource types to meet complex user demands. Allocation of grid services also has to consider possible resource limitations of providers. This is usually not considered in the case of enterprise services, since they are not directly related to specific resources and thus resource limitation can be handled at provider side (e.g. by assigning more computational power to a service). Moreover, most enterprise services are pure information services that can easily handle a considerable amount of requests and therefore resource limitations are not a problem.

Example 4.2 *Increasing demand for high-performance computational resources in academic as well as commercial organizations has lead to numerous initiatives such as UK'S e-Science, Germany's D-Grid or IBM's utility computing program aiming at the provision of on-demand computing resources. Recently the first commercial grid services have been released that can be used by customers "on demand" and are paid for by use. Examples are the Amazon Elastic Compute Cloud (Amazon EC2)*[1] *services which provide a virtual computing environment. They provide the following quality of service guarantees:*

- ***Guaranteed performance**: power equivalent to a system with a 1.7Ghz x86 processor, 1.75GB of RAM, 160GB of local disk, and 250Mb/s of network bandwidth is provided.*
- ***Reliability**: replacement of system instances provides a high degree of reliability, for stored data availability of 99.99% is guaranteed*
- ***Security**: Web service interfaces to control network security (customizable)*

Prices are calculated according to the time for which the system is required: $0.10 per instance-hour consumed (or part of an hour consumed), $0.20 per GB of data transferred into/out of Amazon (i.e., Internet traffic). In addition, the services has to be bundled with the Amazon Simple Storage Service (Amazon S3)[2] *for which a price of $0.15 per GB-Month*

[1] `http://www.amazon.com/gp/browse.html?node=201590011`

[2] `http://www.amazon.com/S3-AWS-home-page-Money/b/ref=sc_fe_l_2/002-0629126-4701622?ie=UTF8&node=16427261&no=3435361&me=A36L942TSJ2AJA`

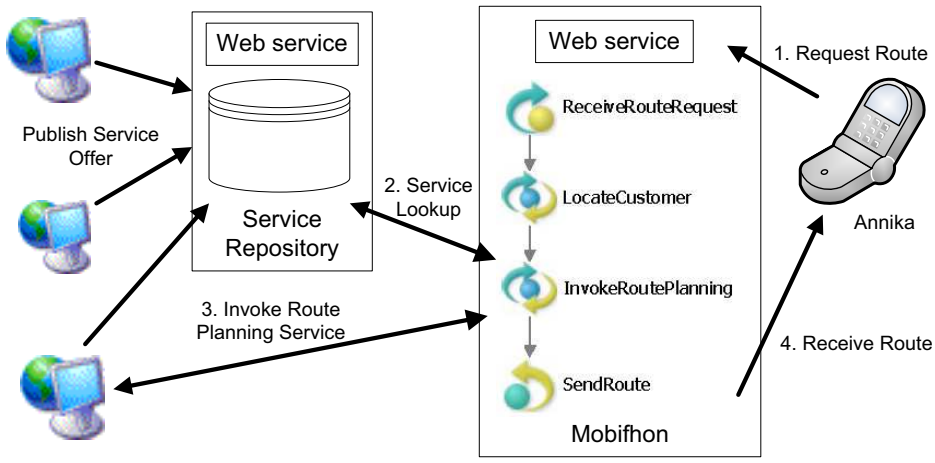

Figure 4.3: Example for mobile service usage [LAGS07].

is charged. Amazon does not provide any formal description of this information beyond a WSDL file for accessing the Web service. Since competitors provide similar grid services such as Sun's One-Dollar-Per-CPU-Hour[3], having formal descriptions would allow requester to dynamically contract the best service currently available.

4.1.3 Mobile Services

In recent years, there is a strong proliferation of mobile devices like mobile phones and PDAs with increasing computational power as well as broadband Internet access. Such devices clear the way for more sophisticated and demanding mobile applications. Usually these applications do not involve solely local computation, but also require information from external services. In general, the type of services required by such mobile application are similar to enterprise or grid services. However, the importance of the applications and the way they are accessed is fundamental differently. In contrast to the previous scenarios, services in a mobile environment are typically less business critical since they are mostly required in a personal context (e.g. calculating a route, agreeing on appointments, receiving sport news), but the environment is more dynamic and the resources, such as bandwidth, are limited. For example, if a mobile phone user travels across a border, automatically, an equivalent service to a previous one has to be discovered in this new context. Of course, such advanced personalization and localization features are additional issues that have to be addressed by the service-oriented architecture.

Example 4.3 *As an example for the Web service usage in a mobile environment, we consider the project SmartWeb.[4] The goal of the project is to enable mobile multi-modal access (e.g. speech input or browsing) to the knowledge available in the Web. This also involves the usage of Web services provided by one of the project partners.[5] The set of available services comprises route planning, weather, event and address services. Obviously, the invocation of these services has to happen in a context sensitive way. This means that service selection*

[3] `http://www.sun.com/service/sungrid/overview.jsp`

[4] SmartWeb is a resarch project financed by the German Federal Ministry of Education and Research (BMBF). More information is available at `http://smartweb.dfki.de`.

[5] `http://services.t-info.de/soap`

	ServiceType	Response Time	Coverage	Indicated Attraction	Traffic Info	Weather Consideration	Price
A	RoutePlanning	10sec.	Ger	Non	no	no	$.2
B	RoutePlanning	12sec.	EU	Attractions	yes	no	$.5
C	RoutePlanning	5sec.	EU	HistoricSites	no	no	$.8
D	RoutePlanning	18sec.	World-Wide	Non	yes	yes	$.5
	…	…	…	…	…	…	…

Table 4.1: Possible route planning service configurations.

could depend on the current location of a mobile phone user, the current time, or the current role of the user, e.g., users acting as car drivers, sports spectators, etc. Thus, dynamic contracting is required in such a scenario to deliver relevant information to the user.

To exemplify this approach, assume the route planning example shown in Figure 4.3. Consider Annika, a mobile phone user, who is currently in the city of Karlsruhe in Germany and wants to know the driving directions to Munich as soon as possible. Annika's mobile network operator, Mobifhon, provides route planning services for several countries to its customers, dynamically outsourced from third party route planning services on the Web, as sketched in Figure 4.3. Thus, the service selection takes place at Mobifhon's end. The service selection is therefore not constrained by the limited resources and partial connectivity of Annika's mobile phone, while allowing Mobifhon to aggregate demands and thus procure better discounts for services than if each customer were to transact individually. Mobifhon only sends the final route to Annika's mobile phone. Mobifhon might implement the process depicted in Figure 4.3, where first the customers' requests are received and the current location of the customer is determined. Based on this context information a suitable service is discovered in a repository. A possible candidate for route planning in Germany is the t-info Route Service.[6] *For other countries other services are available that can be used in case Annika travels abroad, e.g. the Yahoo! Maps Web Services*[7] *or the Google Maps API*[8]*.*

Since often route planning services provide various kinds of routes (the fastest, the shortest etc.) with or without highways, identifying different kinds of attractions on the way and different levels of service quality, the requester has also to decide on a concrete configuration of the service. A more complex route planning, for example, will cost more than a simple route and similarly a quick response will cost more than a slower one. The various route planning services need to be able to describe their capabilities and configurations to Mobifhon, such that it can choose the appropriate one at the desired QoS level. Examples for different configurations of a route planning service are given in Table 4.1. Configurations may differ in the guaranteed 'ResponseTime', the country for which the service can be used ('Coverage'), the attractions that can be indicated along the route ('IndicatedAttractions'), and whether traffic information ('TrafficInfo') or weather information ('WeatherInformation') is considered for calculating the route. Naturally for each configuration a different price is charged depending on the provided functionality and quality.

In order to find out which configuration is most suitable for a given requester the requester's preferences have to be considered. Typically there is a trade-off between price and

[6] `http://services.t-info.de/soap/routeservice/index.jsp`
[7] `http://developer.yahoo.com/`
[8] `http://code.google.com/`

functionality/quality leading to the problem that no absolutely dominant configurations can be identified.

In this section we have introduced the main inter-organizational application scenarios for service-oriented architectures. In order to realize these scenarios the service infrastructure has to provide a set of features to support service contracting and execution. In the next section we discuss these required features in more detail.

4.2 Requirements Analysis

The goal of this work is to develop an infrastructure for the specification of Web service offers, requests and contracts, and for efficient handling these communication primitives in the contracting and settlement process of a Web service market. In line with our research methodology outline in Section 1.2, we first discuss the requirements regarding the market communication language (Section 4.2.1) and regarding the market mechanism (Section 4.2.2). The requirements that can be derived from the scenarios presented in the previous section are listed in the table below:

	Label	Requirement
language-specific	(*R1*)	Web-compliance
	(*R2*)	Multi-attribute Descriptions
	(*R3*)	Combinatorial Requests and Offers
	(*R4*)	Context Dependency
	(*R5*)	Communication Efficiency
	(*R6*)	Interoperability
mechanism-specific	(*R7*)	Automation
	(*R8*)	Flexibility
	(*R9*)	Optimality
	(*R10*)	Computational Tractability
	(*R11*)	Legal Reliability

After introducing these requirements in detail, we summarize the chapter in Section 4.3 by relating the requirements to the scenarios introduced above and discuss how they are addressed in our work.

4.2.1 Language-specific Requirements

(*R1*) Web-compliance

One of the central requirements when considering the adoption of new technologies is the compliance to existing standards. For designing languages usable within

service-oriented architectures this means that communication has to conform to existing Web as well as Web service languages and protocols. Thus, messages should be serialized using XML, which is the basis of most Web languages. In addition, Web service technologies like SOAP and WSDL should be supported. In this context, Web resources should be identified by Uniform Resource Identifiers (URI). Another important feature for languages to be used in a heavily decentralized environments such as the Web is the support of modularization, i.e. the vocabulary can be specified in a decentralized way. In addition, modularization enables reducing the problem of over-committing which arises if people have to agree on an extensive set of terms although a small subset would already be sufficient for a certain purpose.

Although Web-compliance is generally an important topic and a sine qua non for Web-based service-oriented architectures, in many mobile scenarios the requirements are bypassed by introducing an intermediary (such as the network operator in Example 4.3) who spares the mobile device from directly handling service-oriented aspects (like SOAP messages). Such an approach has been widely adopted to reduce the required bandwidth and computational power of the mobile component (e.g. [gri05]). However, the trend towards more powerful devices has also led to a paradigm shift towards pure service-oriented architectures in the mobile domain [KKS06].

(*R2*) Multi-attribute Descriptions

An important way to enable service differentiation are *multi-attribute descriptions of service offers, requests and contracts*. Attributes are inherent to configurable products, i.e. there are usually important aspects beyond just the price. Often these attributes considerably influence the price of a product in the market. As discussed in Section 2.1.2, Web services are products which can be easily differentiated – often even without changing the implementation. In particular, Web services quality of service attributes are an essential part of the service description although they do not address the service functionality itself. Since service differentiation is generally an important instrument for competing in service markets, expressing service configurations is essential in all of our scenarios mentioned above.

(*R3*) Combinatorial Requests and Offers

When considering grid applications such as the one presented in Example 4.2, rarely are single services required by the application, but rather a *combination of services* which exhibit utility only when combined. For example, typically computational power is usable only when bundled with storage capacity. To a lesser extent this is also true for mobile and enterprise service scenarios. However, especially in the mobile scenario usually individual services are acquired that already provide the full required functionality.

In general, when specifying combinations of services, one could define either complements or substitutes. In case of complements, participants have super-additive valuations for the services, as the sum of the valuations for the single services is less than the valuation for the whole bundle of services. In this case *bundles* can be used to ensure that either all services or non is acquired. In case of substitutes, participants have sub-additive valuations and thus the valuation of the bundle

is lower than the valuations of the individual services. In this case offers or requests excluding each other might be required.

(*R4*) Context Dependency

Often customers do not have absolute preferences about the services they need. Their preferences rather depend on their current context, which could include their location, current activities or time, etc. This is particularly true for mobile scenarios, where typically a lot of different context dimensions influence the preferences. Considering Example 4.3, Annika might typically prefer to travel on highways, but she is currently on vacation and wants to travel through scenic country roads, possibly making several stops at attractions on the way. Annika's preferences may also depend on her implicit context. For example, if she has an upcoming appointment in Munich the next day, she is more likely to prefer a short route than a long scenic route. In fact, her context-dependent preferences may be predefined, allowing her to define general policies that are automatically considered by her device. This enables the dynamic application of the right preferences in a certain context. Therefore, we need a way to describe *preferences for particular service configurations declaratively in terms of the customer's context.*

(*R5*) Communication Efficiency

The possibility of customizing services to the customers' needs tremendously increases the complexity of expressing services offers and requests. This arises from the exponential size of the configuration space defined by service attributes and prohibits us from enumerating all possible configurations as proposed by related literature (e.g. [TPP02]). For instance, a service described by five attributes, each with five attribute values, already involves over 3000 configurations. In addition, some attributes might be described on a continues scale which makes enumeration of configurations impossible.

Thus, a critical requirement – especially for resource-constrained environments such as mobile services – is that the chosen representation of information in the Web service market has to be designed for *communication efficiency*. This requires not only that the amount of data that has to be communicated between the market participants is minimized, but also that the amount of data that has to be locally stored by the participants and intermediaries in the market should be minimal. The communication overhead can be determined in terms of the bytes required for expressing offers, requests and contracts and we thus use it as a measure of communication efficiency.

(*R6*) Interoperability

Depending on the market mechanism used, either offers have to be communicated to the buyer, requests to the seller or both to an intermediary. Thus, *interoperability* becomes an important issue. This is particularly crucial in open markets on the Web, where participants may use highly heterogeneous data formats, and participants as well as vocabulary may change frequently. Therefore, a *standardized syntax and semantics* is essential that ensures valid matching in the market although requests

and offers are often specified differently (e.g. using a different level of abstraction or alternative term definitions).

In Example 4.3, a service provider might specify that routes between all cities in Europe are supported, while a customer might look for a route between exactly two cities Karlsruhe and Munich. To bridge these different levels of abstraction, sophisticated *logical inferencing mechanisms* are required which, in this case, utilize the knowledge about cities being in countries and countries belonging to continents. In order to apply such inferencing mechanisms Web service offers and requests have to be described using a formal logic-based language. Different levels of abstractions occur due to the different kind of information available for providers and requests. For instance in Example 4.3, when defining her preferences Annika may not know which attributes are used by the providers to describe their services and even if she did, it would be too tedious to define preferences for all attribute values. She may rather want to say that she generally prefers historical sites to museums without specifying which particular types of each she prefers and by how much. Therefore, in our description approach, *attribute hierarchies* that allow some degree of abstraction have to supported.

The difficulty of guaranteeing interoperability increases with the openness and the heterogeneity of the system. It is thus a major problem for enterprise and grid service markets, since here usually no restriction on openness and heterogeneity is assumed. In mobile scenarios often the network operators have a market power that allows them to ensure a certain homogeneity, e.g., regarding the vocabulary or the language syntax used. Thus, interoperability can be considered a minor problem in mobile scenarios. However, there is also a trend towards more openness in mobile environments, for instance, involving P2P communication between mobile devices where interoperability becomes increasingly important.

4.2.2 Mechanism-specific Requirements

(*R7*) Automation

As already discussed in the scenario descriptions in Section 4.1, *automation of the contracting process* is required in each scenario. Generally, automation refers to decision making without or with minimal human intervention. In the context of contracting, this requires discovering, selecting and closing a contract in an automated fashion.

In addition, the concept of service differentiation may lead to highly diverse Web service contracts. For example, some contracts might guarantee high service quality, whereas others may allow for slow answers and bad quality (e.g. in turn for a cheap price). To cope with this diversity, additional automation within the settlement phase is required: the infrastructure should support *automated verification* whether a service execution meets the obligations of the other party specified in the contract. Due to the high diversity of contracts, manual verification would be very time-consuming or even impossible.

(*R8*) Flexibility

All scenarios defined in Section 4.1 assume an open environment that comes with a high degree of dynamism. For example, as new providers or requesters join the

market, new types of services emerge. This might introduce additional, new vocabulary and could require new forms of market mechanisms. Therefore, we need a *flexible infrastructure* that is easily adaptable to a changing environment. The infrastructure should support changing and adding vocabulary terms and market mechanism functionality (e.g. matching and allocation rules) during runtime of the system. This is essential to provide the required freedom for market participants to compete by offering novel services.

(*R9*) Optimality

The contracting process requires the determination of an allocation between the set of offers and the set of requests. As outlined in Section 2.3 and 3.1.3, an allocation should meet the goal of the market designer. No matter which mechanism is used the goal should be addressed optimally. Possible goals range from mechanisms maximizing the sum of requesters' and/or providers' utility to mechanisms maximizing market volume or mechanisms with the goal of load balancing. From an economic perspective the goal might also include design criteria stemming from the theory of mechanism design [Par01]. However, in a heterogeneous environment, it is not sufficient to provide an optimal allocation algorithm, but one has also to make sure that all relevant offers and requests are considered. For example, when requesting a financial service, credit as well as loan services might provide the desired information and thus both have to be considered in the optimization algorithm. Even an optimal allocation algorithm does not lead to optimal results if only a subset of relevant matches are considered.

(*R10*) Computational Tractability

Although the market mechanism used should be able of dealing with complex (e.g. multi-attribute, combinatorial) offers and request descriptions, the contracting process also has to be computationally tractable, i.e. the time for Web service discovery, determining the allocation and closing the contract, has to be short enough to be applicable in our scenarios. However, the duration of the contracting process is not always crucial: its importance depends on the concrete application scenario. For example, in the case of grid and enterprise services, the contracting is often done at deployment time of the business process, once for several invocations, or once in a certain time period. Thus, in such use cases, enough time for the contracting process can be scheduled. The contracting process is therefore less time-critical here compared to the mobile environment, where services are usually contracted on demand, i.e. at the time they are required. Nevertheless, there are also applications of enterprise and grid services that require individual contracting at runtime.

(*R11*) Legal Reliability

As already discussed in Section 4.1, the *contract closed between providers and requesters must be reliable* in a sense that all duties defined in the contract have to be exercised as promised, or the legal consequences that are caused by an infringement of the contract are applicable. This is especially important for the enterprise and grid scenarios since the failure of crucial applications might lead to a considerable financial loss for the requester.

Requirement	Enterprise Services	Grid Computing	Mobile Computing
Web-compliance	●	●	◐
Multi-attribute Descriptions	●	●	●
Combinatorial Requests/Offers	◐	●	○
Context Dependency	◐	◐	●
Communication Efficiency	◐	◐	●
Interoperability	●	●	◐
Automation	●	●	●
Flexibility	●	●	◐
Optimality	●	●	●
Computational Tractability	◐	◐	●
Legal Reliability	●	●	○

Table 4.2: Relevance of requirements with respect to scenarios.

4.3 Discussion

In this chapter, the environmental analysis of the market engineering process has been conducted. Recapitulating, by analyzing three major scenarios for service-oriented architectures we derived 11 requirements. Table 4.2 summarizes these results by relating the requirements and the scenarios they are derived from. The table entries are interpreted as follows:

● : very important feature for the considered scenario

◐ : feature is not necessarily required, could be helpful in some specific settings

○ : rather unimportant feature for the considered scenario

The table shows that the different scenarios are rather similar with respect to their requirements for the service market infrastructure. This is particularly true for grid and enterprise services, where the only major difference is the importance of combinatorial offers and requests in the grid scenario. More differences can be found between the enterpise/grid and the mobile scenario. These differences are due to three major observations: (*i*) In the mobile scenario the openness can be restricted by the network operators, which may lead to less importance of Web-compliance, interoperability and flexibility. (*ii*) Services in the mobile scenario are combined less often with other services and typically provide personal services which are less business crucial. Thus, combinatorial requests and offers as well as legal reliability are less important than in the enterprise and grid scenario. (*iii*) Another important finding is that computational tractability and communication efficiency is more important in the mobile scenario due to the resource-restrictions in

terms of computational power and bandwidth. However, as outlined above we expect all these differences to vanish or at least to become less important within the next years.

As already discussed in Chapter 3, we realize a Web service market by combining different technologies:

Web service technology: Web service technologies provide us with the basic technology for implementing a service-oriented architecture in an open and heterogeneous environment. By providing standardized protocols and languages, a first step towards interoperability (*R6*) can be achieved, while adhering to existing Web standards (*R1*).

Semantic technologies: This effort is further reinforced by introducing ontologies. Ontologies facilitate interoperability (*R6*) beyond the pure syntactic level by means of standardized logics that provide reasoning capabilities. Reasoning is required in open environments to match requests and offers in a meaningful and complete manner, which is also important to derive optimal results (*R9*).

Policy-based computing: By expressing policies with ontologies, we are able to specify multi-attribute requests and offers (*R2*) in an efficient (*R5*) and context-sensitive (*R4*) way. In addition, the ontology enables expressing combinatorial requests and offers (*R3*) and legal contracts between requesters and providers (*R11*).

Market mechanisms: Based on the vocabulary defined in the ontology a set of market mechanisms are declaratively defined via rules, which leads to a very flexible systems (*R8*) that enables automated contracting of Web services (*R7*). Inspired by optimization techniques from operations research these rules seamlessly integrate efficient optimization techniques for determining the allocation with semantic matching approaches in order to realize computationally tractable mechanisms (*R10*) that lead to optimal results (*R9*).

In Chapter 5, we present the conceptual design of the contracting mechanism. After the conceptual design phase, the embodiment phase of the market engineering process is carried out in Chapter 6 and 7. In this context, the abstract conceptual model is concretized and implemented using an appropriate ontology language. The ontological implementation of the conceptual model finally meets the requirements defined above.

Chapter 5

Abstract Web Service Market Model

In this chapter, we introduce a conceptual market model for trading Web services and thereby implement the conceptual design phase of the market engineering process. The model is formalized using an abstract mathematical notation, which is independent from the concrete representation language used in the market. We thus refer to the model as *abstract model*. The notation we use throughout the chapter is shortly summarized in Table 5.1.

In Section 5.1, we first define a policy model that enables the definition of constraints and preferences over attribute values. In doing so, we concretize the concepts of goal and utility function policies introduced in Section 2.2.3. Subsequently, a policy-based model for specifying multi-attribute, combinatorial requests and offers is presented in Section 5.2. Based on the request and offer specification the contracting process is outlined in Section 5.3. Section 5.3.1 introduces the matching algorithms that are used in Section 5.3.2 for determining the allocation between offers and requests. Finally, a policy-based contract representation is introduced in Section 5.3.3 that enables automated compliance checking of Web service invocations.

This chapter partly assembles results from several publications: A model for goal policies is intuitively introduced in [GLA+04, LEO05]. The motivation and the model for utility function policies is given in [LA05, LASW06, LAO+06]. [LA07] presents first versions of the matching and selection algorithms, which are further developed in [LAGS07]. An informal discussion of the contract compliance monitoring algorithm can be found in [LLM07].

5.1 Policies Specification

As illustrated in the policy classification schema presented in Section 2.2.3, the dimensions for classifying polices are their type, their level of formalization and their level of abstraction. Since automation is needed according to Requirement (*R7*), a high level of formalization is necessary (e.g., for matching offers and requests or for compliance checking of contracts). Further, we want to express abstract business-oriented as well as low-level system-specific policies. The policy model should thus allow the expression of both variants. Concerning the policy types, we are mainly interested in policies that allow us to specify the desired outcome of a decision. Since this is not possible with action policies, we only consider goal and utility function policies in the following.

Notation	Meaning	Example
L	set of attribute identifiers $L = \{l_1, \ldots, l_n\}$	$L = \{$'ResponseTime','Coverage',$\ldots\}$
A	set of attributes with $A = \{A_1, \ldots, A_n\}$	$A = \{$'ResponseTimeValues','Countries',$\ldots\}$
$a_{l_e} \in A_l$	attribute value of A_l with $0 < e \leq \lvert A_l \rvert$	$a_1 =$ '1$sec.$', $a_2 =$ '2$sec.$',etc.
C	set of configurations $C = \{c_1, \ldots, c_{\lvert \prod_{l=1,\ldots,n} A_l \rvert}\}$	$c_1 =$ ('1 sec.','GER',$\ldots$)
$G(\cdot)$	function representing a goal policy	$G(($'1 sec.','GER',$\ldots)) = 1$
Φ	set of constraints $\Phi = \{\phi_1, \ldots, \phi_k\}$	
$\phi \in \Phi$	represents a constraint $\phi = (scp_\phi, rel_\phi)$	$\phi_1 = (($'ResponseTime'$), \{($'1$sec.$'$), ($'2$sec.$'$)\})$
scp_ϕ	scope of a constraint ϕ	$scp_\phi =$ ('ResponseTime','Coverage')
rel_ϕ	tuple of allowed values for $l \in scp_\phi$	$rel_\phi = \{$('1sec.','Ger'),('2sec.','Ger')$\}$
$U(\cdot)$	function representing a utility function policy	$U(($'4 sec.','GER',$\ldots)) = 0.4$
U^S / U^P	represent scoring and pricing policies, respectively	
$r_i \in R$	a request $r_i = (C_i, U_i^S)$ of requester $i \in I$ specifies a set of desired configurations C_i and a scoring policy U_i^S defining the willingness to pay	$r_i = (\{($'1 sec.','GER',$\ldots),\ldots\}$, $\frac{1}{2}(u_1^S(c_1) + u_2^S(c_2)))$
I	set of Web service requester in the market	
$o_j \in O$	an offer $o_j = (C_j, U_j^P)$ of provider $j \in J$ specifies a set of provided configurations C_j and a pricing policy U_j^P defining the reservation price	$o_j = (\{($'4 sec.','EU',$\ldots),\ldots\}$, $\frac{1}{5}(4u_1^P(c_1) + u_2^P(c_2)) + 3)$
J	set of Web service provider in the market	
$t_{ij} \in T$	the trade $t_{ij} = (c, \pi)$ between a provider j and a requester i of a service with configuration c to a price π	$T = \{($'1 sec.','GER',4), ('2 sec.','GER',3),$\ldots\}$
T_i', T_j	trades acceptable for requester i and provider j	
$\delta \in \Delta$	represents a context dimension	$\Delta = \{$'Time','Location',$\ldots\}$
$k \in K$	represents an execution context	$k = ($'12 : 00',$'Karlsruhe',\ldots)$
$b \in B$	the set of bids $B = (C, p)$ with $B = O \cup R$ represents requests or offers, B comprises atomic bids B^A, AND-bids $B^\wedge$, OR-bids $B^\vee$ and XOR-bids $B^\oplus$	
$\mathcal{G}$	set of service types defined via disjoint subsets of configurations $C_1, \ldots, C_m \subseteq C$	$C_1 = \{($'1sec.','France'$), ($'3sec.','France'$)\}$ $C_2 = \{($'1sec.','GER'$), ($'3sec.','GER'$)\}$
$\mathcal{S}$	a bundle of service types $\mathcal{S} \in \wp(\mathcal{G})$	
$\gamma \in \Gamma$	a contract Γ is a set of obligations $\gamma = (y, G_\Phi)$ with $y \in I \cup J$	$\Gamma = \{('CompA', G_{\Phi_1}), ('CompB', G_{\Phi_2})\}$
$\varrho(\cdot)$	An evaluation function $\varrho(\Gamma, c)$ determines whether a configuration c fulfills a contract Γ	

Table 5.1: Summary of notation

5.1.1 Goal Policies

As defined in Section 2.2.3, goal policies define the desired states of a decision maker. In our case states represent concrete *configurations* of Web services and goal policies are used to define whether a certain configuration is admissible for the decision maker. Thus, we can adapt the goal policy definition (as introduced in Section 2.2.3) as follows: a goal policy $G : C \rightarrow \{0,1\}$ assigns a value of 0 or 1 to each configuration $c \in C$, where 0 represents a forbidden configuration and 1 an admissible one. Configurations are described by a set of *attributes identifiers* $L = \{l_1,\ldots,l_n\}$. This set might not only comprise attributes of Web services, but also other attributes relevant to the decision, such as attributes of the service provider. The corresponding domain of the attributes are given by $A = \{A_1,\ldots,A_n\}$. Attribute values a_k of an attribute A_k can be discrete and continuous. The potential configuration space C is defined as the cartesian product $A_1 \times \cdots \times A_n$. A configuration $c \in C$ is a n-ary tuple containing exactly one attribute value of each attribute.

The function G is defined by specifying a set of *constraints* Φ on the attributes that describe the configuration. We denote a policy G that is defined via the constraints Φ by G_Φ. Under the assumption that the a constraint is defined on the first k attributes, let the *scope* of a constraint scp be a k-tuple of attribute labels $(l_1,\ldots,l_k) \in L^+$ (e.g. $l_1 =$ 'ResponseTime' and $l_2 =$ 'Coverage') and the *relation rel* of a constraint the set of k-tuples defining the allowed attribute values $rel \subseteq A_1 \times \cdots \times A_k$ for the scope. Then the k-ary constraint $\phi \in \Phi$ is a tuple (scp_ϕ, rel_ϕ) with scp_ϕ representing the k labels of constrained attributes and rel_ϕ representing the k-tuples of allowed values. A constraint involving one attribute only is called *unary constraint* and a constraint with two attributes *binary constraint*. For example, a 2-ary constraint that restricts the attribute 'ResponseTime' to 1 sec. or 2 sec. and the attribute 'Coverage' to Germany is written as follows: $(('\text{ResponseTime}','\text{Coverage}'),\{('1\text{sec.}','\text{Ger}'),('2\text{sec.}','\text{Ger}')\})$.

Given a k-ary constraint with $scp_\phi = (l_1,\ldots,l_k)$ and $rel_\phi = \{(a^{rel}_{11},\ldots,a^{rel}_{k1}),\ldots,(a^{rel}_{1q},\ldots,a^{rel}_{kq})\}$, and a configuration $c = (a^c_1,\ldots,a^c_n)$, a goal policy can be defined as follows:

$$G_\phi(c) = \begin{cases} 1 & \text{iff } \exists j \in [1,q], \forall i \in [1,k] : \text{match}(a^{rel}_{ij}, a^c_i) = \text{true} \\ 0 & \text{else} \end{cases} \tag{5.1}$$

The Equation 5.1 is evaluated to 1 for a given constraint ϕ and a given configuration c if there is a tuple in the relation rel_ϕ, for which each attribute value a^{rel}_{ij} matches the corresponding attribute value a^c_i in the configuration. The predicate *match* is used to compare two attribute values. In the most simple case, where attribute values represent "flat" datatypes, such as integers or strings, this could be realized by a simple syntactic comparison, e.g. $\text{match}(a^{rel}_{ij}, a^c_i) = true$ iff $a^{rel}_{ij} = a^c_i$. However, as discussed in Section 5.3.1 and in Chapter 7 specifically for our representation mechanism, we do not restrict our approach to simple datatypes. In particular, set based attribute values (e.g. 'Cities in Germany') and hierarchical structures (e.g. the set 'Cities in Germany' is included by 'Cities in Europe') are required in order to provide the interoperability postulated by Requirement (*R6*).

In order to judge a configuration $c \in C$ as admissible, Equation 5.1 has to hold for all constraints $\phi \in \Phi$. This is ensured by the following formula:

$$G_\Phi(c) = \prod_{\phi \in \Phi} G_\phi(c) \tag{5.2}$$

A more detailed discussion about the combination of policies can be found in Section 5.1.3. The constraint specification above is only applicable for a discrete domain. Extending the constraint language to an infinite domain using mathematical operators such as $+, -$, etc., boolean operators like $\wedge$, $\vee$, $=$, etc., and constants is described, e.g., in [LF04, BdVS02].

In this work, goal policies provide the following important features:

- Policies can be used to define guidelines how a service can be configured. In doing so, they enable automation of the selection processes postulated by Requirement (*R7*).
- Goal policies can be used to define indifference classes, i.e. classes of configurations which are equally desired by the decision maker. Since minimal and maximal prices can be defined for the entire class, rather than for each member of the class individually, they avoid enumerating all possible configurations in case of coarse preferences and therefore reduce the communication overhead as required by Requirement (*R5*).

The following example illustrates how goal policies can be used to describe admissible Web service configurations.

Example 5.1 *Consider the route planning service introduced in Example 4.3. In this context, route planning service offers are described by the attributes* $L = \{\text{'Category'}, \text{'ResponseTime'}, \text{'Coverage'}, \text{'Attraction'}, \text{'Price'}\}$. *Assume a provider of a route planning service wants to announce that the service only provides routes in Germany. This can be realized by expressing a goal policy for the attribute 'Coverage'. The domain of the attribute is given by* $A_{Coverage} = \{\text{'Germany'}, \text{'France'}, \text{'UK'}\}$. *In this case an unary constraint* $\phi \in \Phi$ *can be defined with* $scp_\phi = \{\text{'Coverage'}\}$ *and* $rel_\phi = \{\text{'Germany'}\}$. *Consequently, the constraint can be used to define a goal policy* G_ϕ *which enables only configurations that support routes in Germany:*

$$G_\phi(c) = \begin{cases} 1 & \textit{iff } match(\text{'Germany'}, a^c_{Coverage}) \\ 0 & \textit{else} \end{cases}$$

5.1.2 Utility Function Policies

Although goal policies are a first step towards automation and more compact offer and request representation, they are not sufficient in scenarios, where preferences over alternatives are fine-grained, e.g. each alternative might have a different price attached or might be desired to a different degree. In this case, goal policies do not improve representational compactness, since an own indifference class would be required for each alternative. Rather than having policies that classify an alternative as acceptable or not acceptable, we require policies that lead to a degree of satisfaction.

To address this problem, a functional relation between alternatives and their value for the decision maker can be used, which is referred to as *utility function*.[1] Over the last decades, there has been a broad stream of work about modeling utility functions [vNM47, Fis70, KR76, WD92, BG95]. The goal is to provide sufficient expressivity for modeling complex decisions, while keeping the elicitation and computation effort at an admissible level. In our specific case, a utility function is defined as a function $U : C \rightarrow \mathbb{R}$ mapping each configuration to a real-valued measure reflecting the value a decision maker attaches to a certain alternative. The utility is measured on a cardinal scale, which allows making statements about the relative as well as absolute suitability of a configuration. They thus generalize the concept of goal policies by allowing not only two levels 'admissible' and 'not admissible', but make all configurations comparable by introducing a *preference structure* over the configurations.

Definition 5.1 (Preference Structure) *A preference structure is defined by the complete, transitive, and reflexive relation $\succeq$. For example, the configuration $c_1 \in C$ is preferred to $c_2 \in C$ if $c_1 \succeq c_2$. The preference structure can be derived from the utility function $U(c)$ by means of the following condition:*

$$\forall c_a, c_b \in C : c_a \succeq c_b \Leftrightarrow U(c_a) \geq U(c_b) \tag{5.3}$$

Preferences often have an underlying structure which is introduced by the independency of the attributes. Relying on this structure substantially improves their compactness and analytic manipulability [WD92]. The most prominent approach in this context are additive models, where the utility function U is decomposed into several lower-dimensional functions. There are several well known approaches for doing this decomposition based on different structural assumptions. In the following, we shortly introduce the *additive* utility model which has favorable computational properties, but also imposes restrictive assumptions.

Definition 5.2 (Additive Utility Function) *An* additive utility function *is a utility function where an individual utility function $u_l(a_l)$ with $a_l \in A_l$ is defined for each attribute $l \in L$ separately. The overall utility measure U for a configuration c is then calculated by the sum of all individual utility measures. Equation* (5.4) *below exemplifies an additive function. The vector λ is used to define the relative importance of attributes.*

$$U(c) = \sum_{l \in L} \lambda_l u_l(a_l), with \sum_{l \in L} \lambda_l = 1 \tag{5.4}$$

The additive form of utility functions as defined above can only be used if attributes are *mutually preferential independent* [KR76].

Definition 5.3 (Mutual Preferential Independence) *The attributes L are considered mutually preferential independent if every proper subset $X \subset L$ of these attributes is* preferentially independent *of its complement $Y = L \setminus X$. The set of attributes X is preferentially independent from the set Y if and only if, for all assignments $x, x' \in X$ and $y, y' \in Y$ the*

[1] In economic literature, often a distinction between *utility functions* and *value functions* can be found referring to decisions under uncertainty and certainty, respectively. As in related computer science literature [WTKD04, KW04], we abstract from such a distinction in our work and focus on the general concepts suitable for both cases.

following condition holds: $[(x,y') \succeq (x',y')] \Rightarrow [(x,y) \succeq (x',y)]$. *Under this assumption, we can decompose the utility function* $U(c)$ *into the individual functions* $u_l(a_l)$ *of the independent attributes* $l \in L$. *The overall value can be calculated by Equation* (5.4)*, where* $\lambda_l \geq 1$ *represents the weighting factor of an attribute normalized in the range* $[0,1]$.

However, in real markets the preferential independency often does not hold. For example, a decision maker might assign a high utility value to 'Historic Sites' in case 'Coverage' is not valued with her native country. Otherwise with a low utility value, since attractions in her native country might be already well-known and thus indication of attractions is not important. In such a scenario, the attributes 'Coverage' and 'IndicatedAttraction' are not preferential independent. In order to at least partly capture this, dependent attributes $A_j, \ldots, A_k \in A$ can be treated as one single attribute A_l^* in our model, where the utility function is modeled as a complex (higher dimensional) function $u_{l^*}(a_j, \ldots, a_k)$. Since this approach allows one attribute A_j to influence several of the aggregated attributes A_l^*, we support the family of *generalized additive* utility functions [Fis70, BG95]. While the generalized model requires expressing high dimensional functions for the entire service, the additive model requires attaching an one dimensional function to each attribute of the service. Since in general determining utility functions of an agent is rather difficult, we assume extensive methodology and tool support in the preference elicitation process (cf. [CP04]).

In the context of electronic markets, utility functions policies can be used on buyer-side to specify preferences, assess the suitability of trading objects and derive a ranking of trading objects based on these preferences. Since the functional form avoids the enumeration of all configurations in order to attach prices, utility function policies can be used to capture the reservation price of the provider and the maximal willingness to pay of the requester. Therefore, they provide an efficient way of communicating pricing information to the customers and preference information to providers (e.g. in a reverse auction). In the remainder of this work, we denote rules that specify the functional relation between configurations and prices defined by a seller as *pricing policies* (denoted by U^P) and rules that define how much a buyer is willing to pay for a certain configuration as *scoring policies* (denoted by U^S).

Example 5.2 *We take up Example 5.1 and assume a provider who specifies prices for the service by means of the following pricing policy. This function has been suggested in [BK05] for configurable products and augments a base price* u^{base} *with an additional surcharge depending on the configuration chosen by the requester. The surcharge is calculated by an additive function as defined in Equation 5.4. This enables the provider to define price increase on per attribute bases. Often this information can be derived from the provider's internal cost structure.*

$$U(c) = u^{base} + \sum_{l \in L} \lambda_l u_l(a_l) \tag{5.5}$$

Independent of the chosen configuration, the provider charges a base price of $u^{base} = 1.2$. *In addition, there is a configuration dependent price component which is defined via functions for each of the independent attributes* $u_j(a_j)$ *as follows:*

- *Attribute 'Response Time':* $u_1(a_1) = 1 - \frac{1}{10}a_1$ *with* a_1 *measured in seconds.*

- *Attribute 'Attraction': $u_2(a_2)$ is defined by the points P("Historic Sites",1), P("Events",0.5), and P("No Attraction Indication",0).*

Furthermore, we assume weights of $\lambda_1 = 0.5$ and $\lambda_2 = 0.5$. The attributes 'Category' and 'Coverage' are not configurable or do not influence the price and are thus omitted. For example, the price for a service indicating historic sites and guaranteeing a response time of 2 *seconds is specified as* 2.1*.*

Since policies can be specified by many different parties (e.g. different departments in a company), methods for aggregating policies to one consistent decision rule are required. We therefore discuss policy aggregation in the next section.

5.1.3 Policy Aggregation

Since policy-based decision making approaches are usually applied in large-scale applications, typically more than one policy is specified in order to regulate a certain decision. For example, a Web service selection process of a company might be regulated by several scoring policies coming from different departments of the company. The information systems department, for instance, might prefer a highly secure service, while the management might prioritize cheap services. Of course, different scoring policies lead to different valuations as well as rankings and thus to different selections of services. In the remainder of this section, we present a method to derive a coherent decision from such diverse policies. Therefore, policies are first evaluated and the results of this evaluation step are then aggregated.

In traditional policy languages there are two major operators that can be used to combine policies [LEO05, W3C06b]: we can use either a logical *and*-operator in order to define a conjunction of policies (i.e. the aggregated policy is admissible if all contained policies are admissible) or a logical *or*-operator to derive a disjunction of policies (i.e. the aggregated policy is admissible if at least one contained policy is admissible).

However, since our utility function policies result in degrees of satisfaction, this traditional interpretation of the logical operators is not sufficient. In order to define the semantics of the logical operators for multi-valued logics, we borrow ideas from fuzzy logic where the semantics of conjunction and disjunction is defined via *T-norms* and *T-conorms*. In the following, we use the T-norm/T-conorm defined by [Zad65] as follows:

$$\top(a,b) = min(a,b) \text{ for } and\text{-operators} \tag{5.6}$$

$$\bot(a,b) = max(a,b) \text{ for } or\text{-operators} \tag{5.7}$$

Not that the above semantics is suitable for both types of polices, i.e. goal and utility function policies. It ensures that if one of the policies is evaluated to 0, the overall valuation of the conjunction of policies is also 0. This guarantees that if one policy in a conjunction is violated, the entire conjunction is violated. In case of disjunctions only one policy has to be fulfilled and thus we take the maximal valuation, which is always 1 in case of goal policies.

In the following, we show how policies can be used within electronic markets to specify requests, offers and contracts.

5.2 Communication Primitives

In this section, we formally define the notion of Web services offers, Web service requests and Web service contracts. For our work, we adapt general bidding languages for multi-attribute products developed in [EWL06, BK05]. This requires specializing the language to the Web service domain and to incorporate the policies defined above. In doing this, we simplify the models in some areas while extending them in others. We start with formally defining our notion of a Web service and then specify how Web service offers, requests and contracts can be expressed.

5.2.1 Generic Web Service Specification

In order to be independent of concrete service description approaches, we take a fairly abstract view of a *Web service* in our model and consider it to be fully described by the attributes $A_1, \ldots, A_n$. Such properties might comprise service input and output, behavioural aspects of a service, QoS attributes, etc., thus covering existing Web service description approaches as well as fulfilling Requirement (*R2*). Such a general description of a Web service allows us to abstract from various existing Web service description frameworks, such as WSDL, OWL-S, SAWSDL, WSMO, while simultaneously allowing us to utilize existing decision-theoretic algorithms for multi-attribute products.

As defined above, the set $C = A_1 \times \cdots \times A_n$ of Web service configurations comprises all possible combinations of attribute values. For example, considering the attributes *Attractions*, *Highways* and *Response Time* of a route planning service, a concrete configuration would be a service that provides routes including highways and information about nearby attractions within 10 seconds.

A Web service *trade* t_{ij} is defined as a tuple (c, π), where agent j provides a Web service with configuration $c \in C$ to a customer i at a price of $\pi \in \mathbb{R}$. Furthermore, let T_j denote the set of all contracts involving provider j, and T_i the set of contracts involving customer i. Not all possible contracts are acceptable by an agent, and thus, only subsets $T_j' \subseteq T_j$ and $T_i' \subseteq T_i$ are offered or requested, respectively.

Based on these definitions, we discuss in the next section how utility function policies can be used to define offers and requests in terms of suitability of trades.

5.2.2 Bid Specification

A bid is a communication primitive that allows market participants to convey their offers or requests to the market mechanism. As discussed in Requirement (*R2*) and (*R3*), bids within a Web service market have to cover multiple attributes as well as combinations of different service types. Moreover, bids may depend on the context of an issuer (Requirement (*R4*)). These issues are addressed in the following:

Multi-attribute Bids

A major problem in designing bidding languages for multi-attribute products is the combinatorial explosion that results from adding price markups to each configuration. For instance in Example 4.3, for each configuration a row in the table is required which already leads to thousands of rows for relatively small scenarios.

Assume, for example, a scenario with 5 attributes that each have 5 attribute values. This leads already to 3125 possible configurations and increases exponentially with the number of attributes. A technique to efficiently encode preference and pricing information (Requirement (*R6*)) is the use of utility function policies that represent the relationship between Web service configurations and their prices. In the following definition pricing policies are used to define the acceptable price π implicitly in a functional form.

Definition 5.4 (Web Service Offer) *An* offer *by a provider j is defined as a pair $o_j = (C_j, U_j^P)$ of a set $C_j \subseteq C$ of configurations and a pricing policy $U_j^P : C_j \rightarrow \mathbb{R}$ mapping each configuration $c \in C_j$ to a real number that represents the price π of invoking service configuration c. The set of all offers in the market is denoted by $O = \{o_j | j \in J\}$. Due to* payment monotonicity *[EWL06], i.e. $\forall \pi < \pi' : (c,\pi) \in T_j' \Rightarrow (c,\pi') \in T_j'$, we interpret $U_j^P(c)$ as the minimal price for which a provider is willing to accept a trade with $T_j' = \{(c,\pi) \in T_j | \pi \geq U_j^P(c)\}$. As suggested by [BK05], the pricing function $U_j^P(c)$ can be described by a base price p_j^{base} and an additive function that aggregates pricing functions for individual attributes:*

$$U_j^P(c) = p_j^{base} + \sum_{l \in L} \lambda_{jl} u_{jl}^P(a_l) \text{ with } \sum_{l \in L} \lambda_{jl} = 1 \tag{5.8}$$

where u_{jl}^P represents the pricing function of provider j for a particular attribute identified by $l \in L$. The weights λ_{jl} are used to adjust the influence of different attributes on the price.

Thus, an offer assigns an additive pricing function to a Web service description, mapping the configurations of the offer to a certain price. In Definition 5.4, an additive function is used for the specification of the pricing policy. While additivity improves compactness of the representation and analytic manipulation (e.g. implementing an efficient market mechanism), it also constrains the pricing models available for the provider.

Analogously, we introduce a functional form for representing Web service requests. One major difference though is that a requester's willingness to pay might depend on a runtime specific context (*R4*).[2] Therefore, we introduce a set $K = \delta_1 \times \cdots \times \delta_m$ of *execution contexts*, where δ_l represents different context dimensions, such as current location of a mobile device, time of service execution, history of past transactions. Any $k \in K$ denotes a concrete execution context.

Definition 5.5 (Web Service Request) *A Web service request by requester i is defined as a pair $r_i = (C_i, U_i^S)$ of a set $C_i \subseteq C$ of acceptable configurations and a scoring policy $U_i^S : C_i \times K \rightarrow \mathbb{R}$ that maps each configuration to a real number score depending on the execution context k. The set of all requests in the market is denoted by $R = \{r_j | i \in I\}$. Due to* payment monotonicity, *i.e. $\forall \pi > \pi' : (c,\pi) \in T_i' \Rightarrow (c,\pi') \in T_i'$, we interpret $U_i^S(c,k)$ as the maximal price for which a customer is willing to carry out the trade, i.e. $T_i' = \{(c,\pi) \in T_i | \pi \leq U_i^S(c,k)\}$. U_i^S is an additive scoring function composed of the attribute-specific*

[2] In general, also context-dependent reservation prices in an offer are possible, but rarely used in practice.

functions u_{il}^S and their relative weights λ_{il}:

$$U_i^S(c,k) = \begin{cases} \sum_{l \in L} \lambda_{il} u_{il}^S(a_l,k) & \text{if } c \in C_i, \\ -\infty & \text{otherwise.} \end{cases} \quad \text{with } \sum_{l \in L} \lambda_{il} = 1 \tag{5.9}$$

A configuration which is not requested is scored as minus infinity.

As discussed in the previous section, due to the additive form of the scoring function U_i^S, we have to assume mutual preferential independency [KR76] between the attributes in the scoring function. This holds if the utility of an attribute A_l does not depend on the value of another attribute. For example, the score for a certain guaranteed response time will not change if the type of indicated attractions changes. Technically, dependent attributes can be implemented by using higher-dimensional functions [LAO+06]. However, the added value of this is questionable, given that it is also much harder for requesters to specify preferences with inter-dependent attributes. Context dimensions can be seen as preferential dependent attributes in a sense that they directly influence the preferences for the other attributes and are represented by an additional dimension in the function. However, we assume a rather modest increase in complexity for the requester by introducing contexts, since most scenarios have a relatively small number of contexts k and most context changes do not influence the preferences.

Combinatorial Bids

In order to model requests and offers that exhibit a complex structure with respect to complementarity and substitutability (Requirement (*R3*)), we introduce primitives for modeling *OR/XOR-formulae* [Nis00] that can be used to combine bids. In order to do so, we first have to introduce the notion of *AND-bids*, *OR-bids* and *XOR-bids*. Generally, a bid $b \in B$ with $B = O \cup R$ represents either a request or offer. For simplicity, we use a generic notation $b = (C,p)$, where we assume that all possible configurations C are provided/requested and where it is not distinguished between offers and request. The variable p represents the maximal (e.g. $p = U^S(c)$) or minimal price (e.g. $p = U^P(c)$) for requests and offers, respectively. The set of atomic bids (C,p) is denoted by B^A.

Combinatorial bids allow bidding on combinations of products, i.e. in our case on combinations of Web services. We thus distinguish between different types of Web services according to the configurations they provide. Let $\mathcal{G} = \{C_1,\ldots,C_{|\mathcal{G}|}\}$ be the set of service types containing pair-wise disjoint set of configurations, viz. $\forall X,Y \in \mathcal{G} : X \cap Y = \emptyset$. This rather general approach avoids explicitly introducing different types of products/items into the model as it is usually the case in related literature. We belief that especially when dealing with services it is often unclear whether we talk about different service types or about different configurations of the same type. For instance, consider our route planning service scenario. One might view a route planning service for Germany and France as two different service types where others might view that as different configurations of a route planning service. Therefore, our modeling approach enables defining service types based on configurations. By explicitly introducing an attribute 'ServiceType' and adding a separate set of configurations $C \in \mathcal{G}$ for each attribute value of 'Service Type', our approach directly maps to the standard formulation of combinatorial bidding languages in literature

(e.g. compare [BH01, San02, dVV03]). A service type is thus defined via a set of viable configurations.

Given the set of service types $\mathcal{G}$ we introduce the notion of AND-bids (aka *bundles*).

Definition 5.6 (AND-Bid/Bundle) *AND-bids represent items that are required together and are therefore considered as complements. AND-bids are expressed via a pair $b^{\wedge} = (\mathcal{S}, p)$, where $\mathcal{S} \subseteq \mathcal{G}$ and p is the overall price for all required or provided service types $C \in \mathcal{S}$. The price can be defined explicitly or as a function of the contained configuration as outlined above. Let $T_q = \{(c, \pi) \in T | c \in C_q\}$ with $C_q \in \{C_1, \ldots, C_{|\mathcal{S}|}\}$. Then the set of acceptable trades for a requester i and provider j is given as $T'_i = \{(t_1, \ldots, t_{|\mathcal{S}|}) \in T_1 \times \cdots \times T_{|\mathcal{S}|} | \pi_1 + \cdots + \pi_{|\mathcal{S}|} \leq p\}$ and $T'_j = \{(t_1, \ldots, t_{|\mathcal{S}|}) \in T_1 \times \cdots \times T_{|\mathcal{S}|} | \pi_1 + \cdots + \pi_{|\mathcal{S}|} \geq p\}$, respectively. The set of AND-bids $b^{\wedge}$ is denoted by $B^{\wedge}$ with $B^{\wedge} \subseteq B$.*

Based on these definitions, OR/XOR-formulae can be specified, which enable the expression of arbitrary combinations of OR- and XOR-bids and thus enable the specification of complementarity as well as substitutability. An OR-bid represents a combination of bids, where a valid allocation has to fulfill any number of the contained bids for a price equal to the sum of the individual prices of the contained bids [Nis00]. In particular, it is possible to get all services contained in the bid or no service at all.

Definition 5.7 (OR-Bid) *Let $b^{\vee} = (b_1 \vee \cdots \vee b_m, p)$ represent an OR-Bid with $b_k \in B$ referring to an arbitrary bid and with $b^{\vee} \in B^{\vee}$. The semantics of the disjunction $\vee$ is defined as a set of separate, not-related bids $\{b_1, \ldots, b_m\}$ and the overall price as the sum of the individual bid prices $p = p_1 + \cdots + p_m$. Thus, the acceptable trades are given by $T'_1 \times \cdots \times T'_m$.*[3]

This means that OR-bids can only represent valuation without substitutabilities [Nis00, Proposition 3.1], e.g. it is not expressible that either a HD-storage service or CD-storage service are required, but not both together. This can be done with XOR-bids. We define XOR-bids as follows:

Definition 5.8 (XOR-Bid) *Let $b^{\oplus} = (b_1 \oplus \cdots \oplus b_m, p)$ represent an XOR-bid with arbitrary bids $b_k \in B$ and with $b^{\oplus} \in B^{\oplus}$. The connective $\oplus$ makes sure that exactly one of the specified service types is allocated to the bid, i.e. exactly one of the contained bids $b_1, \ldots, b_m$ is fulfilled. The set of acceptable trades is given by $T' = \{t \in T | t \in T'_1 \vee \cdots \vee t \in T'_n\}$, where T'_q with $q = 1, \ldots, m$ is the set of acceptable trades for an atomic bid $b_q \in B^A$.*

XOR-bids can represent all possible valuations [Nis00, Proposition 3.2]; however, not always in an efficient manner. Additive valuations of m service types require a XOR-bid of size 2^m (OR-bids require only a size of m in this case).[4] Therefore, Definition 5.7 and 5.8 are recursively defined. The recursive nature of the OR- and XOR-bid definition enables us to express arbitrary OR/XOR-formulae and thus to represent XOR-of-OR-bids and OR-of-XOR-bids [San99]. As shown in [Nis00],

[3] Note that for each set $T'_1, \ldots, T'_m$ we have to add an *null*-element referring to 'No Trade', which indicates that no trade to a price of zero is also acceptable. This is required since fulfilling an OR-bid does not necessarily mean all service types are required.

[4] The size of a bid is the number of atomic or AND-bids contained in the OR/XOR-bid.

OR/XOR-formulae are suitable for all kinds of combinatorial valuations and additionally can be transformed to the bidding language $\mathcal{L}_B^{OR^*}$ [FBS99]. It can be shown that this language allows an efficient representation of all kinds of combinatorial bids [Nis00]. Merits and problems of the different formulations – in particular with respect to the family of $\mathcal{L}_G$ languages – are discussed in Chapter 10.

The overall set of bids that can be specified in our Web service market is given as $B = B^A \cup B^\wedge \cup B^\vee \cup B^\oplus$.

After an allocation between offers and requests has been determined, an automatically enforceable and legally reliable contract has to be generated. Therefore, in the next section a contract model is presented that relies on goal policies for the specification of the contractual obligations.

5.2.3 Contract Specification

In service-oriented architectures, applications are assembled as required by pulling together various services offered by different service providers. So as to make sure that a service meets the requirements, customers and providers have to agree on terms of a contract. According to the market phases introduced in Section 2.3.1, these terms are determined in the agreement phase and unambiguously specified in a contract. In the later settlement phase, this contract represents normative relations between the contracting parties that specify, e.g., what must be done, should be done or can be done. There is a vast amount of literature on different normative relations that can be used in this context. The starting point for most work in this area is the seminal work of Hohfeld [Hoh13], which introduces the legal relations *duty*, *right*, *power* and *liability*. A formal, more fine-grained analysis of Kangar [Kan72] already yield 26 different "normative positions" and a further development of the theory by Lindahl [Lin77] 35 different positions.[5] However, within the Web service settlement phase we are only interested whether the obligations in the contract are met by the Web service execution. A Web service execution can be seen as the execution of a concrete configuration c from the set of possible configurations C. Obligations in our Web service model are constraints on Web service attributes that classify the set of configurations C into acceptable and not acceptable ones. We can thus use goal policies to specify obligations as captured by the following definition.

Definition 5.9 (Web Service Contract) *A Web service contract Γ is a bilateral agreement specifying a set of obligations γ. Obligations are represented as goal policies that have to be met by a contracting party. An obligation γ is a tuple (y, G_Φ) specifying the contracting party that is obliged to execute a certain task ($y \in J \cup I$) and one or several goal policies G_Φ mapping each possible alternative to the set $\{0,1\}$. Moreover, let $\varrho(\Gamma, c)$ be a function that evaluates a contract $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ with respect to an observed configuration. $\varrho(\Gamma, c)$ is defined as follows:*

(5.10)

$$\varrho(\Gamma, c) = \begin{cases} 1 & \text{iff } G_{\Phi_1}(c) \wedge \cdots \wedge G_{\Phi_q}(c) = 1, \text{ with } \gamma_1 = (y, G_{\Phi_1}), \ldots, \gamma_q = (y, G_{\Phi_q}) \\ 0 & \text{else.} \end{cases}$$

[5] For a good overview of normative positions the interested reader is referred to [Ser01]. Moreover, some relevant approaches based on deontic logic (i.e. the logic of permissions and obligations) are presented in Chapter 10.

The semantics of a contract conjunction is given by $\varrho(\Gamma_1 \wedge \cdots \wedge \Gamma_q, c) = \varrho(\Gamma_1, c) \wedge \cdots \wedge \varrho(\Gamma_q, c)$ *and of a contract disjunction by* $\varrho(\Gamma_1, \vee \cdots \vee \Gamma_q, c) = \varrho(\Gamma_1, c) \vee \cdots \vee \varrho(\Gamma_q, c)$.

According to this definition, goal polices are used to define whether or not a certain configuration is admissible. This is, of course, a very restricted view on legal contracts which is specific to our Web service model, since it requires that all essential properties of a service are described by attributes.

Example 5.3 *Consider a contract* $\Gamma = \{\gamma_1, \gamma_2\}$ *specifying that a Web service provider j has to grant access to a route planning service which returns a route within 2 seconds. The route has to contain information about historic sites along the way and route calculations should be possible for starting points and destinations within Germany. This can be captured by the provider obligation* $\gamma_1 = (j, G_{\Phi_1})$, *where* Φ_1 *contains a 3-ary constraint* $\phi_1 = (('ResponseTime', 'Attraction', 'Coverage'), (\{('1sec.', 'HistoricSite', 'Ger'), ('2sec.', 'HistoricSite', 'Ger')\}))$. *A second obligation* $\gamma_2 = (i, G_{\Phi_2})$ *captures the duty of the requester i that requires paying the agreed amount of money for the service.* Φ_2 *contains the unary constraint* $\phi_2 = (('Amount'), \{('3')\})$.

Although this approach does not allow a full specification of all possible contracts, it is sufficient to capture the information relevant for Web service contracting and monitoring. This issue is further discussed in Section 6.4.

5.3 Web service Contracting and Contract Monitoring

Having defined the communication primitives available in the market, the conceptual design of the market process is specified. Our focus in this context is the contracting and monitoring phase. As discussed in Section 3.2.2, the contracting process comprises three main phases: the matching phase is discussed in Section 5.3.1, the allocation phase in Section 5.3.2 and the contract formation phase in Section 5.3.3. In the latter, we also discuss the monitoring of Web service contracts with respect to a concrete service invocation, which can also be considered as a part of the settlement phase.

5.3.1 Matching of Bids

Depending on the market mechanism used (e.g. hit-and-take-mechanisms, negotiations or auctions), offers and requests have to be compared with respect to their suitability for the other parties in the market. In case of configurable services, this involves matching of all attributes described in the offer as well as in the request in order to make sure that at least one provided configuration is suitable for the requester. Generally, an offer matches a request if the intersection of the set of provided trades T_j' and the set of requested trades T_i' is not empty, i.e. $T_j' \cap T_i' \neq \oslash$. Given the policy-based definition of multi-attribute offers (Definition 5.4) and requests (Definition 5.5), we can phrase this condition as follows:

$$T_j' \cap T_i' = \{(c, \pi) \in T_j \cap T_i | U_j^P(c) \leq \pi \leq U_i^S(c, k)\} \tag{5.11}$$

This means we can determine if there is a match between an offer and request by evaluating if the condition $U_i^S(c,k) - U_j^P(c) \geq 0$ holds for at least one configuration

$c \in C_i \cap C_j$. However, in many situations it is not sufficient to know if there is a match, but also which of the trades $t \in T'_j \cap T'_i$ is the best match, i.e. the trade maximizing the surplus u that can be realized. Finding this best match is denoted by *Multi-attribute Matching Problem* (MMP) [EWL06]. Assuming no knowledge about the providers internal cost function and therefore indifference between the different configurations on provider side, we define the surplus simply as the difference between score and price $U_i^S(c,k) - U_j^P(c)$. In this case the MMP is defined as follows:

Definition 5.10 (MMP) *Given an atomic request $r_i = (C_i, U_i^S)$, an atomic offer $o_j = (C_j, U_j^P)$ and an execution context k, we solve the MMP by maximizing the requester's utility per service configuration. The solution of the MMP for a given request $r_i \in B^A$ and provider $o_j \in B^A$ is referred to as $u_{ij}(r_i, o_j)$ and is defined as follows:*

$$u_{ij}(r_i, o_j) = \max_{c \in C_i \cap C_j} U_i^S(c,k) - U_j^P(c) \tag{5.12}$$

When assuming additive pricing and scoring functions as done in Definition 5.4 and 5.5, we can considerably simplify the MMP. In particular, we can decompose the calculation into individual subproblems which can be solved independently. The following formulae (5.13-5.15) use this property to solve Formula 5.12 efficiently by reducing the the search space from $O(|\prod_l A_l|)$ to $O(\sum_l |A_l|)$. The binary decision variable x_{le} is associated with each attribute value and denotes whether the value is part of the best configuration. Equation 5.14 ensures that exactly one attribute value is selected for each attribute. Since the following integer programming formulation has a totally unimodular constraint matrix and only integers on the constraints' right-hand sides, the problem can be solved efficiently using the simplex algorithm [PS82].

$$\max \quad \sum_{l=1}^{n} \sum_{e=1}^{|A_l|} (w_{il} f_i(a_{le}, k) - w_{jl} p_j(a_{le})) x_{le} - p_j^{base} \tag{5.13}$$

$$s.t. \quad \sum_{e=1}^{|A_l|} x_{le} = 1 \quad for\ 0 < l \leq n, \tag{5.14}$$

$$x_{le} \in \{0,1\} \quad for\ \ 0 < l \leq n,\ \ 0 < e \leq |A_l| \tag{5.15}$$

The MMP formulation introduced above efficiently determines the optimal configuration given an atomic request and offer. Combinatorial bids are not addressed, since they become not relevant until several requests and offers are considered, which is discussed in the next section.

5.3.2 Web Service Allocation

As introduced in Section 2.3.2, determining the allocation between offers and requests in a market can be done by means of a wide range of different allocation mechanisms. In this section, we consider the problem of designing a suitable mechanism for Web service markets. A general problem in this context is the trade-off between the expressivity of the bidding language that can be handled by a mechanism and the computational tractability of determining the allocation.

As the requirement analysis in Section 4.2 has shown, an allocation mechanism suitable for *all* different scenarios has to support multi-attribute, combinatorial bids. Moreover, especially in the grid scenario, we usually have serious resource restrictions at provider side that should be handled by the mechanism. To address this resource allocation problem market mechanisms with dynamic pricing – in particular different types of auctions – have been suggested in literature [WPBB01, BAGS02, LZR03, SNVW06]. While using such mechanisms enables realizing efficient allocations from an economical point of view (Requirement (*R9*)), they are often computationally very demanding and thus are not suitable for all settings (e.g. scenarios that require selecting the most suitable services from hundreds of Web service offers).

In this section, our intention is therefore not to propose an allocation mechanism generally applicable for all kinds of Web service scenarios, but rather to show how the MMP formulation introduced in Section 5.3.1 can be applied in different allocation algorithms. In doing this, we seamlessly realize semantic matching of attribute values. In the remainder of this section, we first introduce a lightweight allocation algorithm that selects the most suitable service provider and the best available Web service configuration for a given request. This mechanism is referred to as *Web service selection* and addresses the requirements in an enterprise and mobile service setting, where we usually do not have any resource limitations or the limitations can be easily handled by the provider. As a second mechanism, we present an auction-based approach featuring dynamic pricing based on supply and demand. Particularly in the grid scenario this mechanism leads to economically more efficient allocations.

Web Service Selection

In general, a *selection* is defined as a decision for the *best* available alternative. That means in our case the goal is to find the Web service that is most appropriate to fulfill a certain goal specified by the requester. Web services selection corresponds to a Hit-and-Take-mechanism, where a requester selects an offer according to her preferences. It provides the basic functionality for determining suitable Web service bindings (see Section 2.1.2). For flexible binding at runtime the Web service selection has to be done automatically by the system.

Finding the best service involves the decision for a certain provider as well as for a certain configuration of the Web service. Therefore, we have to solve two maximization problems: First, the best contract for a given provider has to be identified which corresponds to the Multi Attribute Matching Problem (MMP) defined in Section 5.3.1. Based on the solution of the MMP, we can determine the best provider by solving *Local Selection Problem* (LSP). This is to find the best provider for a given request from the set of all offers. Obviously this implies that the best contract for each provider is known, i.e. the *MMP* is solved for each pair of request and offer. For formulating the LSP, we assume that all requesters get the service they want if they fulfill the providers' policies (e.g. pay the required price). This assumption is realistic especially for information services such as a route planning service, because of their low operational demands with respect to the required resources. Since each requester gets the chosen service, no combinatorial bids are needed in such a scenario. The LSP is formally defined as follows:

Definition 5.11 (LSP) *Given a single atomic request r and a set of atomic offers O, the Local Selection Problem can be solved by iterating over all offers $o_j \in O$ and determining the offer which leads to a maximal solution for MMP. We thus have to solve the following optimization problem:*

$$\max_{j=1,\ldots,|O|} u_{ij}(r_i, o_j) \tag{5.16}$$

Given the solution of all possible MMPs, solving LSP is linear with respect to the number of offers and requires $O(|O|)$ steps, since we have to calculate the surplus exactly once for each offer. However, as already mentioned the LSP formulation above is based on several simplifying assumption and thus there are several scenarios where LSP is not sufficient for service selection:

- First, the problem can be relaxed to the multi-unit case where a request might require more than one invocation of a service. This might be the case if the binding of services is done once at deployment time of the business process. If we further do not assume quasi-linearity of the utility functions, e.g. by allowing one time costs, the problem will get considerably more complex. It can be shown by reduction of the *Uncapacitated Facility Location Problem* that computing the optimal service in such scenarios is in $\mathbf{FP}^{\mathbf{NP}}$ [BF05].

- Second, the LSP is formulated for the selection of a single service required in a business process. However, often several services might be required within such a process. These services might depend on each other in a sense that not all are compatible to each other (and thus cannot be used within one process instance) or that service prices change if several services are bought from one provider. When the LSP is generalized to an entire service compositions, the problem is called *Global Selection Problem* (GSP). The GSP is addressed in [ZBN+04, SBM+04, AVMM04, YL05, JMG05]. The goal of GSP is to optimize attributes for the entire business process of a requester, such as the overall runtime of the process, the overall costs, etc.

- Third, for the problem formulation of LSP we have assumed that offered services are always available for all requesters and that possible resource limitations are handled at the provider side, e.g. by adapting the guaranteed service levels or by increasing server capacity. Of course, handling resource limitation at provider side is not always possible (especially in grid or utility computing scenarios). Therefore, the allocation mechanism has to handle the problem. In case of LSP this can be done simply by applying the first-come-first-serve principle. However, this may lead to economically inefficient outcomes, since a requester with the highest valuation of a service might not get the service. More efficient solutions can be realized by means of dynamic pricing mechanisms such as auctions, as done in [SNVW06] for computational grids, or by sophisticated scheduling algorithms as done in [BK06b].

As the Local Allocation Problem introduced in this section is mainly applicable to the enterprise and mobile scenarios where the assumption of no resource limitation is usually valid, in the next section we address the grid services scenario by relaxing this assumption.

Web Service Auction

Each provider of a grid system has only limited resources that can be offered to customers. For example, a provider of a storage service is restricted by the capacity of its hard discs and a computing service provider by the power of the available processors. In this context, the providers also have to make sure that the quality of service guaranteed to the customers is maintained. Therefore, it is sensible for service providers to restrict access to their resources as the maximal load is approached. This leads to the problem that not always the requester who value a certain service most can invoke this service, since the maximal load might be already reached. In such situations, the overall surplus realized by the system is not optimal and thus the allocation is not efficient in an economic sense. This obviously contradicts Requirement *R9*.

Introducing an *auction mechanism* where prices are dynamically determined by the interplay of supply and demand could improve the situation. Consider a simple so-called English auction, where an auctioneer announces a Web service offer on behalf of the provider and the requesters interested in using the service submit their bids iteratively in an open-cry manner. In this case, the requester with the highest bid wins the auction and acquires the right to invoke the services. In subsequent auctions, the service can be allocated to the requester with the second, third, fourth, etc. highest bid. Thus, the allocation realizes a higher degree of efficiency compared to the Web service selection mechanism, where a first-come-first-serve-principle is applied.

A major problem with traditional single-item auctions as the English auction described above is that the outcome depends on the behavior of other market participants and is therefore unknown. For example, a requester that needs two Web services for her business process might end up with only one service and a process which is not executable. This is an example for services complementing each other. As already discussed, bids on complements can be expressed via AND-bids. In order to minimize the risk of a requester to end up without a required service, parallel bidding on the same service might be necessary. This introduces the risk of purchasing both services although only one is required. Such sub-additive valuations refer to substitutability, which can be expressed via XOR-bids. It is therefore easy to see that a combinatorial auction that supports AND- and XOR-bids further increases the efficiency for the market participants [dVV03]. In addition, combinatorial auctions provide lower transaction costs and higher transparency [Sch05].

In order to illustrate how we can use the MMP formulation defined in Section 5.3.1 within an auction, we consider a simple combinatorial auction that allows us to allocate a set of AND-bids containing service types $\mathcal{S} \subseteq \mathcal{G}$ to a set of requesters $i \in I$. Before defining the corresponding optimization problem, we generalize the MMP to handle arbitrary AND-bids of the form $b = (\mathcal{S}, p)$ with $b \in B^{\wedge}$. In doing so, the price p_i a requester i is willing to pay for a service can be determined as a function $p_i(\mathcal{S}) = p(U_i^S(c_1^*), \ldots, U_i^S(c_{|\mathcal{S}|}^*))$, where c_l^* represents the utility maximizing configuration of a service type $C_l \in \mathcal{S}$. Based on this definition, the *Combinatorial Auction Problem* [RPH98] can be formulated as an integer program (Equation 5.17-5.20), where $x_i(\mathcal{S})$ represents a binary decision variable indicating whether a bundle $\mathcal{S}$ is allocated to requester i. Equations 5.18 ensures that no overlapping sets of

service types are allocated to a requester. Constraint 5.19 ensure that no requester gets more than one subset of $\mathcal{G}$.

$$\max \quad \sum_{i \in I} \sum_{\mathcal{S} \subseteq \mathcal{G}} x_i(\mathcal{S}) p_i(\mathcal{S}) \tag{5.17}$$

$$s.t. \quad \sum_{\mathcal{S} \ni l} \sum_{i \in I} x_i(\mathcal{S}) \leq 1 \; \forall l \in \mathcal{G} \tag{5.18}$$

$$\sum_{\mathcal{S} \subseteq \mathcal{G}} x_i(\mathcal{S}) \leq 1 \; \forall i \in I \tag{5.19}$$

$$x_i(\mathcal{S}) \in \{0,1\} \; \forall \mathcal{S} \subseteq \mathcal{G}, i \in I \tag{5.20}$$

However, the conceptual advantage of the combinatorial auctions compared to traditional auctions is partially offset by the computational hardness of the algorithms for determining winners and prices. Most instances of integer programm formulation for the winner determination problems above are not efficiently solvable, since the problem is reducible to the set packing problem [RPH98], which is shown to be NP-complete [Kar72]. In literature, a vast amount of (exact as well as approximate) algorithms have been suggested to solve the winner determination problem in combinatorial mechanisms under wide range of different assumptions and in different settings. A detailed discussion of this problem can be found in [RPH98, FBS99, Nis00, ATY00, BH01, San02, dVV03].

Although computationally very demanding, the combinatorial auction model presented above features only basic functionality which is not sufficient in many scenarios. There are approaches to extend the model in several directions. For example, MACE [SNVW06] introduces competition on both sides by providing a double-sided mechanism, where requesters as well as providers simultaneously submit bids to the auctioneer. In addition, XOR-bids $B^{\oplus}$, co-allocation constraints and time-issues can be handled. Other extensions allow exchanging multiple-units of a service as suggested by [dVV03, BK05]. This is especially important if service bindings are determined at deployment time of a process for several executions at once.

While these extensions further improve market efficiency, such additional feature often increase the complexity of the auction and limit the computational tractability (Requirement *R10*). Thus, complex combinatorial auctions are typically not applied for runtime-selection of services. However, they can be used for small scenarios with a moderate number of providers or for selecting services at development or deployment time.

5.3.3 Contract Formation and Monitoring

After allocating offers to requests a contract is needed that unambiguously specifies the involved parties, agreed service levels and prices. As introduced in Section 5.2.3, a contract Γ is formally defined as a set of obligations $\gamma = (y, G_\Phi)$, where G_Φ represents a goal policy with constraints Φ and y the party obliged to meet the constraints. In the allocation phase, the set of admissible trades is determined by the intersection of the acceptable sets of requester i and provider j: $T_i' \cap T_j'$. The obligations in the contract have to identify exactly these acceptable trades $t = (c, \pi)$ with

$t \in T_i' \cap T_j'$. Thus, each trade t defines a contract with a *provider obligation* γ_j and a customer obligation γ_i as illustrated in Example 5.3 on page 89. In case several trades $t_1, \ldots, t_q \in T_i' \cap T_j'$ are admissible, for each of the trades a separate contract is closed. The individual contracts are aggregated via disjunctions, i.e. $\Gamma_1 \vee \cdots \vee \Gamma_q$, since only one of the trades has to be executed.

Once a contract has been executed, both parties evaluate whether the other party's obligations stated in the contract have been fulfilled. This requires monitoring the execution of the Web service. The requester is interested in the properties exposed by the service. Therefore, monitoring methods are required that enable the collection of execution information for each attribute within the configuration. For example, this might include measuring 'Response Time', 'Availability' or other quality of service guarantees, evaluation of error messages returned by the service, judging quality of the data returned, etc. Metrics and methods for measuring such quality of attributes are presented in [SMS+02, Lud03], for instance. The result of the monitoring is thus exactly one configuration $c^E \in C$ that has been executed by the service. Consequently, a contract Γ is interpreted simply by evaluating the function $\varrho(\Gamma, c^E)$ defined in Equation 5.10.

However, some service characteristics are hardly observable by the other party and thus cannot be verified automatically. For example, consider the obligation which requires the provider to not disclose private data of the customer to third parties. Although this is an important regulation in the contract, it cannot be directly monitored by the customer. Nevertheless, the clause has to be part of the contract in order to provide a legal instrument for the customer to sanction illegal disclosure of her private data. In order to enable automated evaluation of contracts in presence of not observable attributes the following methods is applied: before evaluating the contract we remove all constraints defined on attributes that cannot be observed. The attribute values of these attributes can be set to any value within the attribute range. After this manipulation the function $\varrho(\Gamma, c^E)$ is evaluated as usual.

5.4 Conclusion

In this chapter, we have presented the conceptual design of a Web service market model using an abstract mathematical notation. Since this notation is not directly processable by computers (Requirement (*R7*)) and there is no standardized serialization for exchanging the communication primitives in the Web (Requirement (*R6*)), in the embodiment phase of the market engineering process the model is implemented by means of ontologies. Ontologies are powerful enough to provide the required expressivity, while fostering interoperability in the Web through a standardized syntax and semantics. In Chapter 6, the policy model as well as the communication primitives introduced in Section 5.1 and 5.2 are formalized by means of ontologies. Based on these ontologies, in Chapter 7 an implementation of the Web service contracting and contract monitoring mechanisms presented in Section 5.3 is introduced.

Chapter 6

An Ontology Framework for Web Service Markets

One of the main contributions of this work is an ontology framework that unambiguously defines the vocabulary for communication in Web service markets. The key modules of the ontology framework are ontologies for representing policies, bids and contracts. In this chapter, we first give a short overview of the entire framework (Section 6.1). This includes a discussion about the foundational ontology that is used as modeling basis and clarifies the relation and interdependencies between the different ontology modules. Subsequently, we introduce the ontologies for expressing policies, bids and contracts in Section 6.2, 6.3 and 6.4, respectively.

6.1 Overview

As discussed in Section 2.4.3, ontologies can be categorized into four major classes: top-level ontologies, core ontologies, domain/task ontologies and application ontologies. The ontology framework is structured according to these categories by providing a stack of ontology modules. This structure is illustrated in Figure 6.1 and comprises the following three layers:

- The framework is based on a philosophically sound formalization of domain-independent concepts and relations that are captured by the top-level ontology DOLCE (see Section 2.4.4). By capturing typical *ontology design patterns* (e.g. location in space and time), foundational ontologies provide the basic vocabulary for structuring and formalization of application ontologies. Reusing these building blocks considerably reduces modeling effort. Furthermore, they provide precise concept definitions through a high degree of axiomatization. Thereby, foundational ontologies facilitate the conceptual integration of different ontologies and thus ensure interoperability in heterogeneous environments. DOLCE provides important ontology design patterns such as contextualization and is available in the ontology language we use (i.e. OWL-DL). The ontology DOLCE together with its modules Ontology of Description and Situations (DnS), Ontology of Information Objects (OIO) and Ontology of Plans (OoP) has been introduced in Section 2.4.4. The concepts that are directly used for alignment of our ontology are introduced in Table 2.3 on page 43.

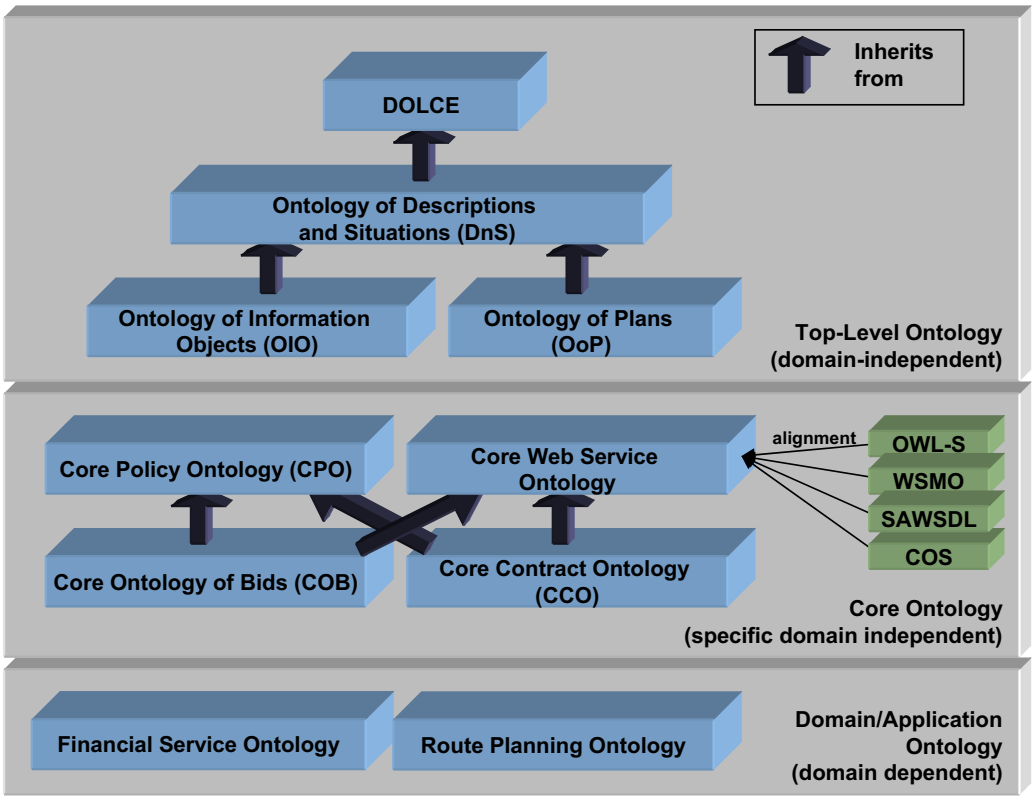

Figure 6.1: Ontology framework for Web service markets.

- By means of the DOLCE vocabulary, additional ontology modules can be defined that capture information within a Web service market. This includes generally applicable ontology modules that formalize the notion of policies, Web services, bids and contracts. The *Core Policy Ontology* (CPO) formalizes the notion of goal and utility function policies defined in Section 5.1. This module is used to define two additional ontology modules that formalize the communication in electronic markets: The *Core Ontology of Bids* (COB) that uses utility function policies to efficiently encode configurable Web service offers and requests; and the *Core Contract Ontology* (CCO) the relies on goal policies to represent requester and provider obligations. In addition, an ontology for representing services is required. The *Core Ontology of Services* (COS) is an ontology module based on DOLCE for modeling the technical aspects of services as presented in [Obe05, OLG+06]. Other ontologies that can be used to describe services – such as OWL-S or WSMO – are not directly aligned with DOLCE.[1] The ontology modules Core Policy Ontology, Core Ontology of Bids, Core Contract Ontology and Core Ontology of Services are classified as core ontologies since they can be used for different purposes, in different domains and for different applications.

- While the first two layers contain domain-independent off-the-shelf ontologies, the third layer comprises ontologies for customizing the framework to

[1]For OWL-S a DOLCE alignment exists which is presented in [MOGS04]. However, due to some problematic aspects of OWL-S, such as conceptual ambiguity, poor axiomatization, loose design and narrow scope, the alignment to DOLCE requires a major restructuring of OWL-S.

specific domains (e.g. an ontology for modeling information that is returned by a service such as credit or route information).

In the following, we focus on the second layer of our ontology framework. We introduce the Core Policy Ontology in Section 6.2, the Core Bidding Ontology in Section 6.3 and the Core Contract Ontology in Section 6.4. The Core Web Service Ontology has already been developed in [Obe05, OLG+06] and is therefore not addressed. For the formalization of the core ontologies, we use the ontology formalisms introduced in Section 2.4.2.

Ontology modules of the first layer are reused and published in [MBG+02a, GBCL04]. They can be downloaded from `http://www.loa-cnr.it/DOLCE.html`. Regarding the second layer, we reuse the Core Software Ontology and the Core Ontology of Services published in [Obe05, OLG+06]. They are available at `http://cos.ontoware.org`. Parts of our contributed ontology modules originate from prior publications. The Core Policy Ontology is partly introduced in [LEO05, OLG+06, LAO+06, LASW06] and the Core Contract Ontology in [LML+05, LLM07]. The latter ontology modules as well as the Core Ontology of Bids are available from `http://emo.ontoware.org`.

6.2 Core Policy Ontology (CPO)

The Core Policy Ontology (CPO) provides primitives for specifying goal and utility function policies that have been introduced in Section 5.1. As outlined in the following, expressing policies by means of ontologies provides several important advantages [GLA+04, UBJ+04, Kag04, KPKH05]:

- Since the semantics of ontology languages is defined using a formal calculus, they provide logical inferencing mechanisms, which allow us to reason about policy containment, i.e. whether the requirements for supporting one policy are a subset of the requirements for another. For example, a provider with a policy A restricting the coverage of a route planning service to routes within Europe matches a requester stating a policy B, which requires a service to support routes within Germany. We thus have to determine whether policy A contains policy B. This is particularly important to address Requirements (*R1*), (*R6*) and (*R9*). Beyond policy containment, the following reasoning task for policy evaluation can be identified [KPKH05]:
  - policy inclusion: if x meets policy A then it also meets policy B
  - policy equivalence: if policy A is evaluated to a measure u then also policy B is evaluated to u and vice versa.
  - policy incompatibility: if x meets policy A it cannot meet policy B
  - policy incoherence: policy A cannot be met
  - policy conformance: x meets policy A
- Since ontologies come with standard languages and pre-existing reasoners they are easily exchangeable in the Web, and policy editing and processing can be done with standard tools, i.e. no separate policy engine is required. In

Ontology entity	DOLCE alignment	Abstract model
Attribute	*DnS:Parameter*	L
AttributeValue	*DOLCE:Region*	A
Configuration	*DnS:Situation*	C
UtilityValue	*DOLCE:Region*	$\{0,\ldots,1\}$
satisfiesPolicy	*DnS:satisfies*	$\{(\Phi,c)\|G_\Phi(c)=1\}$
overallDegree	*DnS:satisfies*	$\{(f,c)\|\exists v : U_f(c)=v\}$
Context	*DnS:FunctionalRole*	K
ContextDimension	*DnS:Parameter*	$\{\delta_1,\ldots,\delta_m\}$

Table 6.1: Correspondence of Core Policy Ontology and Abstract Policy Model. A row in the table with the *ontology entity* ϕ, the *DOLCE concept* ψ and a set of the abstract model E should be understood as follows: $\phi \sqsubseteq \psi$ and there is an interpretation $\mathcal{I}$ such that $\phi^{\mathcal{I}} = E$ holds.

addition, existing policy languages can be mapped into the same background formalism, which makes the different policy specifications interoperable (Requirement (*R6*)).

- Ontologies feature declarative policy specification, which provides the possibility to change policies at runtime (Requirement (*R8*)) and to exchange them with other parties in the system (e.g. share them with other market participants) as required in Figure 3.1 on page 48. Moreover, representing policies as part of the knowledge base provides the ability to state knowledge about policies. For example, one could define the application area of a policy, the author or the context in which the policy should be applied (Requirement (*R4*)).

In the remainder of this section, we implement the abstract policy model introduced in Section 5.1 as follows: Since policies require the expression of the functions $G(\cdot)$ and $U(\cdot)$, we first extend the DOLCE ground ontology by modeling primitives required for representing functions (Section 6.2.1). Second, based on these functions, we show how the DOLCE ontology module Descriptions & Situations is applied to model Web service configurations and policies over these configurations (Section 6.2.2). In addition, we discuss in this section how configurations are evaluated according to the specified policies, i.e. how the functions $G(\cdot)$ and $U(\cdot)$ are evaluated. Finally, a mechanism to specify and evaluate combinations of policies is introduced in Section 6.2.3. Table 6.1 illustrates the mapping between the core policy ontology and the abstract policy model introduced in Section 5.1. Since the ontological model is more comprehensive and fine-grained than the abstract model, only a subset of the concepts and relations are mapped to the abstract model. While the abstract model is geared specifically towards Web service markets, the core policy ontology is applicable in a much broader context.[2]

[2]Note that although in the following we focus on applying the policy ontology for specifying scoring and pricing functions in electronic markets, due to its generality it is not restricted to this

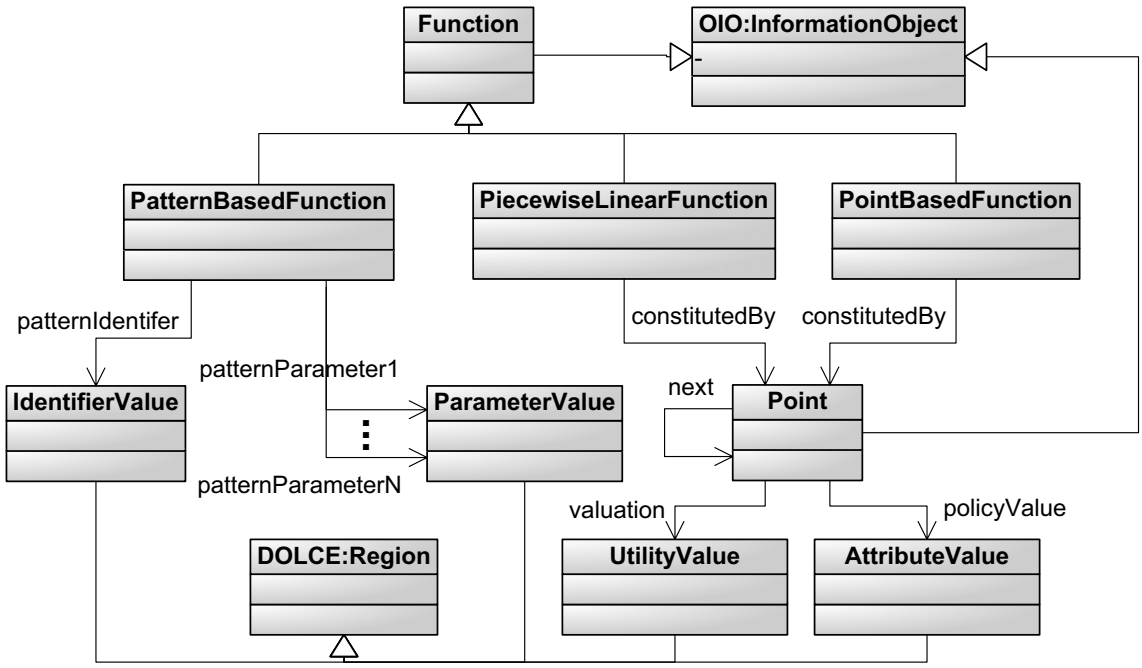

Figure 6.2: Representation of value functions. For the reader's convenience, we define DL axioms informally via UML class diagrams, where UML classes correspond to OWL concepts, UML associations to object properties, UML inheritance to subconcept-relations and UML objects to OWL individuals [BVEL04].

6.2.1 Valuation Functions

As discussed in Section 5.1, utility function policies are expressed via functions $V : C \rightarrow \mathbb{R}$ that map each configuration $c \in C$ to a corresponding valuation between 0 and 1 (or $-\infty$), where a valuation of $-\infty$ refers to forbidden alternatives and a valuation of 1 to the optimal alternative [LEO05].[3] We now show how the fundamental concepts formalized in DOLCE can be extended to allow expressing utility functions.

As depicted in Figure 6.2, a *Function* is a specialization of *OIO:InformationObject* which represents abstract information that exists in time and is realized by some entity [GBCL04]. Currently our framework supports three ways of defining *Functions*: (*i*) *Functions* can be modeled by specifying sets of points that explicitly map attribute values to valuations. This is particularly relevant for nominal attributes. (*ii*) We allow to extend these points to piecewise linear value functions, which is important when dealing with continuous attribute values, such as the response time of a service. (*iii*) Thirdly, we allow reusing typical function patterns, which are mapped to predefined, parameterized valuation rules. Note that such patterns are not restricted to piecewise linear functions since all mathematical operators provided by the rule language can be used. The different ways of declarative modeling functions are discussed next in more detail.

domain. In fact, it can be used for a wide range of multi-attribute decision problems, e.g. to define preferences over agent strategies or penalties in electronic contracts.

[3]Note that normalization to the range [0,1] is not required. For example, for specifying scoring and pricing policies a range $\mathbb{R}_0^+$ might be more convenient.

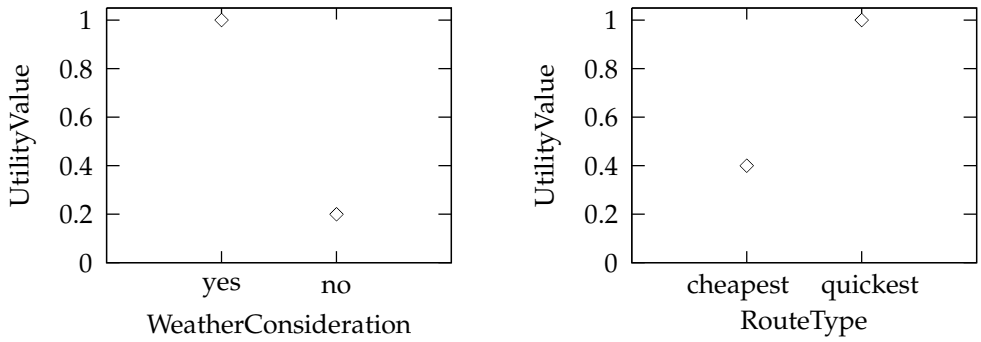

Figure 6.3: Example of a point-based value function

Point-based Functions

As depicted in Figure 6.2, *PointBasedFunctions* are *Functions* that are *constitutedBy* a set of *Points* (Axiom (A9)). Each *Point* has a property *policyValue* referring to an *AttributeValue* $a \in A$ and a property *valuation* that assigns exactly one *UtilityValue* to this attribute value (Axiom (A10)).[4] For the reader's convenience we will often use the syntax (x,y) to refer to a *Point* instance with *policyValue* x and valuation y. An *AttributeValue* is a specialization of *DOLCE:Region* that defines which attribute values a certain attribute may adopt (Axiom (A11)). It thus corresponds to the set A in the abstract model. For example, the attribute *WeatherConsideration* in the route planning example (Example 4.3) requires the *DOLCE:Region WeatherConsiderationValue* containing the elements "yes" and "no". Similarly, the *DOLCE:Region UtilityValue* comprises the range $[0,1]$ and $-\infty$. The following axioms formally capture these relations:

$$\text{(A9)} \quad \mathit{PointBasedFunction} \sqsubseteq \mathit{Function} \sqcap \exists \mathit{constitutedBy}.\mathit{Point}$$

$$\text{(A10)} \quad \mathit{Point} \sqsubseteq =_1 \mathit{policyValue}.\mathit{AttributeValue} \sqcap =_1 \mathit{valuation}.\mathit{UtilityValue}$$

$$\text{(A11)} \quad \mathit{AttributeValue} \sqsubseteq \mathit{DOLCE{:}Region}$$

$$\text{(A12)} \quad \mathit{UtilityValue} \sqsubseteq \mathit{DOLCE{:}Region}$$

Example 6.1 *In our route planing example, a requester might specify her preferences with respect to the service attribute* WeatherConsideration *by a* PointBasedFunction*, which is* constitutedBy *two instances of* Point *with* ("yes",1) *and* ("no",0.2)*. Thus, the requester would highly prefer weather information to be taken into account, but has some small use for routes calculated without weather information. Similarly, the preferences for the attribute* RouteType *calculation can be defined with* Points ("quickest",1) *and cheapest* ("cheapest",0.4)*. These mappings are illustrated in Figure 6.3.*

In order to evaluate this function, additional axioms are required that more closely define the semantics of the concepts *PointBasedFunction* and *Point* as well

[4]Note that in case we have dependent attributes and thus complex value functions $v_{j^*}(x_k,\ldots,x_l)$ (cf. Section 5.1) each *Point* might have several *policyValue* relations, i.e. $\mathit{policyValue}_k,\ldots,\mathit{policyValue}_l$.

as their relations. Rule (R2) below defines how the *UtilityValue* v of a *AttributeValue* x can be determined based on the specification of the *PointBasedFunction* f. For this purpose, we iterate over all *Points* constituting the function and compare their property *policyValue* to the desired attribute value x.

$$\text{(R2)} \quad degree(f,x,v) \leftarrow PointBasedFunction(f), constitutedBy(f,p), policyValue(p,pv), match(x,pv), valuation(p,v)$$

The comparison of attribute values is realized by the *match*-predicate. This predicate has to be customizable since the way attributes are compared depends on the domain of interest, i.e. on the concrete attribute. In order to keep Rule (R2) applicable for all attributes, we specify this in a separate matching rule. For example, considering the attribute *WeatherConsideration*, for matching the attribute values a simple string matching predicate as provided with the built-in *swrlb:equal* is sufficient. Rule (R3) illustrates this by defining the matching rule for the attribute *WeatherConsideration*.

$$\text{(R3)} \quad match(x,y) \leftarrow WeatherConsiderationValue(x), WeatherConsiderationValue(y), swrlb{:}equal(x,y)$$

Unfortunately, in many cases attribute values have to be described in a more complex way beyond simple strings or numbers, e.g. to express subclass relations between attribute values. In such cases, it might be required to model attribute values as concepts in OWL. Since in our ontology they are modeled as individuals, a meta-modeling approach is required where a URI can be treated as concept as well as instance.[5] This allows us to specify preferences on a more abstract level and thus avoids enumerating all possible attribute values.

Example 6.2 *Consider an attribute* IndicatedAttraction *that specifies which types of attractions along the route can be suggested by a certain service. In this case, the corresponding value space* IndicatedAttractionValue *might comprise the alternatives* CulturalAttraction, HistoricSite, Museum *and* Castle *which are all related to each other. In particular,* CulturalAttraction *can be seen as a class containing all other values.* HistoricSite*, in turn, comprises* Castles *but not* Museums*. Consequently, a scoring function mapping* HistoricSites *to a valuation of* 0.8 *has to assign the same value to information about* Castles *along the route (although this might not be specified explicitly). Such a behavior can be realized by defining a* Point *that maps the* AttributeValue HistoricSite *to a* UtilityValue *of* 0.8 *and another* Point *that maps everything else to* 0 *using the concept definition* $Attraction \sqcap \neg HistoricSite$*. Similar to the attributes above, we can define a matching rule for the attribute* IndicatedAttraction *by replacing the built-in implementing string matching with the built-in* subsumes *that features DL subsumption checking between two concepts.*

[5] Although such an approach is outside of the ontology formalism at hand and part of OWL-Full, many reasoners such as KAON2 can handle meta-modeling to some extent [Mot05].

(R4) $match(x,y) \leftarrow IndicatedAttractionValue(x), IndicatedAttractionValue(y), subsumes(y,x)$

Beyond subsumption several other notions of matching description logic concepts have been proposed in literature (e.g. [LH03, NSDM03, GMP04, BK06a]). We support these different notions of match by providing a flexible framework that can be customized via declarative matching rules.

Piecewise Linear Functions

In order to support defining *Functions* also on continuous properties, we introduce *PiecewiseLinearFunctions* as shown in Figure 6.2. Continuous attributes exhibit a natural ordering between the attribute values A which can be utilized for specifying the function. We utilize the property *next* which connects two *Points* with adjacent attribute values in order to interpret *Points* as continuous functions. A *PiecewiseLinearFunction* is defined by at least two points. This is captured by the following axiom:

(A13) $PiecewiseLinearFunction \sqsubseteq Function \sqcap \geq_2 constitutedBy.Point$

Such adjacent *Points* can be connected by straight lines forming a piecewise linear value function as depicted in Figure 6.4. For every line between the *Points* (x_1,y_1) and (x_2,y_2) as well as a given *AttributeValue* x, we calculate the valuation v as follows.

$$v = \begin{cases} \frac{y_2-y_1}{x_2-x_1}(x-x_1)+y_1, \text{ if } x_1 \leq x < x_2 \\ 0, \text{ otherwise} \end{cases}$$

This equation is formalized by a predicate $cal(v,x,x_1,y_1,x_2,y_2)$. This predicate can be realized either directly by means of a built-in or by exploiting the math as well as the comparison built-in predicates provided by the rule language.[6]

Using this predicate, Rule (R5) defines the valuation of a certain attribute value x (as Rule R2 does for *PointBasedFunctions*). The rule makes sure that only adjacent *Points* are considered in the calculation.[7]

[6] Although predicates with arity higher than two cannot be modeled with the formalism at hand directly, many reasoning tools support them. Moreover, techniques for reifying higher arity predicates are well known [HSTT00].

[7] For the readers convenience, throughout the work we avoid repetition of predicates by using the $\bigwedge$-operator. For example, the term '$\bigwedge_{i\in\{1,2\}}(policyValue(p_i,pv_i))$' is written instead of '$policyValue(p_1,pv_1), policyValue(p_2,pv_2)$'.

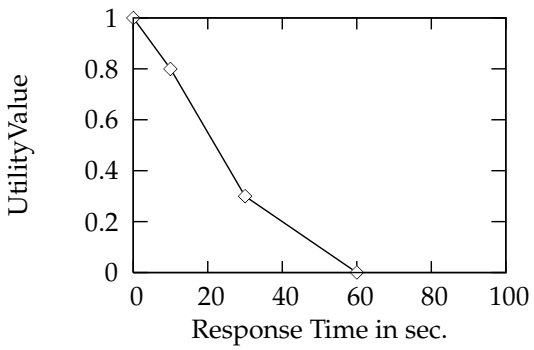

Figure 6.4: Example of a piecewise linear value function

(R5) $degree(f,x,v) \leftarrow PiecewiseLinearFunction(f),$
$\bigwedge_{i\in\{1,2\}} (constitutedBy(f,p_i), policyValue(p_i,pv_i),$
$valuation(p_i,v_i)), next(p_1,p_2), swrlb{:}lessThan(x,p_2)$
$swrlb{:}greaterThanOrEqual(x,p_1), cal(v,x,pv_1,v_1,pv_2,v_2)$

Example 6.3 *As an example, let us assume that the* Function *for the attribute* Response Time *of the route planing service is given by a* PiecewiseLinearFunction *with the* Points $(0,1)$, $(10,0.8)$, $(30,0.3)$, $(60,0)$ *as depicted in Figure 6.4. Now, we can easily find out which* UtilityValue *v a certain* AttributeValue *x is assigned to. The predicate $cal_v(v,x,x_1,y_1,x_2,y_2)$ is true iff the* policyValue *x is between two adjacent* Points *(x_1,y_1) and (x_2,y_2) and the* UtilityValue *of x equals v. For instance, for a* Response Time *of 20 sec. cal_v evaluates the straight line connecting the adjacent* Points *$(10,0.8)$ and $(30,0.3)$, which results in a* UtilityValue *0.675.*

Pattern-based Functions

Alternatively, value functions for continuous attributes can be modeled by means of *PatternBasedFunctions*. This type refers to functions like $u_{p_1,p_2}(x) = p_1 e^{p_2 x}$, where p_1 and p_2 represent parameters that can be used to adapt the function. In the Core Policy Ontology, these *Functions* are specified through parameterized predicates which are identified by *patternIdentifiers*. A *patternIdentifier* points to a *DOLCE:Region IdentifierValue* that uniquely refers to a specific rule predicate.

(A14) $PatternBasedFunction \sqsubseteq Function \sqcap =_1 patternIdentifier.IdentifierValue$

This predicate is denoted *pattern*. A *patternParameter* defines how a specific parameter of the *pattern*-predicate has to be set. For allowing an arbitrary number of parameters in a rule, universal quantification over instances of *patternParameter*

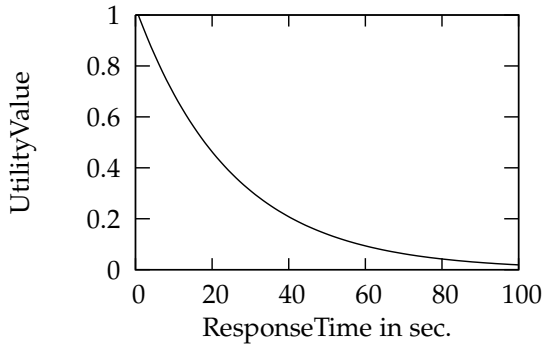

Figure 6.5: Example of a pattern-based valuation function

would be necessary in the body of the rule. Since this is not expressible with the rule language at hand, the different parameters are modeled as separate properties in the ontology, viz. *patternParameter1*,..., *patternParameterN*. Although this restricts the modeling approach as the maximal number of parameters has to be fixed at ontology design time, for most applications this is sufficient and we believe that keeping the logic decidable justifies this limitation.

As shown in the example below (Rule (R6)), each *pattern* is identified by a hard-coded internal string. This is required to specify, which pattern is assigned to a certain attribute in the ontology. Thus, in order to find out which *pattern*-predicate is applicable, the *patternIdentifier* specified in the policy is handed over to the predicate and then it is compared to the internal identifier. If the two strings are identical, the predicate is applied to calculate the *UtilityValue* that is assigned to a *AttributeValue*.

(R6) $$\begin{aligned} &\mathit{pattern}(v, id, x, p_1, \ldots, p_n) \leftarrow \\ &\quad \mathit{String}(id), \mathit{AttributeValue}(x), \mathit{UtilityValue}(v), \\ &\quad \mathit{swrlb{:}equal}(id, \text{“}\mathit{id{:}exp}\text{”}), \mathit{swrlb{:}multiply}(x, t_1, p_2), \\ &\quad \mathit{swrlb{:}pow}(t_2, \text{“}2.70481\text{”}, t_1), \mathit{swrlb{:}multiply}(v, p_1, t_2) \end{aligned}$$

Example 6.4 *We again focus on the attribute* ResponseTime *of the route planning service. Assume the preferences for* Response Time *are given by the exponential function* $u_{p_1,p_2}(x) = p_1 e^{p_2 x}$ *with the* patternParameter $p_1 = 1.03$ *and* $p_2 = -0.04$ *(Figure 6.5). Rule* (R6) *formalizes the pattern. The internal identifier in this example is* ‘id:exp’. *The corresponding comparison is done by the built-in equal, which is satisfied if the first argument is the same as the second argument.*

SWRL supports a wide range of mathematical built-in predicates (cf. [HPSB+04]) and thus nearly all functions can be supported. As in our example, these functions are typically parameterized only by a rather small number of parameters. Therefore, we believe that constraining the number of parameters at ontology design time has only few practical implications.

Based on the definition of the *pattern*-predicate, we can calculate the *UtilityValue* of an *AttributeValue* according to a *PatternBasedFunction* using the following rule.

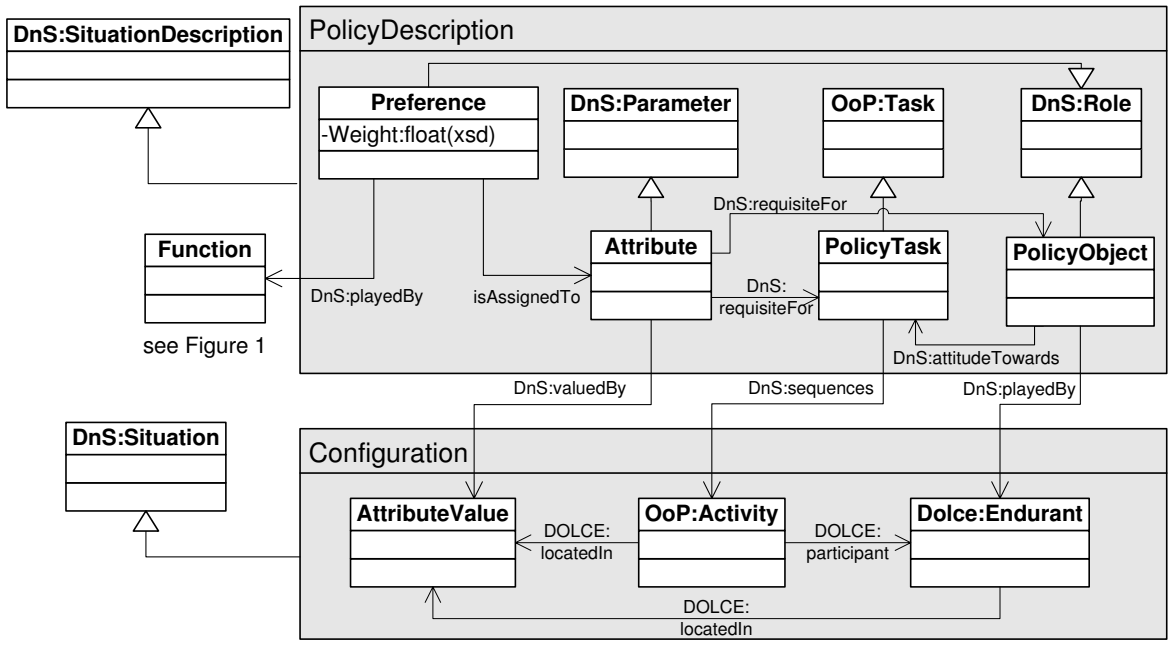

Figure 6.6: Policy description framework. To improve the readability we illustrate certain relations by plotting UML classes within other UML classes: The class *PolicyDescription* has a *DnS:defines*-relation and the class *Configuration* a *DnS:settingFor*-relation to each contained class.

$$\text{(R7)} \quad degree(f,x,v) \leftarrow PatternBasedFunction(f), patternIdentifier(f,id), \\ patternParameter(f,p_1),\ldots,patternParameter(f,p_n), \\ pattern(v,id,x,p_1,\ldots,p_n)$$

Based on the notion of *Functions* introduced above, we show in the following how they are used to define and reason about policies.

6.2.2 Modeling Policies and Configurations

As discussed in Chapter 5, we formalize preferences of a user as well as pricing information of a provider in a functional form by means of utility function policies. For instance, a price-conscious user might prefer a cheap service although the service has a rather slow response time, whereas a time-conscious user might accept any costs for a fast service. Hence, policies can be seen as different views on a certain configuration. For modeling such views, we use and specialize the DOLCE module Descriptions & Situations (DnS) which provides a basic theory of contextualization [GBCL04]. Hence, a certain configuration can be considered as more or less desirable depending on the scoring policies of a buyer or a configuration can be priced differently depending on the pricing policies of a seller.

As depicted in Figure 6.6, when using DnS with DOLCE, we distinguish between DOLCE *ground entities* that form a *DnS:Situation* and *descriptive entities* composed in a *DnS:SituationDescription*, i.e. the view in which *DnS:Situations* are in-

terpreted. We specialize the *DnS:SituationDescription* to a *PolicyDescription* that can be used to evaluate concrete *Configurations* which are modeled as special kind of *DnS:Situations*. This distinction enables us, for example, to talk about products as roles on an abstract level, i.e. independent from the concrete entities that play the role. For instance, a certain product configuration can be evaluated in the light of either a pricing policy of the seller or the preferences of a user depending on the point of view.

In the following, we describe how such *Configurations* and *PolicyDescriptions* are modeled and then show how the evaluation of policies is carried out.

Configuration

In a first step, we define the ground entities that constitute a *DnS:Situation*. In our context, such *DnS:Situations* reflect multi-attribute descriptions of decision alternatives (e.g. real-world objects or activities). In a concrete *DnS:Situation* these products have one distinct configuration. Recall that in Section 5.1 we defined the set of configurations C as the cartesian product of the attributes $C = A_1 \times \cdots \times A_n$. Hence, we model *Configuration* as a subclass of *DnS:Situation* that exactly defines one configuration $c \in C$ of a product (Axiom (A15)). Since there are various different ways of describing products, a generic approach is used in this work, where concrete objects and activities are represented by instances of *DOLCE:Endurant* and *OoP:Activity*, respectively. Attributes of *DOLCE:Endurants* and *OoP:Activities* are modeled via the *locatedIn* property that points to a value range represented by the *DOLCE:Region AttributeValue* [GST05].[8] This approach is illustrated in the lower part of Figure 6.6.

The following axioms capture this notion by ensuring that each *Configuration* comprises at least one multi-attribute object (Axiom (A15)). Axiom (A16) ensures that each *AttributeValue* belongs to exactly one *DOLCE:Endurant* or *OoP:Activity*. Moreover, we can define a context for each *DOLCE:Perdurant* and *DOLCE:Endurant*. This is realized by means of the concept *ContextRegion* (Axiom (A17)) which contains the allowed values for a context dimension δ, i.e. for the context dimension 'Location' *ContextRegion* might represent a list of countries.

$$\text{(A15)} \quad \mathit{Configuration} \sqsubseteq \mathit{DnS{:}Situation} \sqcap \exists \mathit{DnS{:}settingFor}.(\mathit{DOLCE{:}Endurant} \sqcup \mathit{OoP{:}Activity}) \sqcap \exists \mathit{DnS{:}settingFor}.\mathit{AttributeValue}$$

$$\text{(A16)} \quad \mathit{AttributeValue} \sqsubseteq \mathit{DOLCE{:}Region} \sqcap =_1 \mathit{DOLCE{:}locatedIn}^{-}.(\mathit{OoP{:}Activity} \sqcup \mathit{DOLCE{:}Endurant})$$

$$\text{(A17)} \quad \mathit{ContextRegion} \sqsubseteq \mathit{DOLCE{:}Region} \sqcap \exists \mathit{DOLCE{:}locatedIn}^{-}.(\mathit{DOLCE{:}Endurant} \sqcup \mathit{DOLCE{:}Perdurant})$$

[8] Note that this approach is more general than our abstract model, where a Web service is fully described by a set of attributes.

Example 6.5 *A configurable Web service can be modeled by a combination of CSO:ComputationalObjects and CSO:ComputationalActivities, which specialize DOLCE:Endurants and OoP:Activities, respectively [Obe05]. Hereby, specializations of CSO:ComputationalActivities capture ServiceActivities like RoutePlanningActivity. Specializations of CSO:ComputationalObjects represent the objects involved in such a ServiceActivity (e.g. inputs and outputs). A RoutePlanningActivity might have several DOLCE:Qualities that are located in specializations of AttributeValue such as WeatherConsiderationValue, IndicatedAttractionValue, ResponseTimeValue or AvailabilityValue. In addition, the RoutePlanningActivity involves a ServiceOutput which specializes CSO:ComputationalObject. ServiceOutput is associated to a RouteTypeValue that defines whether the output is the cheapest or the fastest route.*

Policy Description

In a second step, we define views on the ground entities defined in Section 6.2.2. This is realized by specializing the descriptive entities *DnS:FunctionalRoles*, *DnS:Courses*, *DnS:Parameters*, and *DnS:SituationDescriptions* as introduced in Table 2.3. Policies are modeled as specialization of *DnS:SituationDescription* (Axiom (A18)). They are called *PolicyDescriptions* and have to *DnS:define* a *PolicyObject*[9] or *PolicyTask* that represent the entity on which the policy is defined, e.g. this could be a certain type of good or a service. Since *PolicyObjects* and *PolicyTasks* are modeled as specialization of *DnS:FunctionalRoles* and *OoP:Tasks* (Axiom (A19) and Axiom (A20)), policies can be defined on an abstract level without referring to a concrete *DOLCE:Endurant* or *OoP:Activity*. For instance, policies can be defined for a certain service category (specialization of *OoP:Task*) such as route planning services in general. Then all *OoP:Activities* that fulfill the task of route planning in a certain *DnS:Situation* are evaluated according to the policy. Axiom (A18) formally defines a *PolicyDescription*. It ensures that at least one entity is constrained by means of the *DnS:Parameter Attribute*. Moreover, each *Attribute* that is introduced has to constrain exactly one *PolicyObject* or *PolicyTask* which can be realized by means of the *DnS:requisiteFor*-relation (Axiom (A21)).

$$\text{(A18)}\quad \mathit{PolicyDescription} \sqsubseteq \mathit{DnS{:}SituationDescription} \sqcap \exists \mathit{DnS{:}defines}.(\mathit{PolicyObject} \sqcup \mathit{PolicyTask}) \sqcap \exists \mathit{DnS{:}defines}.\mathit{Attribute}$$

$$\text{(A19)}\quad \mathit{PolicyObject} \sqsubseteq \mathit{DnS{:}FunctionalRole} \sqcap \exists \mathit{DnS{:}definedBy}.\mathit{PolicyDescription}$$

$$\text{(A20)}\quad \mathit{PolicyTask} \sqsubseteq \mathit{OoP{:}Task} \sqcap \exists \mathit{DnS{:}definedBy}.\mathit{PolicyDescription}$$

$$\text{(A21)}\quad \mathit{Attribute} \sqsubseteq \mathit{DnS{:}Parameter} \sqcap =_1 \mathit{DnS{:}requisiteFor}.(\mathit{PolicyObject} \sqcup \mathit{PolicyTask})$$

[9] *PolicyObjects* can represent entities that are passively or actively involved in a certain *OoP:Task*. For example, a *PolicyObject* could represent a user that actively executes an service invocation as well as a Web service, which is passively involved in an invocation. Sometimes these two aspects are distinguished using two separate concepts. For instance in [LEO05, OLG+06], an additional concept *PolicySubject* is introduced that refers to the active entities.

Up to now, a *PolicyDescription* can be used to define constraints $\phi \in \Phi$ on certain properties of an entity. This is exactly what we consider as goal policies. A similar approach is used in [OLG+06] for expressing policies such as access rights. As discussed in Section 5.1, utility function policies generalize goal policies by addressing the fact that configurations are preferred to varying degrees depending on the concrete attribute values. Therefore, some extensions to the basic model are required. The *DnS:FunctionalRole Preference* is introduced which assigns *Preferences* to an *Attribute* (via the *isAssignedTo*-relation). This enables modeling additive preference functions. Thus, preference structures on attributes are imposed by *Functions*. As discussed in Section 6.2.1, *Functions* are *OIO:InformationObjects*. They play the role of *Preferences* in a *PolicyDescription* and define how *AttributeValues* are mapped to *UtilityValues* (Axiom (A22)). *Preferences* might be applicable only in a certain context. That means that a policy defines which *Function* should be used for which attribute and in which context. The set of contexts K are captured by the *DnS:FunctionalRole Context*, which comprises an instance for each $k \in K = \delta_1 \times \cdots \times \delta_m$ (Axiom (A24)). The set $\{\delta_1, \ldots, \delta_m\}$ is modeled by *ContextDimension* (Axiom (A25)).

(A22) $\mathit{Preference} \sqsubseteq \mathit{DnS:FunctionalRole} \sqcap =_1 \mathit{DnS:playedBy.Function} \sqcap =_1 \mathit{DnS:requisites.Weight} \sqcap \forall \mathit{applicableIn.Context}$

(A23) $\mathit{applicableIn} \sqsubseteq \mathit{DOLCE:part}$

(A24) $\mathit{Context} \sqsubseteq \mathit{DnS:FunctionalRole} \sqcap \exists \mathit{DnS:requisiteFor}^{-}.\mathit{ContextDimension}$

(A25) $\mathit{ContextDimension} \sqsubseteq \mathit{DnS:Parameter} \sqcap \exists \mathit{DnS:requisiteFor.Context} \sqcap \forall \mathit{DnS:requisiteFor.Context} \sqcap \forall \mathit{DnS:valuedBy.ContextRegion}$

Besides defining *Functions*, *Preferences* also define the relative importance of the given *Attribute* via the *DnS:Parameter Weight* and the *DOLCE:Region WeightValue* (omitted in Figure 6.6), which corresponds to factor λ in the abstract model.

Example 6.6 *As an example, consider the scoring policy for the property response time of a Web service. To express this, we introduce in Axiom* (A26) *a new instance of OoP:Task, called WebServiceTask . In addition, Axiom* (A27) *introduces an Attribute ResponseTime that represents a constraint (DnS:requisiteFor) that has to be fulfilled by WebServiceTask (Axiom* (A28)*). In order to define preferences over all possible attribute values, ResponseTime is DnS:valuedBy a AttributeValue ResponseTimeValue comprising the entire value space (e.g. represented by a subclass of DOLCE:Temporal-Region) as specified in Axioms* (A29) *and* (A30)*. Moreover, the instance RTPreference of the concept Preference is assigned to ResponseTime and is played by an instance of a PatternBasedFunction or PiecewiseLinearFunction (Axioms* (A32) – (A34)*). These Functions map AttributeValues to UtilityValues as discussed in Section 6.2.1.*

(A26) $\mathit{OoP:Task}(\mathit{WebServiceTask})$

(A27) $\mathit{Attribute}(\mathit{ResponseTime})$

(A28) *DnS:requisiteFor*(*ResponseTime*, *WebServiceTask*)
(A29) *AttributeValue*(*ResponseTimeValue*)
(A30) *DnS:valuedBy*(*ResponseTime*, *ResponseTimeValue*)
(A31) *Preference*(*RTPreference*)
(A32) *isAssignedTo*(*RTPreference*, *ResponseTime*)
(A33) *PatternBasedFunction*(*RTFunction*)
(A34) *DnS:playedBy*(*RTPreference*, *RTFunction*)

After presenting how *Configurations* as well as *PolicyDescriptions* are modeled, we introduce the rules for evaluating concrete *Configurations* with respect to given *PolicyDescriptions*. We show how pricing policies are applied to determine the price of a configuration or scoring policies to determine the willingness to pay.

Policy Evaluation

With our approach, policies that define *Preferences* no longer lead only to a pure boolean statement about the conformity of a *Configuration*, but rather to a degree of conformity of the *Configuration*. Therefore, the original *DnS:satisfies*-relation between a *DnS:Situation* and *DnS:SituationDescription* is not sufficient any more since additional information about the degree of conformity has to be captured. However, since checking for satisfaction can be interpreted as the evaluation of the goal policy aspect in the *PolicyDescription*, meeting the constraints in the the goal policy can be seen as a necessary prequisite. This is captured by the following rule which refines the *DnS:satisfies*-relation. The reader familiar with DOLCE will notice that Rule (R8) largely corresponds to the *completely-satisfies* relation described in [GST04]. Since the formalism at hand is not expressive enough to capture this relation directly, we provide a workaround that explicitly enumerates the attributes $A_1, \ldots, A_n$ and checks for classification of an appropriate ground entity, thus implementing *qualified satisfaction* (cf. [GST04]). Note that we assume an own *AttributeValue* concept for each *Attribute*.

$$\begin{aligned}(\text{R8}) \qquad \mathit{satisfiesPolicy}(c,p) \leftarrow\ & \mathit{Configuration}(c), \mathit{PolicyDescription}(p),\\ & \mathit{DnS{:}satisfies}(c,p), \bigwedge_{i\in 1\ldots,n} (\mathit{Attribute}_i(a_i),\\ & \mathit{DnS{:}defines}(p,a_i), \mathit{DnS{:}valuedBy}(a_i,av_i),\\ & \mathit{DnS{:}settingFor}(c,cv_i), \mathit{match}(av_i,cv_i))\end{aligned}$$

Ontologically, modeling utility function policies requires putting in relation the *PolicyDescription*, a concrete *Configuration* and an *overallDegree* that represents the value to which the latter satisfies the former. For the sake of simplicity and compact representation, we use predicates of higher arity in the following. If *satisfiesPolicy* does not hold no further evaluation will be necessary and a value of $-\infty$ is assigned by Rule (R9).

$$(\text{R9}) \qquad \mathit{overallDegree}(c,p,v) \leftarrow \neg \mathit{satisfiesPolicy}(c,p), \mathit{assign}(v, "-\infty")$$

In line with the additive utility model defined in Equation (5.4), we first calculate the valuation for each independent set of attributes individually and then aggregate the individual valuations to get the overall degree of a configuration. The local utility values can be calculated by Rules (R2), (R5) and (R7) depending on the type of function used. The valuation derived from these rules can be interpreted as the valuation a single attribute contributes to the overall valuation. Note that the non-additive case is a simplification of this approach, where the local utility value corresponds to the overall value.

$$\begin{aligned}(\text{R10}) \quad \mathit{overallDegree}(c,p,v) \leftarrow \mathit{satisfiesPolicy}(c,p), \bigwedge_{i=1,\ldots,n} (&\mathit{DnS{:}defines}(p,a_i),\\ &\mathit{DnS{:}defines}(p,pf_i), \mathit{isAssignedTo}(pf_i,a_i), \mathit{DnS{:}valuedBy}(a_i,av_i),\\ &\mathit{DnS{:}settingFor}(c,cv_i), \mathit{match}(av_i,cv_i), \mathit{DnS{:}playedBy}(pf_i,f_i),\\ &\mathit{degree}(f_i,cv_i,v_i)), \mathit{sum}(v,v_1,\ldots,v_n)\end{aligned}$$

Rule (R10) is simplified in a sense that predicates for weighting of attributes according to their relative importance λ_i are omitted. However, adding the relevant terms for executing this calculation using SWRL built-in predicates is straightforward.

Example 6.7 *To illustrate this approach, we assume a customer with the scoring policies p based on the example* Functions *defined in Examples 6.1, 6.3, and 6.4. We can query the knowledge base to compare the* overallDegree *for* Configuration *c with respect to the* PolicyDescription *p. As an example, we assume a* Configuration *of a route planning service, which returns the cheapest route that includes information about historical sites while considering weather information. Further, a response time of 20 sec. is guaranteed. Evaluating the (local)* degree *-predicates for each* Attribute *leads to a score of* 1 *for the* Attribute WeatherConsideration, 0.4 *for* RouteType, 0.8 *for* IndicatedAttraction *and* 0.47 *for* Response Time, *respectively. Provided that all* Attributes *are equally important this* Configuration *results in a* overallDegree *of* 0.67.

The policy evaluation rules (R8) - (R10) defined above are all context independent, i.e. the context in which *PolicyTask* are executed is not considered in the evaluation. In order to extend our approach to allow context dependent policies, we introduce the predicate $\mathit{isValidIn}(p,c)$ that is true if a *PolicyDescription* p should be applied in a *Configuration* c.

$$\begin{aligned}(\text{R11}) \quad \mathit{isValidIn}(p,c) \leftarrow &\mathit{PolicyDescription}(p), \mathit{Configuration}(c)\\ &\mathit{DnS{:}defines}(p,k), \bigwedge_{l=1,\ldots,m} (\mathit{DnS{:}requisiteFor}(d_l,k)\\ &\mathit{DnS{:}valuedBy}(d_l,v_p), \mathit{DnS{:}settingFor}(c,v_s),\\ &\mathit{ContextRegion}(v_s), \mathit{match}(v_p,v_s))\end{aligned}$$

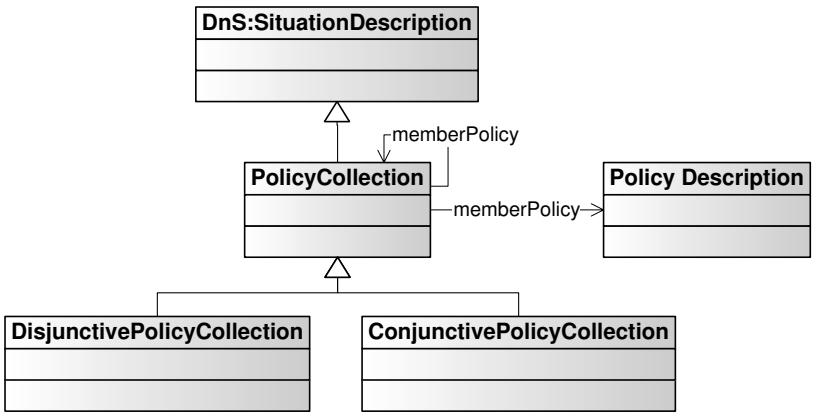

Figure 6.7: Representation of a *PolicyCollection*.

Similar to the matching of attribute values, for comparing context dimension values we also rely on the *match*-predicate, which can be easily adapted to new context ontologies. For evaluation of context dependent policies, we simply have to add the predicate *isValidIn* to the corresponding evaluation rules (R8) - (R10).

6.2.3 Policy Aggregation

Up to now we focused on scenarios where only one policy was used by a buyer or seller. However, as discussed in Section 5.1.3, policies can be combined by either a logical *and*-operator referring to a conjunction of policies (i.e. the aggregated policy is admissible if all contained policies are admissible) or a logical *or*-operator to derive a disjunction of policies (i.e. the aggregated policy is admissible if at least one contained policy is admissible). Equation 5.6 and 5.7 define the *T-norm* and *T-conorm* to combine policy conjunctions and disjunctions, respectively.

We introduce the modeling primitives required for representing conjunctions and disjunctions of policies, as shown in Figure 6.7. To be able to evaluate a certain *Configuration* with respect to a set of policies, we adapt Rule R10 in a way that it can be used not only for a single *PolicyDescription*, but also for a *PolicyCollection*. A *PolicyCollection* is defined as a *DnS:SituationDescription* that has exactly two *memberPolicy*-relations pointing to *PolicyDescriptions* or *PolicyCollections*. This is formalized using the following DL axioms:

$$\text{(A35)} \quad \begin{aligned} \mathit{PolicyCollection} \sqsubseteq\ & \mathit{DnS{:}SituationDescription} \sqcap \\ & =_1 \mathit{memberPolicy1}.(\mathit{PolicyDescription} \\ & \sqcup \mathit{PolicyCollection}) \sqcap \\ & =_1 \mathit{memberPolicy2}.(\mathit{PolicyDescription} \\ & \sqcup \mathit{PolicyCollection}) \end{aligned}$$

$$\text{(A36)} \quad \mathit{memberPolicy1} \sqsubseteq \mathit{DnS{:}expands}$$

$$\text{(A37)} \quad \mathit{memberPolicy2} \sqsubseteq \mathit{DnS{:}expands}$$

The reason why we restrict a *PolicyCollection* to exactly two *memberPolicy*-relations is the fact that SWRL does not support universal quantification in the rule body. Hence, we cannot iterate over an arbitrary number of *PolicyDescriptions* contained in the collection (e.g. the first order logic term '$\forall y.memberPolicy(x,y)$' is not expressible in SWRL). However, restricting a *PolicyCollection* to exactly two *memberPolicy*-relations is in fact no limitation, since an arbitrary number of *PolicyCollections* with two *memberPolicy*-relations can be nested. This has the same effect as multiple *memberPolicy*-relations within one *PolicyCollection*.

In order to define a relation between the members of a *PolicyCollection*, we introduce two subclasses of *PolicyCollection*, namely *ConjunctivePolicyCollection* and *DisjunctivePolicyCollection*. Then, for each of these subclasses a rule is introduced that calculates the *overallDegree* of the collection based on the *overallDegrees* of the elements contained. The following rule does the calculation for a *ConjunctivePolicyCollection* where the individual elements are connected by a logical *and*-relation based on the T-norm defined in Equation (5.6).

$$\text{(R12)} \quad overallDegree(c,p,v) \leftarrow ConjunctivePolicyCollection(p), \bigwedge_{i\in\{1,2\}} (memberPolicy_i(p,p_i), overallDegree(c,p_i,v_i)), min(v,v_1,v_2)$$

Note that Rule (R12) recursively calculates the *overallDegree* of the elements contained in the collection. Rule (R12) will only be used if a *ConjunctivePolicyCollection* is passed to the *overallDegree*-predicate. If it refers to a single *PolicyDescription*, Rule (R10) will be applied as before.

Analogously, we can define the Rule (R13) for *DisjunctivePolicyCollections* where the T-conorm (Equation (5.7)) is used to calculate the *overallDegree*.

$$\text{(R13)} \quad overallDegree(c,p,v) \leftarrow DisjunctivePolicyCollection(p), \bigwedge_{i\in\{1,2\}} (memberPolicy_i(p,p_i), overallDegree(c,p_i,v_i)), max(v,v_1,v_2)$$

DisjunctivePolicyCollections and *ConjunctivePolicyCollections* can be nested within each other provided that the leafs of the emerging tree structure are always primitive *PolicyDescriptions*.

6.3 Core Ontology of Bids (COB)

After having introduced a policy ontology for specifying valuation functions over multi-attribute objects or activities, we show how such policies are used for efficiently attaching price information to Web services in the following. As introduced in Chapter 5, a statement that captures such information is called a *bid*. It repre-

Ontology entity	DOLCE alignment	Abstract model
TradeSituation	*DnS:Situation*	T
Issuer	*DnS:FunctionalRole*	$I \cup J$
Offer	*DnS:SituationDescription*	O
Request	*DnS:SituationDescription*	R
PriceValue	*DOLCE:Region*	$\mathbb{R}$
Price	*DnS:Parameter*	$\{\pi \in \mathbb{R} \mid \exists C : (C, \pi) \in T\}$
price	*DnS:satisfies*	$\{(b,t,\pi) \in B \times T \times \mathbb{R} \mid \exists c \in C, U :$ $t = (c,\pi) \wedge b = (C,U) \wedge U(c) = \pi\}$
AtomicBid	*DnS:SituationDescription*	B^A
ANDBid	*DnS:SituationDescription*	$B^\wedge$
XORBid	*DnS:SituationDescription*	$B^\oplus$
BundleBid	*DnS:SituationDescription*	B
satisfiesBid	*DnS:satisfies*	$\{(b,t) \in B \times T'\}$

Table 6.2: Correspondence of Core Ontology of Bids and the Abstract Market Model. A row in the table with the *ontology entity* ϕ, the *DOLCE concept* ψ and a set of the abstract model E should be understood as follows: $\phi \sqsubseteq \psi$ and there exists an interpretation $\mathcal{I}$ such that $\phi^\mathcal{I} = E$ holds.

sents a set of trades that are acceptable to a requester or provider. For modeling bids we apply the pattern Descriptions & Situations again. In line with the structure of the previous section, we first define how to specify trades as *DnS:Situations* in Section 6.3.1. A trade captures one possible transaction in the market and defines exactly the objects and services to be exchanged and their concrete configuration. Based on this definition, we introduce the specification of *Bids*, which can be seen as *DnS:SituationDescriptions* that provide views on the set of trades (Section 6.3.2). Finally in Section 6.3.3, a method for evaluating bids is presented. The correspondence between abstract model and the ontology introduced in the following is illustrated in Table 6.2.

6.3.1 Specification of Trades

As defined in Section 5.2, a (bilateral) *trade* $t_{ij} = (c, \pi)$ is a potential transaction between two parties i and j where agent i buys a concrete configuration c of an object or service from agent j for a certain amount of money π. We model this by introducing a special *DnS:Situation* called *TradeSituation* which extends the *CPO:Configuration* (Axiom (A40)). Trades might contain two classes of products: the class of *Goods* as specialization of *DOLCE:Endurant* and the class *Service* as specialization of *OoP:Activity* (Axiom (A38) and (A39)). Since these *Services* or *Goods* are multi-attributive they have to refer to a *CPO:Configuration* that defines the values of the different properties of these products. A concrete *TradeSituation* should refer to exactly one *CPO:Configuration* and could specify the corresponding price π which is

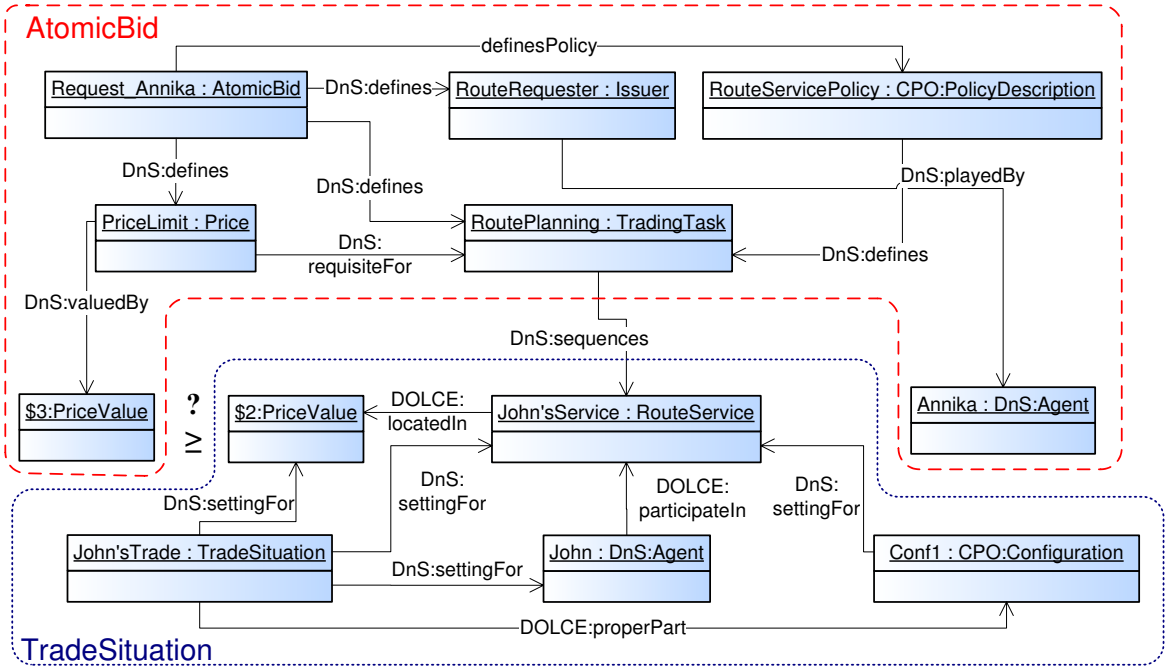

Figure 6.8: Example for a *TradeSituation* and *AtomicBid*. The parts of the diagram are successively introduced in Sections 6.3.1–6.3.3.

modeled via the *DOLCE:Region PriceValue* (Axiom (A41)). Moreover, at least one *DnS:Agent* has to be part of the *TradeSituation* (Axiom (A40)). Note that this formalization does not require to specify both participants – i and j – of a trade, since this information is usually not needed in the bid evaluation process.

(A38) $Service \sqsubseteq OoP{:}Activity \sqcap \exists DnS{:}definedBy.CPO{:}Configuration$

(A39) $Good \sqsubseteq DOLCE{:}Endurant \sqcap \exists DnS{:}definedBy.CPO{:}Configuration$

(A40) $TradeSituation \sqsubseteq DnS{:}Situation \sqcap \exists DnS{:}settingFor.(Service \sqcup Good) \sqcap =_1 DOLCE{:}part.CPO{:}Configuration \sqcap =_1 DnS{:}settingFor.PriceValue \sqcap \exists DnS{:}settingFor.DnS{:}Agent \sqcap \leq_2 DnS{:}settingFor.DnS{:}Agent$

(A41) $PriceValue \sqsubseteq DOLCE{:}Region$

The lower part of Figure 6.8 illustrates the specification of a *TradeSituation*–called *John'sTrade* – by means of an example: *John* provides the route planning service *John'sService* to a price of $2 per invocation. Moreover, John provides a certain configuration *Conf1*. The specification of *Conf1* is omitted in Figure 6.8, since an example for modeling *Configurations* is already given in Section 6.2. Since the price is explicitly modeled as a property of the service, for each additional configuration John wants to provide, a new *TradeSituation* instance has to be introduced. Therefore, enumeration based approaches are only feasible for very low number of

configurations.[10] In order to avoid such enumerations, the concept of *Bid* is presented in the following section.

6.3.2 Specification of Bids

Not all trades T that are possible in a market are favorable for an agent. According to Engel et al. [EWL06], a bid expresses the willingness to participate in trades. We thus model a bid as *DnS:SituationDescription* that *DnS:classifies* exactly those *TradeSituations* $T' \subseteq T$ in which the *Issuer* of a bid is willing to participate. In such *BidDescriptions Goods* and *Services* of a concrete *TradeSituation* play the role of *TradingObjects* and *TradingTask*, respectively. In order to implement matching in a market, one has to define what entities can be *DnS:classifiedBy* a *TradingObject* or *TradingTask*, e.g. that a *RoutePlanningTask DnS:sequences* only *RoutePlanningServices*. Moreover, the description defines a *DnS:Parameter Price* that constrains these *TradingObjects* and *TradingTasks*. This *Price* can be defined explicitly for each service configuration or implicitly by means of *CPO:PolicyDescriptions* as introduced in Section 6.2.2. In the Core Ontology of Bids, we reuse the same idea by introducing the concept *AtomicBid* as follows:

(A42) $\mathit{AtomicBid} \sqsubseteq \mathit{DnS:SituationDescription} \sqcap \exists \mathit{DnS:defines}.(\mathit{TradingObject} \sqcup \mathit{TradingTask}) \sqcap =_1 \mathit{DnS:defines.Price} \sqcap \forall \mathit{CPO:definesPolicy}.(\mathit{CPO:PolicyDescription} \sqcap \mathit{CPO:PolicyCollection})$

(A43) $\mathit{definesPolicy} \sqsubseteq \mathit{DnS:expandedBy}$

(A44) $\mathit{TradingObject} \sqsubseteq \mathit{CPO:PolicyObject} \sqcap \exists \mathit{DnS:playedBy}.\mathit{DOLCE:Endurant}$

(A45) $\mathit{TradingTask} \sqsubseteq \mathit{CPO:PolicyTask} \sqcap \exists \mathit{DnS:sequences}.\mathit{DOLCE:Perdurant}$

(A46) $\mathit{Price} \sqsubseteq \mathit{DnS:Parameter} \sqcap =_1 \mathit{DnS:requisiteFor}.(\mathit{TradingObject} \sqcup \mathit{TradingTask}) \sqcap \forall \mathit{DnS:valuedBy}.\mathit{PriceValue}$

(A47) $\mathit{Issuer} \sqsubseteq \mathit{DnS:FunctionalRole} \sqcap \exists \mathit{DnS:playedBy}.\mathit{DnS:Agent}$

According to Axiom (A48), a *Price* could represent a maximal price (*MaxPrice*) or a minimal price (*MinPrice*). As formalized in Axiom (A49) and (A50), we denote an *AtomicBid* with minimal price as *Offer* (Definition 5.4) and an *AtomicBid* with maximal price as *Request* (Definition 5.5). That means *Offers* classify *TradeSituations* where the property *PriceValue* is above a certain threshold defined via a pricing policy, and thus they implement $T'_j = \{(c,\pi) \in T_j | \pi \geq U_j^P(c)\}$. Analogously, *Requests* classify *TradeSituations* where *PriceValue* is below a threshold defined by a scoring policy, i.e. $T'_i = \{(c,\pi) \in T_i | \pi \leq U_i^S(c,k)\}$.

[10] In addition, this approach is imprecise from an ontological point of view, since a price is not an inherent quality of a product that can be observed, but might depend on the context and other factors.

(A48) $Price \sqsupseteq MinPrice \sqcup MaxPrice$

(A49) $Offer \sqsubseteq AtomicBid \sqcap \exists DnS{:}defines.MinPrice$

(A50) $Request \sqsubseteq AtomicBid \sqcap \exists DnS{:}defines.MaxPrice$

Some market mechanisms support more complex bid specifications beyond the simple case of *AtomicBids*. In Section 5.2, combinatorial bids that enable expressing superadditive as well as subadditive prices for a bundle of products have been introduced. Superadditivity is modeled by introducing *ANDBids* and subadditivity by means of *XORBids*. *ORBids* are not necessarily required in terms of expressivity,[11] but in many cases enable more compact bids representation. Intuitively, *ANDBids* are bids on several products where one would like to have all of them or none. In case of *XORBids*, exactly one product should be allocated, whereas *ORBids* correspond to a set of independent bids. As formalized in Axiom (A53), (A54) and (A53), *ANDBids*, *XORBids* and *ORBids* are specialization of *BundleBid* which are all *Bids* that consist of exactly two other *Bids* (Axiom (A51)). A *Bid* represents the superconcept of *AtomicBids* and *BundleBids*. While *ANDBids* have to contain a *Price* attached to each bundle, no *Prices* can be attached to *XORBids* and *ORBids*, since in this context only the *Prices* of the *AtomicBids* are relevant. Note that since each *BundleBid* has to contain exactly two *Bids*, all *BundleBids* have to terminate solely with *AtomicBids* in a consistent knowledge base (possibly after an arbitrary number of nested *BundleBids*). The axioms below formalize combinatorial bids.

(A51) $BundleBid \sqsubseteq DnS{:}SituationDescription \sqcap =_2 consistsOf.Bid$

(A52) $consistsOf \sqsubseteq DnS{:}expandedBy$

(A53) $ANDBid \sqsubseteq BundleBid \sqcap =_1 andRelated1.(AtomicBid \sqcup ANDBid) \sqcap$
$=_1 andRelated2.(AtomicBid \sqcup ANDBid) \sqcap$
$\exists DnS{:}defines.Price \sqcap$
$\forall definesPolicy.(CPO{:}PolicyDescription \sqcap$
$CPO{:}PolicyCollection)$

(A54) $XORBid \sqsubseteq BundleBid \sqcap =_1 xorRelated1.Bid \sqcap =_1 xorRelated2.Bid$

(A55) $ORBid \sqsubseteq BundleBid \sqcap =_1 orRelated1.Bid \sqcap =_1 orRelated2.Bid$

(A56) $Bid \equiv AtomicBid \sqcup BundleBid$

The relations *andRelated1* and *andRelated2* as well as *xorRelated1* and *xorRelated2* are all modeled as subproperties of *consistsOf*. As done in Section 6.2.3 for the concept *PolicyCollection*, we fix the number of *Bids* in a *BundleBid* by explicitly introducing two *consistsOf*-relations. This technique allows us to avoid universal quantification in rule bodies which is not supported by our rule language. Due to the fact that bundles can be nested, an arbitrary number of *AtomicBids* can be combined.

[11] Recall that all possible valuations can be represented by XOR-bids [Nis00, Prop. 3.2].

A simple *AtomicBid* is exemplified in the upper part of Figure 6.8. Annika needs a service for a route planning task. Therefore, she instantiates *AtomicBid* and *DnS:defines* a *TradingTask* called *RoutePlanning*. Her willingness to pay is specified implicitly via her policy *RouteServicePolicy*. That means the *DnS:Parameter Price* is *DnS:valuedBy* a *PriceValue* that has to be calculated with respect to a concrete *TradeSituation*. This evaluation of a bid is discussed in Section 6.3.3.

6.3.3 Bid evaluation

Bids are *DnS:SituationDescriptions* that select *TradeSituations* that fulfill the specified requirements. Requirements are expressed via *PolicyDescriptions*. Therefore, evaluation of *Bids* can be largely reduced to policy evaluation. Rule (R14) determines the *PriceValue* p of a *AtomicBid* b with respect to a concrete *TradeSituation* t using the predicate *overallDegree* which has been introduced in Section 6.2.2. Since in the case goal policies are violated the *overallDegree*-predicate is evaluated to $-\infty$, we need to distinguish two cases: If the *Bid* represents a *Request*, a score of '$-\infty$' is correct as this leads to a final rejection (Axiom (R14)). If we deal with an *Offer*, a price of $-\infty$ is obviously not what we want to express. We rather want to express a price of ∞ in order to make sure that a certain *TradeSituation* is not acceptable (Axiom (R15) and (R16)).

(R14) $price(b,t,p) \leftarrow Request(b), CPO{:}PolicyDescription(d),$
$definesPolicy(b,d), TradeSituation(t), DnS{:}settingFor(t,c),$
$CPO{:}Configuration(c), overallDegree(c,d,p)$

(R15) $price(b,t,p) \leftarrow Offer(b), CPO{:}PolicyDescription(d),$
$definesPolicy(b,d), TradeSituation(t), DnS{:}settingFor(t,c),$
$CPO{:}Configuration(c), overallDegree(c,d,p),$
$swrlb{:}notEqual(p, "-\infty")$

(R16) $price(b,t,p) \leftarrow Offer(b), CPO{:}PolicyDescription(d),$
$definesPolicy(b,d), TradeSituation(t), DnS{:}settingFor(t,c),$
$CPO{:}Configuration(c), overallDegree(c,d,r),$
$swrlb{:}equal(pr, "-\infty"), assign(p, "\infty")$

For *BundleBids*, we apply Rule (R14) for each *AtomicBid* contained in the bundle. In case of *XORBids* only one *Bid* in the bundle has to be fulfilled. We thus evaluate the *TradeSituation* with each contained *Bid* separately and then determine the price of the *AtomicBid* that is most suitable. Rule (R17) captures this in a recursive manner.

(R17) $price(b,t,p) \leftarrow XORBid(b), xorRelated1(b,b1), price(b1,t,p1),$
$xorRelated2(b,b2), price(b2,t,p2), swrlb{:}max(p,p1,p2)$

Similarly, we define the evaluation rule for *ORBids*. In Definition 5.7 the price of an OR-bid is defined as the sum of the contained *Bids*.

(R18) $price(b,t,p) \leftarrow ORBid(b), orRelated1(b,b1), price(b1,t,p1),$
$orRelated2(b,b2), price(b2,t,p2), swrlb{:}add(p,p1,p2)$

For calculating the price of an *ANDBid* only policies attached to the *ANDBid* itself are considered. This is realized by adapting Rule (R14) as follows:

(R19) $price(b,t,p) \leftarrow ANDBid(b), CPO{:}PolicyDescription(d),$
$definesPolicy(b,d), TradeSituation(t), DnS{:}settingFor(t,c),$
$CPO{:}Configuration(c), overallDegree(c,d,p)$

After introducing the calculation of a bid's *PriceValue*, we can define the *satisfiesBid*-relation that determines if a certain *TradeSituation* is acceptable according to a *Bid*. For the case where *Services* are traded, the following rule checks whether the right service is provided in the *TradeSituation* and whether the price is in an acceptable range (which is defined by the policy). For comparing the *TradingTask*, we use the built-in *subsumes*, which has already been used in Rule (R4). Thereby, we make sure that the provided *Service* fulfills the same purpose as the *Service* sequenced by the *TradingTask*. As already discussed in Section 6.2.1, a meta-modeling approach is required where *Services* are seen as concepts as well as individuals. Moreover, due to price monotonicity requests also include trades with prices that are cheaper as desired and offers include trades with a higher price. Rules (R20)-(R21) ensure that the *TradeSituation* provides the right *TradingTask* and the price is in a valid range.

(R20) $satisfiesBid(b,t) \leftarrow Request(b), TradeSituation(t), DnS{:}defines(b,o),$
$TradingTask(o), DnS{:}sequences(o,e), Service(e),$
$DnS{:}settingFor(t,d), Service(d), subsumes(d,e),$
$price(b,t,pb), MaxPrice(pb), DnS{:}settingFor(t,pt),$
$swrlb{:}lessThanOrEqual(pt,pb)$

(R21) $satisfiesBid(b,t) \leftarrow Offer(b), TradeSituation(t), DnS{:}defines(b,o),$
$TradingTask(o), DnS{:}sequences(o,e), Service(e),$
$DnS{:}settingFor(t,d), Service(d), subsumes(d,e),$
$price(b,t,pb), MinPrice(pb), DnS{:}settingFor(t,pt),$
$swrlb{:}greaterThanOrEqual(pt,pb)$

To illustrate this approach, we come back to the example in Figure 6.8.

Example 6.8 *We are interested if the* TradeSituation *John'sTrade illustrated in Figure 6.8 is relevant for Annika's bid (*Request_Annika*). In order to determine the maximal price Annika is willing to pay for the* Configuration *Conf1 provided by John, we use Rule* (R14)*. Assume the result of this evaluation step is a* MaxPrice *of* $3*. For checking if John provides the right service, we assume the following definition:* RoutePlanning $\sqsubseteq$ *OoP:Task* $\sqcap \forall$*DnS:sequences.RouteService. Since John provides exactly this type of service for* $2*, the* subsumes*-predicate as well as the* swrlb:lessThanOrEqual*-predicate in Rule* (R20) *evaluate to true and the* TradeSituation *satisfies the Bid.*

How the Core Ontology of Bids can be used in the contracting process is outlined in Section 7.1.

6.4 Core Contract Ontology (CCO)

As outlined in Section 2.1.2, the concept of service customization enables the same service to be offered at different service levels for different prices. Usually a specification of the service levels agreed upon is called a *service level agreement* (SLA) [LKD+03]. According to Definition 5.9, we call a legally binding specification of such service level agreements together with additional obligations that result from the contracting process (such as paying a certain price as compensation) a Web service contract [HF05]. This corresponds to the definition given by Reinecke [RDS89], where "a contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights."

Due to the cross-organizational and collaborative nature of business processes, which are supported by today's service-oriented architectures, contracts have become a key governance mechanism regulating business interactions. In spite of their importance, today's enterprises still treat contracts merely as paper documents regulating the case where something goes wrong and without linking them to the cross-organizational interactions that they govern. Dealing with contract management task such as contract execution and monitoring is very cumbersome, time-consuming, inefficient and thus expensive. Therefore, a more holistic approach to contract handling is required that supports the following features [MG05]:

- formal contract languages that provide open, transparent and up-to-date information about contract data and the status of a contract;
- mechanisms that use information from contracts as a basis for monitoring of contract compliance and subsequent notifications and enforcement measures;
- mechanisms and tools that support management of the entire contract life cycle, including contract formation, contract execution and contract monitoring;
- tools that support personnel in meeting their obligations that arise from the contract.

That means, formal representation of contracts is crucial for enabling more efficient contract management. In a service-oriented architecture, a formalized Web service contract can be directly used to govern the business interactions executed via Web service invocations. As specified in Requirements (*R7*) and (*R11*), Web service contracts have to be found automatically and have to be legally reliable. This

requires, on the one hand, a formal machine-interpretable language that enables automated contract formation, execution and compliance checking; and on the other hand, the expressivity to specify all legally required clauses. This ensures that any violation of pre-agreed service levels results in a penalty for the party that is responsible for the violation.

Over the last decades a lot of work has been devoted to the formalization of contracts and legal norms in general – mainly in the areas Artificial Intelligence, Computer Science and Philosophical Logic [DS97]. However, up to now formalization of legal contracts has been restricted either to relatively simple contractual clauses expressed via a standard syntax [Mil95, GBW+98, CM01, GP03, AG03, Glo06, Gov05] which lack standardized declarative semantics required to ensure interoperability, or they rely on very complex logical formalism [Hag96, TT98, Ser01] which are not computational tractable (Requirement (*R10*)) or lack any support for inter-organizational interactions (Requirement (*R6*)). Since the trade-off between expressivity and tractability cannot be easily resolved, we focus on semi-automated approach where a natural language umbrella contract is manually closed with different service providers and only some of the terms are fully formalized. In fact, to meet Requirement (*R7*) only obligations that have to be dynamically settled during the contracting process or that should be monitored automatically after contract execution have to be formalized. This eases the formalization task and allows the usage of more simple, computationally tractable logical formalisms.

In this section, we propose the use of ontology languages for formally representing Web service contracts. As already outlined in Section 2.4 and 6.2, ontologies come with a logical calculus that enables representing information in a formal and standardized way. Thereby, ontologies provide interoperability (Requirement (*R6*)), flexibility (Requirement (*R8*)) and extensive tool support. These advantages carry over to contract specification and management. By providing an open, transparent and interoperable view on contractual data, ontology-based contract representation enables a tight integration of up-to-date contractual information with the collaborative business interactions they govern. This means, the machine-interpretable contractual information can be easily accessed by contracting and contract monitoring tools, and it can be easily shared with business partners. In addition, standard tools supporting the logical formalisms of the ontology can be used to perform sophisticated contract monitoring that involves logical inferencing.

The different parts of the Core Contract Ontology are introduced in this section as follows: In Section 6.4.1, the idea of a semi-automated approach to contracting and contract monitoring is presented. In this context, an informal umbrella contract is closed, which constitutes the environment that enables automated contracting of formal individual contracts on a per-invocation-basis. The formalization of these individual contracts as specializations of *DnS:SituationDescriptions* is then presented in Section 6.4.2. In order to support the settlement phase, Web service monitoring information has to be formally represented. How this can be realized by means of a *DnS:Situation* is outlined in Section 6.4.3. Finally, in Section 6.4.4 modeling primitives for evaluating contracts are presented. This requires knowledge how specific contractual clauses have to be interpreted. Since this knowledge is usually available only as tacit knowledge of legally educated persons, it also has to be externalized into a machine readable and executable form.

6.4.1 Semi-Automated Contracting and Monitoring

Full automation of the contracting lifecycle has so far been investigated only for very simple contracts. Semi-automated contracting can be seen as an approach, in which a contract is composed of two separate parts: an *umbrella agreement* which is directly negotiated by human beings and an *individual contract* automatically negotiated and closed by software agents.

Umbrella Contract

The umbrella agreement is presently necessary to define the legal conditions under which software agents can enter into binding agreements as not all jurisdictions acknowledge negotiating and contracting by software agents. The service requestors agree on an umbrella agreement with several Web service providers. The umbrella agreement will therefore define the framework for several software agents to negotiate the individual contracts. The umbrella agreement regulates the following issues:

- the beginning of the contractual relations between all parties, how long the umbrella agreement is valid and how and when it can be terminated;
- the types of Web services to be negotiated;
- the timeframe for negotiations (preferably 24/7);
- auxiliary duties of the parties such as maintenance or the obligation to treat customer information confidentially;

These clauses form the continuous contractual relations between the parties and span more than one Web service invocation. In particular, they are not customizable and not negotiable. Often each umbrella contract closed by a requester with different providers contains the same clauses. Differentiation between the providers is realized in the individual contract, which captures content such as price, license type, payment terms, response time guarantees, etc.

Individual Contract

In an individual contract most aspects are customizable. Requests and offers specified during the contracting phase can be seen as contract templates, where for each attribute several values are possible. Formally, offers O and requests R define the sets of acceptable trades T_j' and T_i' (see Definition 5.4 and 5.5), which correspond to contract templates. In the matching and allocation phase, one value has to be chosen for each attribute and a contract can be concluded. That means a trade $t \in T_i \cap T_j$ acceptable to both parties has to be chosen. The configuration identified in the trade $t = (c, \pi)$ is then used in the contract formation process that generates the appropriate provider and customer obligations of the contract.

In the following, we illustrate the general content of an individual contract for a typical information service, such as a route planning service or credit information service. In this context, we discuss for different content categories whether certain clauses should be in the individual contract.

Scope of Agreement (§1). Although the general type of trading object might be defined in the umbrella agreement, it is important that the agents have some flexibility to find agreements. For example, let an umbrella agreement define the scope of the agreement as Web services providing 'Credit Information' or 'Route Information'. Then in the individual contract the exact type of information (e.g. 'Business Background Information' or 'German Route Information') can be automatically determined in the matching process.

Provider Obligation (§2). In this category obligations of the provider are defined that can be customized for each invocation of the service. This is thus the main category where service level agreements are contained. For example, it is usually price relevant how old the credit or route information is. The software agents might therefore negotiate the update periods of the provided information. Of course, also other quality of service guarantees, such as maximal response time or the period in which errors have to be corrected, can be specified here.

Use of Information (§3). The individual contract will specify how the customer may use the information. This category may also involve obligations that restrict the use of information. For example, a contract clause specifying such use may grant a transferable license to use the information or a non-transferable license and define further to what extent the customer may use the credit information within its company or towards third parties.

Warranties and Liabilities (§4). Since warranties and liabilities directly influence the costs of a provider, they are highly price relevant. We let the software agents negotiate about the warranty level but not about the legal obligations resulting from a breach of warranty. The legal complexity, including the restrictions by law to contract out certain statutory warranties and liabilities, does not allow full automation at present. For example when customizing warranty levels following scheme can be applied: (*1*) The service provider does not give any warranty as to the accuracy of the information. (2) The service provider does not warrant the accuracy of the information, but warrants that it has put the information together with utmost care and state-of-the-art-methods. (*3*) The service provider guarantees that the information is 100% correct.

Delivery Time (§5). The delivery of the information can be automatically customized in a way that the service has to be provided immediately after the individual contract is concluded or at a later, negotiated time. The legal consequences of non- or late delivery however are set forth in the umbrella agreement.

Prices and Payment Terms (§6). Finally, the prices and payment terms have to be specified, which can be mostly seen as customer obligations. While the parties define the details of invoicing in the umbrella agreement, some parameters, such as price for the individual Web service or the due date of the payment, can be dynamically fixed.

After closing a contract, in the settlement phase the participants monitor whether the contractual duties are fulfilled. However, full automation of the monitoring

Ontology entity	DOLCE alignment	Abstract model
ContractDescription	*DnS:SituationDescription*	Γ
Obligation	*DnS:SituationDescription*	γ
ContractParty	*DnS:FunctionalRole*	$I \cup J$
Customer	*DnS:FunctionalRole*	I
CustomerObligation	*DnS:SituationDescription*	$\{\gamma \in \Gamma \mid \gamma = (i, G_\Phi) \wedge i \in I\}$
Provider	*DnS:FunctionalRole*	J
ProviderObligation	*DnS:SituationDescription*	$\{\gamma \in \Gamma \mid \gamma = (j, G_\Phi) \wedge j \in J\}$
satisfiesContract	*DnS:satisfies*	$\{(c, \Gamma) \mid \varrho(\Gamma, c) = 1\}$

Table 6.3: Correspondence of Core Contract Ontology and Abstract Contract Model. A row in the table with the *ontology entity* ϕ, the *DOLCE concept* ψ and a set of the abstract model E should be understood as follows: $\phi \sqsubseteq \psi$ and there is an interpretation $\mathcal{I}$ such that $\phi^{\mathcal{I}} = E$ holds.

step is impossible since assessing the quality of a Web service can only be done by taking external and not quantifiable factors into account. Nevertheless, some aspects can be monitored by the system automatically. For instance, it can be assured that a contracted service is provided at all and in the negotiated timeframe. For this purpose, all clauses that are relevant to evaluate whether the contract is met also have to be represented formally (even if they are not customizable).

6.4.2 Contract Representation

In this section, we show how contract information can be represented by reusing the Core Policy Ontology. In doing so, goal policies are used to represent obligations and permissions in a contract. The correspondence between the abstract model introduced in Section 5.3.3 and the Core Contract Ontology is illustrated by Table 6.3. After introducing this general contract ontology, we exemplify their usage by modeling the content of the individual contract identified in Section 6.4.1.

General Contract Ontology

As defined above, a contract can be seen as a set of obligations and rights that are binding for all parties. In the case of Web services we restrict ourselves to contracts between exactly two parties, namely Provider and Customer. We model this by introducing a *ContractDescription* as a *DnS:SituationDescription* containing only *Obligations* and *Permissions* (Axiom (A57)). *Obligations* and *Permissions* are *CPO:PolicyDescriptions* that represents obligations and permission for *ContractParties*. An *Obligation* is a *CPO:PolicyDescription* where the *DnS:attitudeTowards*-relation is refined to *DnS:obligedTo* (Rule (R22)) and a *Permission* a *CPO:PolicyDescription* where it is refined to *DnS:rightTo* (Rule (R23)). A *ContractParty* is an active *PolicyObject* that is played by *DnS:Agents*. This can be formalized using the following axioms and rules:

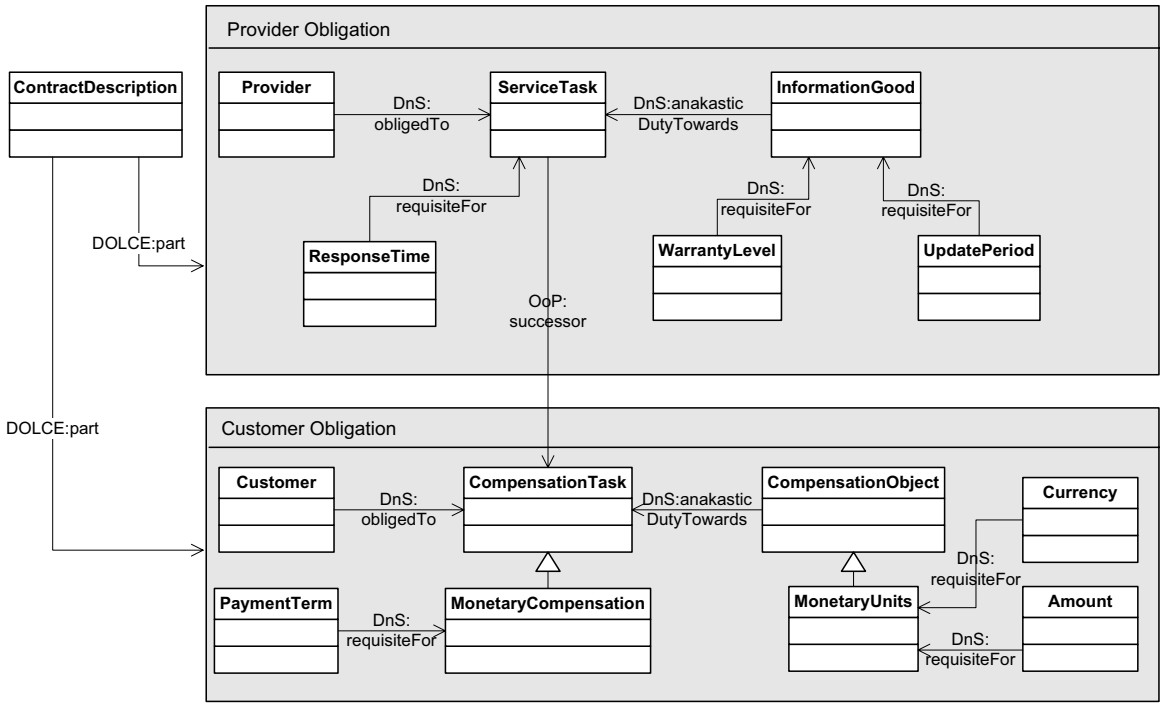

Figure 6.9: Representation of the Core Contract Ontology. Note that plotting UML classes within an *Obligation*-class illustrates a *DnS:defines*-relation between the *Obligation* and the contained classes.

(A57) $$\mathit{ContractDescription} \equiv \mathit{DnS{:}SituationDescription} \sqcap \forall \mathit{DOLCE{:}part}.(\mathit{Obligation} \sqcup \mathit{Permission})$$

(A58) $$\mathit{ContractParty} \sqsubseteq \mathit{PolicyObject} \sqcap \forall \mathit{DnS{:}playedBy}.\mathit{DnS{:}Agent}$$

(R22) $$\mathit{Obligation}(x) \leftarrow \mathit{PolicyDescription}(x), \mathit{DnS{:}defines}(x,y), \mathit{ContractParty}(y), \mathit{DnS{:}defines}(x,z), \mathit{PolicyTask}(z), \mathit{DnS{:}obligedTo}(y,z)$$

(R23) $$\mathit{Permission}(x) \leftarrow \mathit{PolicyDescription}(x), \mathit{DnS{:}defines}(x,y), \mathit{ContractParty}(y), \mathit{DnS{:}defines}(x,z), \mathit{PolicyTask}(z), \mathit{DnS{:}rightTo}(y,z)$$

The definition of *ContractDescription* and *Obligation* corresponds exactly to Definition 5.9 in the abstract model.[12] A *ContractDescription* Γ defines a set of *Obligations* γ (Axiom (A57)), where each *Obligation* specifies a *ContractParty* $y \in I \cup J$ and a *PolicyDescription* G_Φ.

Consequently, as depicted in Figure 6.9, the most elementary contract about purchasing Web services in exchange for money results in two simple Obligations:

[12] In contrast to the Core Contract Ontology, permissions are not contained in the abstract model since they are currently not used in the contract monitoring process.

Provider Obligation. A *ProviderObligation* specifies that the provider is obliged to make certain functionality accessible to the customer (Axiom (A59)). This functionality is represented by a *CPO:PolicyTask ServiceTask*, which is played by a *COS:WebService* in a *DnS:Situation* (Axiom (A61)). In addition, an (active) *CPO:PolicyObject Provider* is introduced that is *DnS:obligedTo* provide the *ServiceTask* (Axiom (A60)). A second (passive) *CPO:PolicyObject InformationGood* is used to represent the information that has to be returned by the *COS:WebService* playing the *ServiceTask* (Axiom A62). Note that the distinction between *ServiceTasks* and *InformationGood* allows modeling the functionality of a service using either explicit or implicit capability representation [SPAS03]. This enables our contract ontology to support major efforts striving for semantic Web service descriptions such as WSMO [DKL$^+$05], OWL-S [SPAS03] and WSDL-S [POSV04].

(A59) $ProviderObligation \sqsubseteq Obligation \sqcap \exists DnS{:}defines.Provider$

(A60) $Provider \sqsubseteq ContractParty \sqcap \exists DnS{:}obligedTo.ServiceTask$

(A61) $ServiceTask \sqsubseteq PolicyTask \sqcap \forall DnS{:}sequences.COS{:}WebService$

(A62) $InformationGood \sqsubseteq PolicyObject \sqcap \forall DnS{:}playedBy.OIO{:}InformationObject$

Customer Obligation. A *CustomerObligation* specifies that the customer is obliged to compensate the provider for using the Web service (Axiom (A63)). This activity is called *CompensationTask* and mostly involves the transfer of a certain amount of money. To define a *CompensationTask* the *CPO:PolicyTask* is specialized to a *CompensationTask* (Axiom A65). A *Customer* is a *ContractParty* that is obliged to carry out a *CompensationTask* (Axiom A64). Moreover, a *CompensationTask* may involve a *CPO:PolicyObject CompensationObject*, which refers to a passive physical or social entity (*DOLCE:NonAgentiveSocialObject* or *DOLCE:NonAgentivePhysicalObject*) such as money or a patent (Axiom (A66)).

(A63) $CustomerObligation \sqsubseteq Obligation \sqcap \exists DnS{:}defines.Customer \sqcap DnS{:}defines.CompensationTask$

(A64) $Customer \sqsubseteq ContractParty \sqcap \exists DnS{:}obligedTo.CompensationTask$

(A65) $CompensationTask \sqsubseteq CPO{:}PolicyTask \sqcap \forall DnS{:}anakasticDutyTowards^{-}.CompensationObject$

(A66) $CompensationObject \sqsubseteq CPO{:}PolicyTask \sqcap \exists.DnS{:}anakasticDutyTowards.CompensationTask \sqcap \forall DnS{:}playedBy.(DOLCE{:}NonAgentiveSocialObject \sqcup DOLCE{:}NonAgentivePhysicalObject)$

Usually contracts also specify in which sequence obligations have to be fulfilled and rights are obtained. In the basic contract outlined above, the *ServiceTask* has

to be executed before the *CompensationTask*. Hence, means for representing sequences of *OoP:Tasks* are required. We reuse the Ontology of Plans which provides primitives for modeling complex processes, e.g. *Sequential Tasks*, *Parallel Tasks*, *Loop Tasks*, etc. In this context, the primary ordering relation for *OoP:Tasks* are *OoP:directSuccessor* and its transitive version *OoP:successor*.

As illustrated in Figure 6.9, concrete obligations are expressed via policies specifying *CPO:Attribute* for *ServiceTask* and *InformationGood* or *CompensationTask* and *MonetaryUnit*, respectively. How this can be realized for the individual contract clauses identified in Section 6.4.1 is shown in the next section.

Individual Contract Clauses

As discussed above, a contract imposes further obligations and permissions that have to be fulfilled by the contractors. These obligations and permissions are modeled within a *CPO:PolicyDescription* by introducing specialized *CPO:Attribute* concepts and specifying the allowed *CPO:AttributeValue* for this *CPO:Attribute*. In the following, we briefly discuss some examples how the obligations that have to be defined in an individual contract can be formalized. Methodologically this is realized by transforming a natural language contractual clause into a formalized goal policy. However, note that this not an exhaustive enumeration. Depending on the service types and scenarios a wide range of different *Attributes* are possible.

Provider Obligation (§2) As discussed on page 124, in this category service levels can be specified a provider has to meet. We exemplify this by considering the *CPO:Attribute UpdatePeriod* which is warranted by the provider. A legal text negotiated by human beings could read as follows:

> *"The Provider warrants that it reviews and, if necessary, updates Route Planning Information/Credit Information every month."*

Since the timeliness is a property of the provided *InformationGood*, we introduce *UpdatePeriod* as a subclass of *CPO:Attribute*. *UpdatePeriod* constraints the set of allowed *CPO:AttributeValues UpdatePeriodValue*. This is captured by the following axiom:

$$\begin{aligned}(\text{A67}) \qquad \mathit{UpdatePeriod} \sqsubseteq \mathit{CPO{:}Attribute} \sqcap\ & \\ \exists \mathit{DnS{:}requisiteFor.InformationGood} \sqcap\ & \\ \forall \mathit{DnS{:}requisiteFor.InformationGood} \sqcap\ & \\ \exists \mathit{DnS{:}valuedBy.UpdatePeriodValue}\end{aligned}$$

The *CPO:Attribute UpdatePeriod* is illustrated in Figure 6.9.

Use of information (§3) This category specifies how the customer may use the information. For example, consider licenses that typically regulate how certain information can be used. An agreed legal text could read as follows:

> *"The Provider grants the customer a non-transferable license to use the Credit Information delivered under the terms of this contract. The Customer may freely copy or forward Credit Information within its company. The Customer may not disclose or make the Credit Information otherwise available to third parties without prior consent of the Provider."*

The license specifies if the right to use a certain *InformationGood* is transferable, if the customer may disclose the *InformationGood* within the company ('Disclose within Company') or to external third parties ('Disclose to 3rd Party'). In order to facilitate contract monitoring, we model the right as an *Obligation* that specifies which alternatives are not allowed. This is realized by introducing an additional *CustomerObligation DisclosureObligation* (Axiom (A68)) with the *CompensationTask TransferInformation* (Axiom (A69)) and the *CPO:Attribute AdmissibleParty*. *AdmissibleParty* may take the values 'Not Transferable', 'Disclose within Company' and 'Disclose to 3rd Party' (Axiom (A70)). The following axioms capture this information. Note that the corresponding *Obligation* is omitted in Figure 6.9.

(A68) $DisclosureObligation \sqsubseteq CustomerObligation \sqcap \exists DnS{:}defines.TransferInformation \sqcap \exists DnS{:}defines.AdmissibleParty$

(A69) $TransferInformation \sqsubseteq CompensationTask \sqcap DnS{:}requisites.AdmissibleParty$

(A70) $AdmissibleParty \sqsubseteq CPO{:}Attribute \sqcap \exists DnS{:}requisiteFor.TransferInformation \sqcap \forall DnS{:}requisiteFor.TransferInformation \sqcap =_1 DnS{:}valuedBy.\{$'Not Transferable', 'Disclose within Company', 'Disclose to 3rd Party'$\}$

Warranties and Liabilities (§4) In this category warranty and liability levels can be defined. In legal practice a wide range of different warranty and liability regulations are used. In this example, we consider a very simple approach, where automatically one level from a predefined set of warranty levels can be chosen. The predefined warranty levels are defined in the umbrella contract. In a natural language contract a level can be defined as follows:

> *"The Provider warrants that the credit information is 100% accurate."*

As shown in Figure 6.9, this can be realized by adding a *CPO:Attribute WarrantyLevel* to the *ProviderObligation* which is valued by a *DOLCE:Region* reflecting the three different warranty levels: 'No Warranty', 'Uttermost Care', and 'Full Warranty'. Since the warranty can be considered as a fundamental

property of a *InformationGood*, we model *WarrantyLevel* as a *Attribute* of *InformationGood*.

(A71) $WarrantyLevel \sqsubseteq CPO{:}Attribute \sqcap \exists DnS{:}requisiteFor.InformationGood \sqcap \forall DnS{:}requisiteFor.InformationGood \sqcap =_1 DnS{:}valuedBy.\{`NoWarranty', `UttermostCare', `FullWarranty'\}$

In general, defining standard quality levels in the umbrella contract is a viable way to reduce the complexity of the individual contract, while avoiding complex, undecidable logics.

Delivery Time (§5) In many applications delivery time is a crucial property that heavily influences the prices. It is also a property that often has to be customized dynamically, e.g., in order to adapt the contract to changing Web server load. A natural language clause could be formulated as follows:

> *"The Provider shall deliver the Route Planning/Credit Information within five seconds after conclusion of the contract."*

In the context of Web services, delivery time usually refers to the *response time*, in which the result is return by the service. The *CPO:Attribute ResponseTime* specifies the period in which the *Service Task* has to be executed. Hence, it is modeled as a constraint of *ServiceTask* which is *DnS:valuedBy* an *CPO:AttributeValue ResponseTimeValue*. The approach is illustrated in Figure 6.9 and captured by the following axiom:

(A72) $ResponseTime \sqsubseteq CPO{:}Attribute \sqcap \exists DnS{:}requisiteFor.ServiceTask \sqcap \forall DnS{:}requisiteFor.ServiceTask \sqcap =_1 DnS{:}valuedBy.ResponseTimeValue$

Prices and Payment Terms (§6) Usually the most important aspect regulated in a contract is the price that has to be paid by the customer for invoking the service. Prices of services may change frequently or are even determined dynamically in a negotiation or an auction process (see dynamic pricing mechanisms outlined in Section 2.3.2). For example, a corresponding clause could be simply specified as follows:

> *"The price for the provided route/credit information is EUR 15."*

We have defined a *Customer* as a *ContractParty* that is obliged to an executing a *CompensationTask* (Axiom (A64)). The nature of this compensation is left open and can be defined by constraining the allowed alternatives using policies. For the case in which no compensation is required (e.g. service usage

is free of charge), simply no policies are defined for the *CompensationTask*. For the usual case where a certain amount of money has to be paid, we have specialized *CompensationTask* to *MonetaryCompensation* which requires the specification of the *MonetaryUnits* that have to be transferred from the customer to the provider (Axiom (A73)). *MonetaryUnits* are *CompensationObjects* which specify a certain *Amount* of money in a given *Currency* (Axiom (A74)). The *CPO:Attribute Amount* is valued by exactly one floating point number representing the *AmountValue* (Axiom (A75)) and the *CPO:Attribute Currency* is valued by exactly one *CurrencyValue* (Axiom (A76)). Thus, *CurrencyValue* comprises Euro, Dollar, Yen, etc. This is formalized by the following axioms.

(A73) $MonetaryCompensation \sqsubseteq CompensationTask \sqcap \forall DnS{:}anakasticDutyTowards^{-}.MonetaryUnit \sqcap \exists DnS{:}anakasticDutyTowards^{-}.MonetaryUnit$

(A74) $MonetaryUnits \sqsubseteq CompensationObject \sqcap \exists DnS{:}requisites.Amount \sqcap \exists DnS{:}requisites.Currency$

(A75) $Amount \sqsubseteq CPO{:}Attribute \sqcap =_1 DnS{:}valuedBy.AmountValue$

(A76) $Currency \sqsubseteq CPO{:}Attribute \sqcap =_1 DnS{:}valuedBy.CurrencyValue$

Furthermore, a contract usually contains a *PaymentTerm* that specifies in which timeframe a *MonetaryCompensation* has to take place. We model the *PaymentTerms* as a *CPO:Attribute* constraining *MonetaryCompensation* as shown in Figure 6.9.

(A77) $PaymentTerm \sqsubseteq CPO{:}Attribute \sqcap \exists DnS{:}requisiteFor.MonetaryCompensation \sqcap \forall DnS{:}requisiteFor.CompensationTask \sqcap =_1 DnS{:}valuedBy.DOLCE{:}Temporal\text{-}Region$

All regulations specified above can be extended either by introducing new *CPO:Attributes* within an existing *CPO:PolicyDescription* or by adding further *Obligations* or *Permissions* to the *ContractDescription*. In the following, we exemplify how a simple *ProviderObligation* could be expressed.

Example 6.9 *Assume a credit information service which is obliged to deliver business background information about the company SAP to a requester. The provider guarantees delivery within 30 seconds. This simple* ProviderObligation *can be expressed with the Core Contract Ontology as shown in Figure 6.10. We introduce an instance of* ProviderObligation *called* ProviderObligationX *that* DnS:defines *a Provider X, the* InformationGood

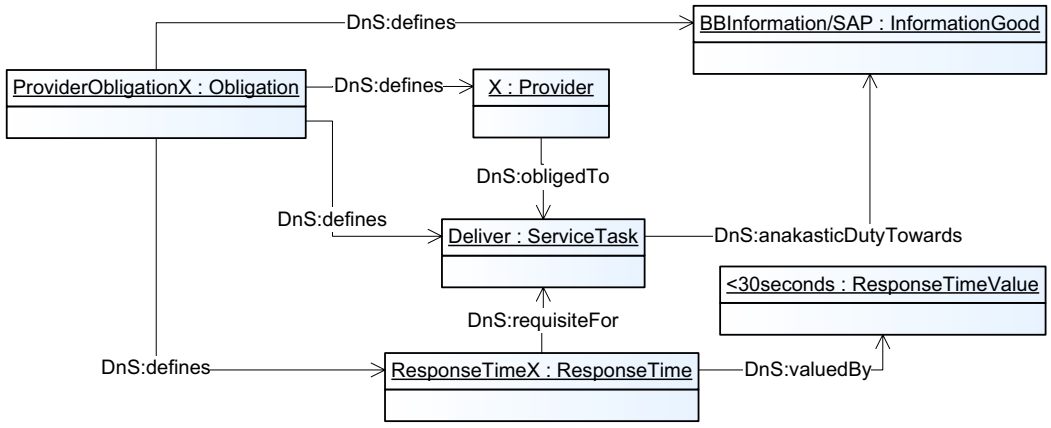

Figure 6.10: Example for representing a *ProviderObligation*.

BBInformation/SAP, a ServiceTask Deliver, and the Attribute ResponseTimeX that has to be valued by a ResponseTimeValue. In our example, this ResponseTimeValue is a DOLCE:Region capturing all time periods below 30 seconds. Since ResponseTime is a constraint on the ServiceTask, we thereby make sure that the delivery has to take place within the 30 second time period.

6.4.3 Representing Monitoring Information

In the previous section, we presented contract information as a collection of *PolicyDescriptions* which are modeled by refining *DnS:SituationDescriptions*. In this section, we extend this approach in order to represent information about the execution of a contract. We call such information *monitoring information* and represent it by means of the *DnS:Situation MonitoringInformation* (Axiom (A78)). *MonitoringInformation* is modeled as specialization of *CPO:Configuration* and represents values of *CPO:Attributes* used in the Web service execution. As defined in Axiom (A15), a *CPO:Configuration* identifies the value of an *CPO:Attribute* belonging to an *DOLCE:Endurant* or *DOLCE:Perdurant*. Since we are dealing only with monitoring Web service invocations, we can specialize our modeling approach. The Core Software Ontology (CSO) and the Core Ontology of Service (COS) [Obe05] introduce the fundamental concepts required for describing software systems. According to [Obe05], the main entities living in the computational domain are *CSO:ComputationalObjects* and *CSO:ComputationalActivities*. *CSO:ComputationalObjects* can be regarded as concrete realization of *CSO:Software* or *CSO:Data*.[13] The execution of *CSO:Software* triggers *CSO:ComputationalActivities* and these *CSO:ComputationalActivities* may involve *CSO:Data*. Rule (R24) and (R25) capture this active and passive aspect by introducing the relations *executes* and *involvedIn*, respectively. Each *MonitoringInformation* instance has to contain at least one *CSO:ComputationalActivity* that is monitored (Axiom (A78)). As for *CPO:Configurations* in general, each *CSO:ComputationalActivity* and *CSO:ComputationalObject* may exhibit certain properties that are captured by *DOLCE:Qualities*.

[13]Note that *CSO:Software* can be seen as a special form of *CSO:Data*, viz., *CSO:Software* $\sqsubseteq$ *CSO:Data*.

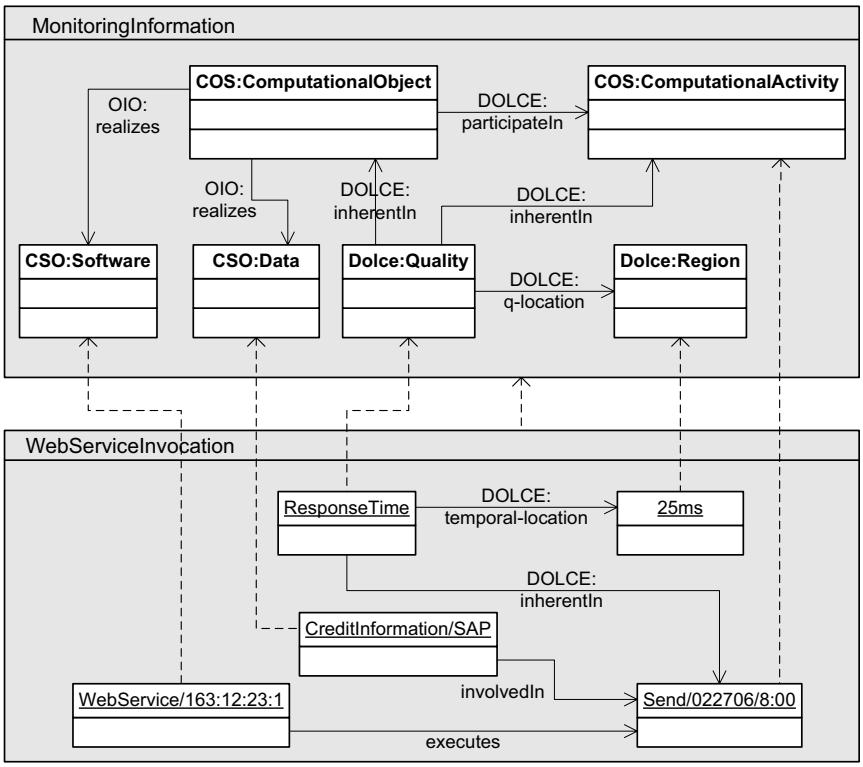

Figure 6.11: Representing *MonitoringInformation* as *DnS:Situation*. Note that plotting UML classes within a *DnS:Situation*-class illustrates a *DnS:settingFor*-relation between the *DnS:Situation* and the contained classes.

$$\begin{aligned}(\text{A78})\quad \mathit{MonitoringInformation} \sqsubseteq\ & \mathit{CPO{:}Configuration} \sqcap \\ & \exists \mathit{DnS{:}settingFor}.\mathit{CSO{:}ComputationalActivity} \sqcap \\ & \forall \mathit{DnS{:}settingFor}.(\mathit{CSO{:}ComputationalActivity} \sqcup \\ & \mathit{CSO{:}ComputationalObject} \sqcup \\ & \mathit{DOLCE{:}Quality} \sqcup \mathit{DOLCE{:}Region})\end{aligned}$$

$$\begin{aligned}(\text{R24})\quad \mathit{executes}(x,y) \leftarrow\ & \mathit{CSO{:}Software}(x), \mathit{OIO{:}expresses}(x,z), \\ & \mathit{OoP{:}Plan}(z), \mathit{DnS{:}defines}(z,t), \mathit{CSO{:}ComputationalTask} \\ & \mathit{DnS{:}sequences}(t,y) \mathit{CSO{:}ComputationalActivity}(y)\end{aligned}$$

$$\begin{aligned}(\text{R25})\quad \mathit{involvedIn}(x,y) \leftarrow\ & \mathit{CSO{:}Data}(x), \mathit{OIO{:}realizedBy}(x,z), \\ & \mathit{DOLCE{:}participantIn}(z,y), \\ & \mathit{CSO{:}ComputationalActivity}(y)\end{aligned}$$

Providing information via a Web service leads to a *CSO:ComputationalActivity* where one party transfers a *CSO:ComputationalObject*, e.g. credit or route information, to another party. In executing this activity various types of monitoring information about the activity itself as well as about participating objects can be mea-

sured or perceived, which are represented as *DOLCE:Qualities* of the corresponding *CSO:ComputationalActivity* or *CSO:ComputationalObject*.

Example 6.10 *Figure 6.11 introduces a concrete example which represents information about a specific Web service invocation as an instance of* MonitoringInformation*. Consider the execution of a* CSO:ComputationalActivity *'Send' carried out on February 27th, 2006 at 8 AM. The activity was executed by a Web service with the IP-address 163:12:23:1 and involved the digital representation of credit information of the company SAP. According to the Core Ontology of Services [Obe05], a* COS:WebService *is a specialization of* CSO:Software *and thus we model 'WebService/163:12:23:1' as an instance of the concept* CSO:Software*, while 'Credit Information/SAP' is modeled as* CSO:Data*. Moreover, the* DOLCE:Quality ResponseTime *of the* Send*-activity is measured and represented by the* DOLCE:Region *'25ms'. Of course, other* DOLCE:Qualities *of the* CSO:ComputationalActivity *as well as the* CSO:ComputationalObject *beyond 'ResponseTime' can be measured and represented in a similar way.*

6.4.4 Contract Monitoring

Having introduced the *CPO:ContractDescription* in Section 6.4.2 and *MonitoringInformation* capturing information about contract execution in Section 6.4.3, we introduce in this section how the compliance of *MonitoringInformation* with respect to concrete *ContractDescription* can be verified. This corresponds to the evaluation of function $\varrho(\Gamma, c)$ specified in the abstract model.

Since *ContractDescriptions* are modeled as a collection of *CPO:PolicyDescriptions* and *MonitoringInformation* as specialization of *CPO:Configuration*, the *satisfiesPolicy*-relation (Rule (R8)) that holds between *PolicyDescriptions* and *DnS:Situations* can be reused in order to check whether the *MonitoringInformation* meets the contractual obligations.

$$\text{(R26)} \quad \mathit{satisfiesObligation}(m,p) \leftarrow \mathit{MonitoringInformation}(m), \mathit{Obligation}(p), \mathit{satisfiesPolicy}(m,p)$$

In order to validate the entire contract, each *Obligation* $\gamma \in \Gamma$ specified in the *ContractDescription* has to be satisfied by the *MonitoringInformation*. If all *Obligations* are valid, the *ContractDescription* is met by the contract execution. The following rule encodes the compliance check for a contract Γ.

$$\text{(R27)} \quad \mathit{satisfiesContract}(m,c) \leftarrow \mathit{MonitoringInformation}(m), \mathit{ContractDescription}(c), \bigwedge_{i=\{1,\ldots,|\Gamma|\}} (\mathit{DOLCE{:}part}(c,p_i), \mathit{satisfiesObligation}(m,p_i))$$

As already discussed in Section 6.2.1, universal quantification would be required to formulate the rule in a way that it supports an arbitrary number of obligations of the contract. Since DL-safe SWRL does not support this construct, maximal number of obligations that have to be verified has to be specified at design time. Although for the most applications this might not be a problem since the obligations can be usually expressed with one customer and one provider obligation, there might be scenarios where predefining the number of obligations is not possible. In this case either a more expressive rule language is required (e.g. CIF/SWRL proposed by [MGP04]) or a contract monitoring tool implementing an iterative algorithm that evaluates the *satisfiesPolicy*-rule for each obligation contained in the contract.

Example 6.11 *In order to illustrate how contracts are evaluated we come back to the* ProviderObligation *introduced in Example 6.9 and to the* MonitoringInformation *outline in Example 6.10. We are now interested whether the* MonitoringInformation *fulfills the* ProviderObligation*. Therefore, the* satisfiesObligation*-predicate can be applied. In doing this, the* satisfiesPolicy*-predicate is used in order to determine if the constraints specified in the contract are meet. This is realized by invoking the corresponding matching-rules for each* Attribute *in the* ProviderObligation*. In our concrete example, we compare the observed 25 seconds with the allowed 30 seconds using a matching rule that defines* DOLCE:TemporalRegions *to be matched with* swrlb:lessThanOrEqual*. We thus get a match and no policy violation in our example.*

In Section 7.2, we provide a more detailed discussion on how the *satisfiesContract* can be applied and adapted.

6.5 Conclusion

In this chapter, we have introduced an ontology framework for Web service markets. In particular, we have presented three novel core ontology modules based on the foundational ontology DOLCE: the Core Policy Ontology, the Core Ontology of Bids and the Core Contract Ontology. Together with the Core Ontology of Services [OLG+06] they constitute the second layer of the ontology framework (see Figure 6.1 on page 98). The framework provides the communication primitives which are required in the market and addresses the language-specific requirements listed in Section 4.2.1. Contracting algorithms that rely on these communication primitives are introduced in the next chapter.

Chapter 7

Ontology-based Contracting and Contract Monitoring

After having introduced a formal, interoperable language for expressing market information in Chapter 6, in this chapter we come back to the Web service contracting and contract monitoring process abstractly introduced in Section 5.3. The goal is to implement algorithms based on the presented ontology framework that meet the mechanism-specific Requirements stated in Section 4.2.2.

The chapter is structured as follows: First, a contracting mechanism is introduced in Section 7.1 featuring semantics-based matching of offers and requests, optimal allocation algorithms, and a contract formation algorithm. After the contracting phase, we move on to the settlement phase in Section 7.2 and introduce a mechanism for contract monitoring based on the Core Contract Ontology.

Most results presented in this chapter are published in conference proceedings and journals. Bits and pieces of the contracting process including the Web service selection algorithm are obtained from [LAGS07], the auction-based allocation is partly covered in [LS06], and the contract monitoring approach is presented in [LLM07].

7.1 Automated Contracting of Web Services

Web service contracting can be seen as the process which transforms given Web service offers and requests to Web service contracts. As outlined in Section 3.2.2, this process can be broken down into three phases: the *matching phase* in which possible transactions are discovered (Section 7.1.1), the *allocation phase* in which the final assignment between offers and requests is determined (Section 7.1.2), and the *contract formation phase* in which a legally binding contract is closed (Section 7.1.3). How these three phases of Web service contracting can be implemented based on the presented ontology framework is introduced throughout this section.

7.1.1 Matchmaking Mechanism

The goal of the matching phase is to determine if a given offer o meets the requirements stated in a request r and vice versa. In Definition 5.10 of the abstract model, this issue has been denoted by Multi-attribute Matching Problem (MMP). Conceptually, solving MMP requires determining the intersection of the acceptable trades

$T'_j \cap T'_i$. This breaks down to the comparison of the attribute values that characterize the trades. Since in our approach attribute values can be described by complex ontological structures (see Example 6.2), sophisticated matching mechanisms are required. These mechanism are described in the next section, before we discuss the matching of entire bids.

Matching of Attribute Values

The comparison between a requested attribute value and an offered attribute value is a fundamental functionality. As discussed in Section 6.2.1, the Core Policy Ontology supports primitive attributes where values are directly expressible with an OWL datatype as well as more complex attributes where values may capture classes of individuals. Semantic technologies provide a wide range of techniques for handling these complex attributes. In this context, various techniques have been proposed for matching of OWL class descriptions based on DL inferencing, distinguishing between several notions of match, which are typically reducible to instance checking, concept satisfiability or subsumption. For example, for two OWL classes a_i and a_j that represent attribute values of a request or an offer, the degrees of match proposed in [PKPS02] are: *exact* if a_i and a_j are equivalent, *plugin* if a_i is a subclass of a_j, *subsumes* if a_j is a subclass of a_i, *intersect* if the conjunction of a_i and a_j is satisfiable, and *fail* if the conjunction of a_i and a_j is unsatisfiable.

As already discussed in the context of the specification of functions (Section 6.2.1), we support these notions of match in our framework, and also allow for others by including customizable matching predicates into the service selection algorithm. In fact, since we use a declarative formalism to describe how attribute values are matched, a user can bring in arbitrarily complex matchmaking behavior expressed in rules. This facilitates the adaption of the selection component to changing service descriptions (e.g. with new attributes) and thus the flexibility of the matching approach. Contrarily, other approaches typically use hard-coded algorithms to process the matching. Since we keep attributes in Web Service configurations rather generic, the way in which two attribute values are compared strongly depends on the domain of interest (e.g. 'ResponseTime' has to be matched in a different way than 'ServiceType' or 'IndicatedAttraction') and on the way in which they are represented (e.g. the structure of the ontology representing the attribute values). In the following, we illustrate this approach using an example where different domain ontologies are "plugged-in" for describing various types of attributes. Since the required attributes and domain ontologies might change dynamically, our matching approach has to support such flexibility.

Example 7.1 *Figure 7.1 exemplifies a* TradeSituation *where the* AttributeValue *could take values from different domain ontologies. To illustrate this, Figure 7.1 describes three domain ontologies: the first ontologies (denoted by* $\mathcal{O}^{scl}$*) defines different service types such a* scl:RoutePlanningService *or a* scl:CreditInformationService*. The second ontology (denoted by* $\mathcal{O}^{geo}$*) describes geographic knowledge. For example, it defines concepts such as a* geo:Country *or* geo:City *and individuals like* geo:Karlsruhe *representing a concrete* geo:City*. Thus, it can be used, e.g., to describe the 'Coverage' of a route planning service. Third, the tourism ontology (denoted by* $\mathcal{O}^{tourism}$*) captures information about different kinds of attractions, such as* tourism:CulturalAttraction *or* tourism:SportsEvent*, and can be*

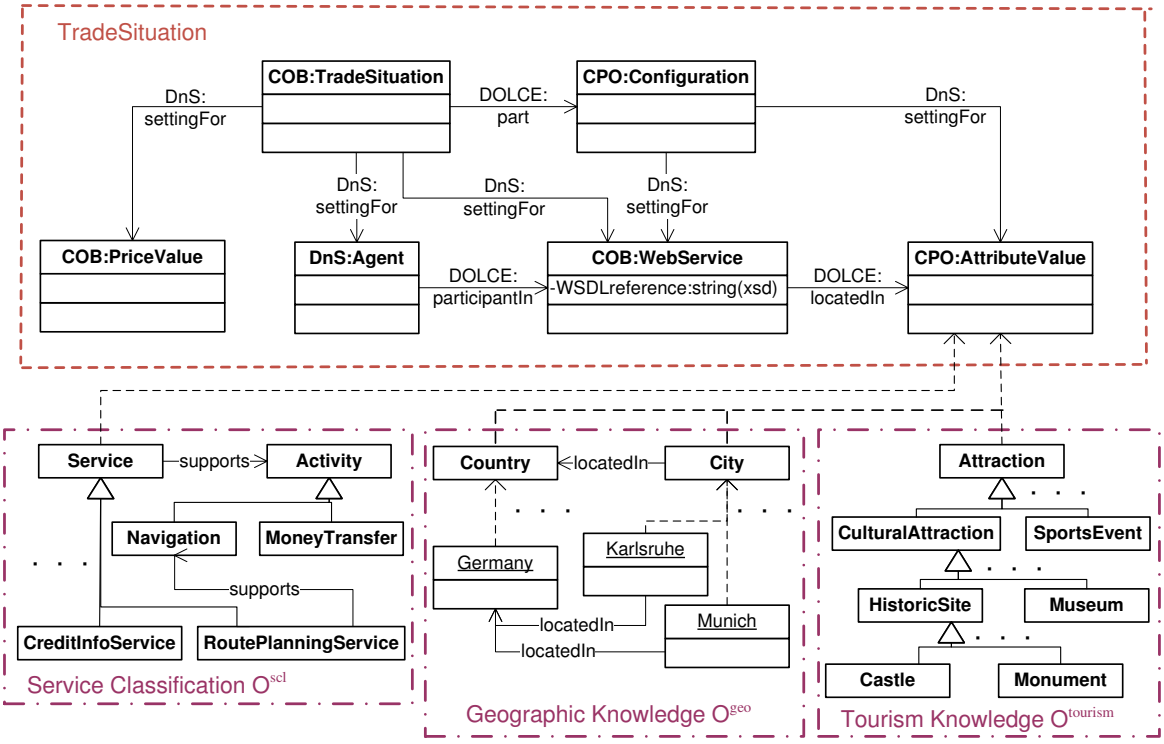

Figure 7.1: Using complex attribute values in a *TradeSituation*

used to describe the attribute 'Indicated Attraction' of a route planning service, for instance (refer to Example 4.3).

Considering now the matching of service types, a requester might ask for a scl:RoutePlanningService *whereas a provider might specify that a service that supports navigation is offered, i.e.* ServiceSupportingNavigation $\equiv$ Service $\sqcap$ $\exists$supports.Navigation. *The following rule definition specifies the matching predicate for service types, requiring them to yield an exact match. This is implemented by the predicate* exact, *which checks for the logical equivalence of a requested concept C_r and provided concept C_p, i.e. $C_r \equiv C_p$.*

$$\text{(R28)} \qquad match(v_1, v_2) \leftarrow ServiceTypeValue(v_1), ServiceTypeValue(v_2), \\ exact(v_1, v_2)$$

In our example, the requested scl:RoutePlanningService *indeed yields an exact match with the provided* ServiceSupportingNavigation, *since O^{scl} entails their equivalence.*

Analogously, values of the attribute IndicatedAttraction *could be matched using the predicate* subsumes(x,y) *instead of* exact(x,y). *Consider the scenario where a requester requires a service that indicates* tourism:Castles *along the route, whereas a provider offers information about* tourism:CulturalAttractions *in general. Again the provided value would match the requested one, since $O_{tourism} \models$ Castle $\sqsubseteq$ CulturalAttraction.*

The values of the attributes StartPoint *and* EndPoint *from Example 4.3 represent individual locations in a geographic ontology and require a different treatment. Here the modeler*

specifies the customized matching behavior for location attributes by introducing a class Location *as a subclass of* CPO:Attribute*, of which the start and destination are instances. The appropriate matching behavior is then captured by the following rule.*

$$\text{(R29)} \qquad match(v_1,v_2) \leftarrow Location(v_1), Location(v_2), matchLocation(v_1,v_2)$$

$$\text{(7.30)} \qquad matchLocation(x,y) = \begin{cases} true & O^{geo} \models DOLCE{:}locatedIn(x,y) \\ false & otherwise \end{cases}$$

The predicate $matchLocation(x,y)$ *is realized as a built-in using a separate call to a description logic reasoner, just as* $exact(x,y)$ *or* $subsumes(x,y)$ *before. Assuming an example in which a requester looks for a service that provides a route from Karlsruhe to Munich and a provider offers a route planning service for Germany, a match would be realized, since Karlsruhe and Munich are both located in Germany according to* O_{geo}*.*

Finally, there are attributes that do not require a complex matching in terms of logical reasoning, but where a simple and efficient string comparison or arithmetic calculations are sufficient. In this case, the modeler of a matching rule can include predefined built-in predicates defined in SWRL. From our example, the QoS attribute ResponseTime *falls into this category, and is processed according to the following rule specification.*

$$\text{(R30)} \qquad match(v_1,v_2) \leftarrow QoSAttributeValue(v_1), QoSAttributeValue(v_2), swrlb{:}equal(v_1,v_2)$$

Here a subclass of AttributeValue*, namely* QoSAttributeValue*, is introduced to enable the specification of the matching behavior for all QoS attributes, such as* ResponseTime*, by a single rule.*

Based on the definitions above, we can define a shortcut for matching two arbitrary configurations as follows. This shortcut enables a more compact representation of the rules to come.

$$\text{(R31)} \qquad compareTrades(t_1,t_2) \leftarrow DOLCE{:}part(t_1,c_1), DOLCE{:}part(t_2,c_2), compare(c_1,c_2)$$

$$\text{(R32)} \qquad compare(c_1,c_2) \leftarrow \bigwedge_{l=1,\ldots,n} attrCompare(a_l,c_1,c_2)$$

$$\text{(R33)} \qquad attrCompare(a,c_1,c_2) \leftarrow DnS{:}settingFor(c_1,v_1), DnS{:}valuedBy(a,v_1), DnS{:}settingFor(c_2,v_2), DnS{:}valuedBy(a,v_2), match(v_1,v_2)$$

After discussing the matching of attribute values, we consider the matching of entire offers and requests in the next section.

Matching of Offers and Requests

In order to solve the Multi-Attribute Matching Problem, the matching predicates introduced in the previous section can be used within the optimization rules that solve the MMP and thus determine the best configuration for a given requester and provider. We define three alternative variants of the *mmp*-predicate, which considerably differ in their underlying assumptions, applicability and performance characteristics. While the first variant [V1] implements the ranking of alternatives based on enumerating all possible trades/configurations (Equation 5.12), [V2] solves MMP on per attribute basis using Equation 5.13-5.15. [V3] goes a step beyond [V2] by utilizing a linear program formulation and applying efficient solving techniques.

Variant [V1] This variant implements Equation 5.12, where a ranking of all offers and configurations is derived by evaluating all possible configurations. We can model the problem purely based on DL-safe rules using some standard SWRL built-in functions. Rule (R34) calculates the difference between score $U_i^S(c,k)$ and price $U_j^P(c)$ of each *Configuration* $c \in C_i \cap C_j$ that is supported by the offer as well as asked for in the request. The following rule is invoked using a concrete Request r, *Offer* o as well as *Context* k, and returns the *Configurations* c with the utility u given the preferences in the Request. By invoking the *mmp*-predicate an allocation can be calculated as shown in Query (R39).

$$\begin{aligned}(\text{R34}) \quad mmp(r,o,k,c,u) \leftarrow & \mathit{COB{:}Request}(r), \mathit{COB{:}Offer}(o), \mathit{Context}(k), \\ & \mathit{COB{:}satisfiesBid}(r,t_1), \mathit{COB{:}satisfiesBid}(o,t_2), \\ & \mathit{compareTrades}(t_1,t_2), \mathit{COB{:}price}(r,t_1,s_1), \\ & \mathit{COB{:}price}(o,t_2,s_2), \mathit{swrlb{:}subtract}(u,s_1,s_2), \\ & \mathit{DOLCE{:}part}(t_2,c)\end{aligned}$$

The *compareTrades*-predicate defined in Rule (R31) is used to match two trades and thus also the corresponding configurations. We use the *satisfiesBid*-predicate defined in Rule (R20) on page 120 in order to determine the acceptable *TradeSituations* T_i and T_j. Since prices are not explicitly given in a *TradeSituation*, the *price*-predicate (refer to Rule (R14) - (R19) on page 119) is used to calculate this information based on the utility function policies defined in the offer or request. Finally, the built-in *swrlb:subtract* is applied to calculate the difference between $U_i^S(c,k)$ and $U_j^P(c)$. The *Context* k is not explicitly required in Rule (R34), since it is handled automatically within the *satisfiesBid*-predicate. However, we add it to the signature of the *mmp*-predicate to be compatible with the other *mmp*-variants where the context is explicitly required.

An advantage of variant [V1] is that one can get a full ranking of all configurations, which might be required in some applications. Furthermore, it can be modeled purely based on standard modeling primitives provided by OWL-DL and SWRL. However, the disadvantages are also evident. Since the approach is based on enumerating all *TradeSituations*/*Configurations*, a finite number

Algorithm 7.1 Determine optimal attribute value for two Policies

procedure OPTFKT(Function f_1, Function f_2, Utility u)
SELECT ?U WHERE {
f_1 CPO:constitutedBy ?P1 . f_2 CPO:constitutedBy ?P2 .
?P1 CPO:policyValue ?V1 ; CPO:valuation ?X .
?P2 CPO:policyValue ?V2 ; CPO:valuation ?Y .
?V1 match ?V2 . EVALUATE ?U := swrlb:subtract(?X,?Y) .
} ORDER BY DESC(?U)
Assign first element of the result set to u
return return u

of configurations is required and thus the approach is not suitable in the presence of continuous attributes. Moreover, all possible/relevant *TradeSituation* have to be stored in the knowledge base, which leads to a high space complexity. As already discussed in Chapter 5, another fundamental problem is the complexity with respect to the number of required utility calculations. Recall that the number of *Configurations* (and therefore also the number of possible *TradeSituation*) that have to be evaluated grows exponentially with respect to the number of attributes.

Variant [V2] The second variant of the *mmp*-predicate implements the decomposed optimization algorithm described in Equations (5.13)-(5.15) and thereby avoids enumerating all *TradeSituations*. In this context, we utilize the additive structure of the pricing as well as scoring functions as follows: the optimal value for each attribute is determined separately and the overall price/score is composed based on these individual measures. Rule (R35) determines the utility of an offer according to request in a specific execution context.

$$\begin{aligned}(\text{R35}) \qquad mmp(r,o,k,u) \leftarrow & Request(r), Offer(o), \\ & \bigwedge_{l=1,\ldots,n} (hasFunction_C(r,k,a_l,f_{rl}), \\ & hasFunction(o,a_l,f_{ol}), optFkt(f_{rl},f_{ol},u_l)), \\ & swrlb{:}add(u,u_1,\ldots,u_n)\end{aligned}$$

The predicates *hasFunction* and $hasFunction_C$ are shortcuts to determine the value function u_{il}^S / u_{jl}^P for a certain attribute A_l without and with context definition. They are defined by the Rules (R36) and (R37), respectively.

$$\begin{aligned}(\text{R36}) \qquad hasFunction(b,a,f) \leftarrow & DnS{:}defines(b,p), Bid(b), Attribute(a), \\ & DnS{:}defines(p,pr), isAssignedTo(pr,a), \\ & Attribute(a), DnS{:}playedBy(pr,f)\end{aligned}$$

(R37) $hasFunction_C(b,k,a,f) \leftarrow DnS{:}defines(b,p), Bid(b), Attribute(a),$
$DnS{:}defines(p,pr), applicableIn(pr,k),$
$isAssignedTo(pr,a), Attribute(a),$
$DnS{:}playedBy(pr,f)$

Since the calculation of the optimal value for a certain attribute requires iterating over an unknown number of attribute values (instances), the calculation cannot be directly expressed in SWRL. We thus use a built-in function, called *optFkt*, to determine the attribute value a_{le} maximizing the utility $u^S_{il}(a_{le}) - u^P_{jl}(a_{le})$ of attribute A_l. Algorithm 7.1 shows the implementation of the built-in-predicate specifically for Point-based Functions. In the predicate *optFkt* for each attribute the requester and provider policies are retrieved from the knowledge base and the attribute value leading to the maximal utility is determined. A major advantage of the approach is that the SPARQL query executed in Algorithm 7.1 uses the *match*-predicate defined in the ontology. Thus, the correct matching algorithm is used for each attribute automatically and the implementation of the built-in is completely domain independent.

Variant [V3] The third variant of the algorithm implements also the decomposed ranking algorithm described in Equation (5.13-5.15), but with some additional optimizations. We formulate the entire MMP as a single linear program that can be solved efficiently by using techniques such as Branch and Bound methods [LW66]. Since the entire problem should be formulated as one linear program, Rule (R38) defines the *mmp*-predicate simply by invoking the built-in *optLP*.

(R38) $mmp(r,o,k,u) \leftarrow COB{:}Request(r), COB{:}Offer(o),$
$CPO{:}Context(k), optLP(r,o,k,u)$

Algorithm 7.2 shows the built-in *optLP* that encapsulates the calculation of the optimal *COB:TradeSituation* for a given *COB:Request* and an *COB:Offer*. The built-in performs a query to get the relevant utilities for the attribute values. This is done again by utilizing the *match*-predicate from the ontology. The optimization problem is constructed and solved using a standard optimization library. Note that for simplicity Algorithm 7.2 is specific to *CPO:PointBasedFunctions*. The corresponding built-ins for *CPO:PiecewiseLinearFunctions* and *CPO:PatternBasedFunctions* are specified by adapting the query in Algorithm 7.2 to the corresponding *Function* definitions. Variant [V3] has the advantage that we can use the efficient implementations for solving integer linear programs provided by standard tools, such as CPLEX[1] or LPSOLVE[2]. Since querying the knowledge base as well as the invocation of the solver involves a costly initialization step, the built-in

[1] `http://www.ilog.com/products/cplex/`
[2] `http://sourceforge.net/projects/lpsolve`

Algorithm 7.2 Optimization built-in using Linear Programming

procedure OPTLP(Request r, Offer o, Context k, Utility u)
resultList := SELECT ?A, ?V1, ?U WHERE {
?A rdf:type CPO:Attribute .
EVALUATE ?F1 := hasFunction(o,?A) .
EVALUATE ?F2 := hasFunction(r,k,?A) .
?F1 CPO:constitutedBy ?P1 ;
?F2 CPO:constitutedBy ?P2 ;
?P1 CPO:policyValue ?V1 ; CPO:valuation ?X .
?P2 CPO:policyValue ?V2 ; CPO:valuation ?Y .
?V1 match ?V2 . EVALUATE ?U := dif(?X,?Y) .}
Initialize matrix $U = (u_{le})_{l=1\ldots n, e=1\ldots \max_l |A_l|}$ with $(A_l, a_{le}, u_{le}) \in$ *resultList*
Initialize matrix $X = (x_{le})_{l=1\ldots n, e=1\ldots \max_l |A_l|}$
determine $u = \max\{\langle U, X\rangle | \forall l \in \{1,\ldots,n\} : \sum_{e=1}^{|A_l|} x_{le} = 1, x \in \{0,1\}\}$
return u for given Offer o and Request r

calculates the utilities for all request/offer-pairs at once and stores the results until they are requested in an internal cache.

In contrast to variant [V1], variants [V2] and [V3] can be easily adapted to handle continuous attributes by introducing appropriate built-ins *optFkt* and *optLP*. However, it is not possible to get a ranked list enumerating all offers and configurations, as it is possible using variant [V1]. Nevertheless, since most application only require selecting the best service with the best configuration, determining the ranked list of offers is sufficient. In the Chapter 9, a detailed analysis of the different modeling approaches with respect to their computational and space complexity is presented.

7.1.2 Allocation Mechanism

Up to now, we have considered only the matching of one offer and one request. In this section, we go beyond this simple case and present different mechanisms that can be used to determine the allocation between several requests and offers. In this context, we first implement the Local Selection Problem (Definition 5.11) based on the ontological descriptions and then extend the approach to dynamic pricing mechanisms.

Web Service Selection

As defined in Section 5.3.2, Web service selection is a Hit-and-Take-mechanism (Definition 2.7) that allows a Web service requester to find the best Web service provider and the best available configuration of this provider. This problem is referred to as the Local Selection Problem (LSP) and does not involve any negotiation mechanisms. Considering the fact that solving the MMP provides already the best configuration for a given offer/request-pair, solving LSP requires only to solve the following simple optimization (Definition 5.11):

$$\max_{j=1,\ldots,|O|} u(r_i, o_j)$$

In this context, $u(r_i, o_j)$ represents the solution for the MMP for a request r_i and an offer o_j. We now assume that the request and several offers are stored in a knowledge base, which is formalized by means of the ontology framework introduced in Chapter 6. In order to derive a ranked list of the *Offer* instances from the knowledge base, we formulate a SPARQL-query that refers to a concrete *Request* instance x and to an instance y representing the current *Context*.

(R39)
```
PREFIX ex: < http://example.org/ns# >
SELECT ?O , ?U WHERE {
   ?O  rdf:type  COB:Offer .
    EVALUATE ?U := mmp(x, ?O, y) .
}ORDER BY DESC(?U)
```

The Query (R39) sorts the result list by decreasing surplus ($U_i^S - U_j^P$), i.e. the best provider is returned as top answer, whereas the worst provider is returned as last element. The utility of one *Offer* with respect to a *Request* is calculated by means of the *mmp*-predicate.

Queries that not only return answers which meet the conditions specified in the query, but also sort the results according to a user's preferences are called *preference queries* [Kie02, LL87, AW00]. In Query (R39) above we realize a preference query by referencing a request in the knowledge base that defines a scoring policy. While storing requests in the knowledge base is a convenient way if the same scoring policies are applied repeatedly, there are other scenarios in which this is not efficient, e.g. if a request is only used once in a service selection. In this case, it is more convenient to represent preferences directly in a query. This can be realized using SPARQL with additional built-in predicates that take a string-encoding of *Functions*. Again, we exemplify the approach using a *PointBasedFunction*. The *pbF*-predicate takes a String representation of the tuples representing the *PointBasedFunction* and an attribute value. The predicate evaluates the *PointBasedFunction* for the attribute value and returns the corresponding score. For example, in the query below a value of 1 is assigned to '*HistoricSites*', a value of 0.5 to '*Museums*' and a value of 0 to '*SportsEvents*'. In addition, to reduce the number of offers that have to be ranked we can add mandatory conditions (viz. goal policies) directly to the query using the SPARQL FILTER element. In the query below, only services that provide a guaranteed response time of less than 5 sec. are retrieved. Note that Query (R40) does not make direct use of the *mmp*-predicate. However, within the *pbF*-built-in the *match*-predicates can be used to semantically match attribute values, e.g. '*Castles*' are matched with '*HistoricSites*' and thus are valuated with 1.

(R40)
```
SELECT ?O, ?C, ?U
 WHERE {
   ?O rdf:type CPO:Offer ; DnS:defines ?A1 ; DnS:defines ?A2 .
   ?A1 rdf:type tourism:IndicatedAttraction ; DnS:valuedBy ?attractionValue .
   ?A2 rdf:type qos:ResponseTime ; DnS:valuedBy ?RTValue .
   EVALUATE ?U := pbF('(tourism:HistoricSites,1),(tourism:Museum,0.5),
         (tourism:SportsEvent,0)',?tattractionValue) .
   FILTER ?RTValue < 5 .
 } ORDER BY DESC(?U)
```

Auction-based Allocation

In this section, we show how the previous selection approach can be extended to auction mechanisms that enable dynamic pricing. As an example, we introduce an extension of the Web service selection approach, where providers are able to improve their offer in case they would not be selected based on their current offer. Since such a mechanism allows the requester to utilize competition on provider side, they are denoted by procurement auctions. Procurement auctions are particularly interesting for large scale enterprise service scenarios, where a large number of Web services (or Web service invocations) is required by a company or public body. Procurement auctions take the form of reverse auctions with a single requester and a set of precertified providers (e.g. providers having an umbrella contract with the requester) negotiating within the context of a private exchange [BKK+02]. Based on the Web service selection algorithm above, we introduce a simple procurement auction as follows:

1. In a first step, the requester performs a Web service selection as defined above, which results in a ranked set of providers. This is done by querying the repository of offers using Query (R39) or (R40).

2. The requester then selects the top-*k* providers from the result set, for which an umbrella contract exists.

3. The requester issues a request for quote (RFQ) to purchase a particular service to the selected providers. This is done by sending a *Request*-instance expressing the requesters requirements specified by means of scoring policies to the selected providers.

4. Each provider assesses the *Request* locally. The provider has a given period of time for updating her *Offer* in the repository.

5. Once the period is over, the requester again performs the usual Web service selection (Query (R39) or (R40)); this time however restricted to the invited providers.

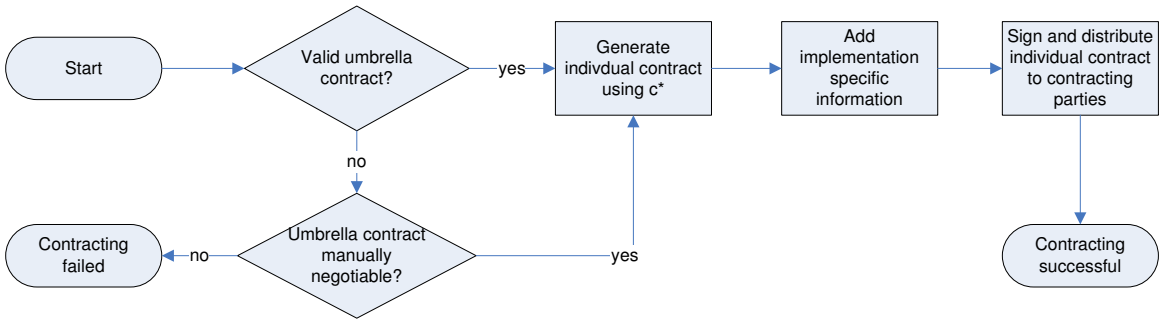

Figure 7.2: Contract formation process.

6. Finally, a Web service contract is concluded with the top-ranked provider using the configuration and price determined in the selection.

The Request-for-Quote-mechanism introduced above corresponds to a traditional sealed-bid auction. A concrete implementation of this protocol is presented in Chapter 8. A major advantage compared to the pure selection approach is that providers can adapt their offer according to a specific request. This is important in many scenarios, where a full *Offer* specification is not possible. For example, considering a flight booking service, it is not possible for a provider to add all available flights to the *Offer*, although this would be required in order to make sure that the service is suitable for a requester looking for a flight between Berlin and Karlsruhe at Friday evening. By allowing to adapt the *Offer* to the *Request*, this problem can be addressed (c.f. [FKL+05]). However, the additional time period for bidding and the communication overhead obstruct the application in time-critical scenarios, where services have to be selected at runtime. A more thorough discussion of the economic properties of (multi-attribute) RFQ auctions can be found in [RS01, Che93, Mil00].

Of course, many other forms of auctions can be implemented based on the MMP problem. For example, in [LS06] an extension to the multi-attribute combinatorial double auction MACE [SNVW06] is presented that features semantic matching of bids based on the Core Ontology of Bids. Due to the double-sided mechanism, this auction handles competition on requester as well as provider side and thus also allows dealing with the resource limitation of service providers. Therefore, this mechanism particularly interesting for grid applications. We discuss this mechanism in more detail together with the corresponding implementation in Chapter 8.

7.1.3 Contract Formation

In the allocation phase, a provider and requester agreed on a configuration of a configurable service and on the price a requester has to pay for this service. In Section 6.4, we presented a semi-automated contracting approach distinguishing between an umbrella contract that is manually closed and an individual contract that can be fully formalized using the Core Contract Ontology. In this section, we discuss how an agreement in the allocation phase is transformed to a legally binding contract.

This contract formation process can be intuitively described as follows: Assume an agreement between a requester i and a provider j has been found with the agreed configuration c^*. Depending on the architecture of the system, the following process

can be executed by one of the parties or by an independent intermediary, such as an auctioneer or a service repository provider. Figure 7.2 sketches the workflow described in the following.

1. First, it has to be determined whether there exists a valid umbrella contract between requester i and provider j.

2. If this is not the case, either an umbrella contract has to be manually negotiated or the contract formation fails. Manual negotiation is only possible in case the Web service binding is determined at development or deployment time; for dynamic binding at runtime it is recommended to consider in the matching and allocation process only providers for which a valid umbrella agreement exists. This can be realized by explicitly stating a corresponding policy within the *Request*.

3. If there is an umbrella contract, the individual contract can be generated based on the agreed configuration $c^* = (a_1,\ldots,a_q,\ldots,a_n)$. Therefore, the *CustomerObligations* and *ProviderObligations* have to be generated. Subsequently, the *Attributes* A_q have to be added to the *Obligations* and the *DnS:valuedBy*-relations to the *AttributeValues* a_q with $q = 1,\ldots,n$.

4. Once the individual contract is generated, implementation specific information is added, which is required for the contract execution. This might include the physical address where the Web service can be invoked (e.g. a WSDL document), properties of the authentication or encryption algorithm, etc.

5. As a next step, the contract is signed by the responsible party (e.g. the market operator) using existing electronic signature infrastructure such as X.509, CA, PGP, PKIX and SKIP [Ger00]. The signed individual contract is then made available to the contracting parties.

6. Finally, the contract can be automatically used by the participants. Therefore, they store the contract in their local knowledge base. The requester uses the contract to generate the service invocation (see Chapter 8) and to monitor the execution as outlined in the next section. The provider queries the knowledge base in order to verify if access should be granted to a specific requester and to determine how the service should be configured for this requester. In addition, a provider could use the set of contracts to internally schedule the execution of the different contracts according to the guaranteed service levels (e.g. to minimize the risk of penalties).

After a contract is closed, the contracting phase is finished. In the next section, we consider the monitoring of contracts which is part of the subsequent settlement phase.

7.2 Automated Monitoring of Web Service Contracts

A major advantage of machine-interpretable, formal contracts is the fact that monitoring whether a Web service execution complies with the contract can be (at least

partially) automated. This is particularly important in a scenario where many (different) contracts are closed and executed within a short timeframe. As discussed in Section 4.3, this is usually the case for large-scale service-oriented architectures involving enterprise and grid services. We already discussed in Section 6.4.4, how the contract evaluation function $\varrho(\Gamma, c)$ is implemented by the *satisfiesObligation*- and *satisfiesContract*-predicates in the Core Contract Ontology. In this section, we exemplify how these rules can be applied together with additional expert knowledge in order to monitor the compliance of contracts.

Since terms within contracts are often context-dependent and require fuzzy interpretations, the evaluation process is not always possible by simply applying the *satisfiesObligation* and *satisfiesContract* predicates. For example, certain obligations have to be done "immediately" or "with utmost care". Although such terms are interpreted in various different ways, lawyers typically have guidelines how to interpret such statements and expressions in a given context. As these guidelines are not part of the contract, we have to add them to the knowledge base in a formalized way, which allows their inclusion in the contract evaluation process. For instance, if the term "immediately" is used to specify a timeframe in which a response of the service is expected, one could use the following rule of thumb: considering the current state of the art a response is received "immediately" only if it is received within 5 seconds after sending the request. Subsequently, we exemplify this approach by changing the *ProviderObligation* introduced in Example 6.9 to include a term that cannot be directly interpreted.

Example 7.2 *The credit information service provider X has to provide a complete set of* Business Background Information *of company SAP to customer Y. This has to be done immediately after receiving the customer's request. Therefore, we derive the following formal definition of the* Provider Obligation*:*

$$
\begin{aligned}
&\mathit{Obligation}(\mathit{ProviderObligationX})\\
&\mathit{Provider}(X)\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, X)\\
&\mathit{InformationGood}(BBInformation/SAP)\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, BBInformation/SAP)\\
&\mathit{ServiceTask}(\mathit{Deliver})\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, \mathit{Deliver})\\
&\mathit{ResponseTime}(\mathit{ResponseTimeX})\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, \mathit{ResponseTimeX})\\
&\mathit{ResponseTimeValue}(\text{`}\mathit{immediately}\text{`})\\
&\mathit{DnS{:}valuedBy}(\mathit{ResponseTimeX}, \text{`}\mathit{immediately}\text{`})\\
&\mathit{DnS{:}obligedTo}(X, \mathit{Transfer})\\
&\mathit{DnS{:}anakasticDutyTowards}(BBInformation/SAP, \mathit{Transfer})\\
&\mathit{DnS{:}requisiteFor}(\mathit{responseTimeX}, \mathit{Transfer})
\end{aligned}
$$

Assume that the requester monitored the execution of the contract *ContractX* with the *ProviderObligation* above and observed the *MonitoringInformation*

shown in Figure 6.11 on page 133. Based on this *MonitoringInformation*-instance, the requester evaluates the contract using the *satisfiesContract*-predicate. To find out if the contract is fulfilled, the following SPARQL-query has to be executed:

(R41)
```
SELECT ?M
 WHERE {
   ?M rdf:type CCO:MonitoringInformation .
   ?M CCO:satisfiesContract ContractX .}
```

If the query returns the *MonitoringInformation* shown in Figure 6.11, the contract has been fulfilled correctly. Otherwise there is at least one obligation violated. However, the above query does not give evidence about the reason of the violation. In order to determine the legal consequences of a violation one might want to specify more fine-grained questions. In legal practise, typically a scheme of questions is applied to determine the source of violation. The following questions exemplify this approach using the obligation defined in Example 7.2:

1. **Was the requested trading object delivered?** To answer this question, we have to find out whether information is delivered by the provider at all, and - in case it is - whether the delivered information is complete with respect to the agreement in the *CCO:ContractDescription*. We realize this by comparing the delivered *CSO:ComputationalObject* contained in the *CCO:MonitoringInformation* with the *CCO:InformationGood* agreed on in the contract. Assuming the hierarchy of credit information presented in Example 4.1, the following *match*-predicate might be defined for credit information:

(R42)
$$match(c_r, c_o) \leftarrow CreditInformation(c_r), CreditInformation(c_o), subsumes(c_o, c_r)$$

We thus allow the provider to send more information than required, while making sure that at least the information agreed on is provided. Based on this *match*-predicate, we determine whether a delivered information is correct as follows:

(R43)
$$correctInformation(m, c) \leftarrow CCO{:}MonitoringInformation(m), CCO{:}ProviderObligation(c), DnS{:}defines(c, t), CCO{:}InformationGood(t), DnS{:}playedBy(t, d1), CSO{:}Data(d1), DnS{:}settingFor(m, d2), CSO{:}Data(d2), match(d1, d2)$$

Alternatively, Rule (R43) could also be expressed by means of a SPARQL-query. This would be more appropriate, e.g., if the evaluation is done only once.

2. **Was the correct service task executed?** In a similar way, we can also answer this question by formulating the *correctActivity*-rule. This time the executed *CSO:ComputationalActivity* stated in the *CCO:MonitoringInformation* is compared to the *CSO:ComputationalActivity* agreed upon in the *CCO:ContractDescription*. Again we use a *match*-predicate that relies on the *subsumes*-predicate. Thereby, we allow a general activity description in the contract to be fulfilled by a more specific activity. For example, a contract might specify that certain information has to be transferred. How this should be done is not specified exactly. Therefore, sending by mail or telling on the phone might be admissible since both are certain types for delivering information.

(R44) $$\begin{aligned} \mathit{correctActivity}(m,c) \leftarrow\ & \mathit{CCO{:}MonitoringInformation}(m), \\ & \mathit{CCO{:}ProviderObligation}(c), \mathit{DnS{:}defines}(c,t), \\ & \mathit{CCO{:}ServiceTask}(t), \mathit{DnS{:}sequences}(t,a1), \\ & \mathit{CSO{:}ComputationalActivity}(a1), \mathit{DnS{:}settingFor}(m,a2), \\ & \mathit{CSO{:}ComputationalActivity}(a2), \mathit{match}(a1,a2) \end{aligned}$$

3. **Was the task executed within the required timeframe?** According to the *CCO:ProviderObligation* a *CCO:ServiceTask* has to be executed within a certain time, which is denoted by *ResponseTime*. This is verified by the rule below, which compares the monitored execution time with the *ResponseTime* in the *CCO:ContractDescription*.

(R45) $$\begin{aligned} \mathit{activityInTime}(m,c) \leftarrow\ & \mathit{CCO{:}MonitoringInformation}(m), \\ & \mathit{CCO{:}ProviderObligation}(c), \mathit{DnS{:}defines}(c,t), \\ & \mathit{CCO{:}ServiceTask}(t), \mathit{ResponseTime}(d), \\ & \mathit{DnS{:}requisiteFor}(d,t), \mathit{DnS{:}valuedBy}(d,r1), \\ & \mathit{DnS{:}settingFor}(x,a), \mathit{CSO{:}ComputationalActivity}(a), \\ & \mathit{DOLCE{:}inherentIn}(r2,a), \mathit{DOLCE{:}Region}(r2), \\ & \mathit{interpretRT}(r1,r2), \end{aligned}$$

However, since *ResponseTime* is expressed by a linguistic term rather than a *DOLCE:Temporal-Region*, we need an interpretation rule, which specifies how the linguistic terms should be interpreted. We use the *interpretRT*-predicate to realize this interpretation independent from Rule (R45). This is important since the interpretation of the linguistic terms may change from time to time (e.g. due to new court decisions) and thus Rule (R46) has to be

adapted. Note that the interpretation of the term "immediately" is not content of the contract but rather common sense knowledge modeled by a company's lawyers.

(R46) $interpretRT(r1,r2) \leftarrow swrlb{:}equal(r1,'immediately'), match('<5s',r2)$

Rule (R46) compares the required response time specified in the contract with the static term 'immediately' and in case of a match uses the static *DOLCE:Temporal-Region* '$<5s$' to judge the observed response time in the *MonitoringInformation*. This is realized using the *match*-predicate applicable for two *DOLCE:Temporal-Regions*.

Of course, similar questions about other elements of the obligations can be formulated and expressed via rules or queries. Thereby, a company- and domain-specific evaluation process can be assembled by including interpretation rules where necessary.

7.3 Conclusion

In this chapter, we have presented algorithms for dynamic contracting of Web services. One of the major contributions is the seamless integration of semantic matching algorithms with efficient optimization techniques. Thereby, we strive for a contracting mechanism that can be executed at runtime even for highly configurable offers and requests (Requirements (*R9*) and (*R10*)). In addition, customizable matching rules are used to define how attribute values of a certain attribute should be matched. Since this approach enables adding attributes to the vocabulary during runtime, it facilitates flexibility in the system (Requirement (*R8*)). Moreover, in contrast to most related work, it also does not require a universal and predefined matching algorithm, which we believe is in many cases not realistic; particularly if existing domain ontologies are used for describing attribute values.

Since our contracting mechanism features automated customization of Web services to the requester's needs (Requirement (*R7*)), each service invocation might be executed with a different underlying contract (Requirement (*R11*)). For example, each contract might specify different quality of service levels. To support the participant with managing these heterogeneous contracts, a monitoring mechanism for contracts expressed with the Core Contract Ontology has been introduced, which can be adapted to different scenarios in a flexible way by formalizing legal contract evaluation rules.

In this part, we have presented the design of a semantic Web service market. In the next part, a concrete implementation of this market design is presented and it is discussed whether the requirements identified in Chapter 4 are addressed appropriately.

Part III

Realization and Evaluation

Chapter 8

Implementation

Having introduced the ontological models for representing offers, requests and contracts as well as the Web service contracting and contract monitoring process, in this chapter, a prototype that implements this process is presented. First, the general architecture of the system is presented in Section 8.1. Subsequently, the individual components of the architecture are discussed: Section 8.2 introduces the business process component which provides the means for designing and executing Web services in a business process. Section 8.3 presents a tool for the specification of policies. Policies are a part of requests and offers. They are used during the execution of the business process in order to bind Web services in a flexible way. Section 8.4 introduces the Web service market platform, which is used to bring service providers and requesters together. Finally, in Section 8.5, we exemplify how the prototype can be used in a mobile and grid scenario.

Implementation details of the prototype have been published in conference proceedings and journals [LS06, LA07, Aga07, LAGS07]. The prototype is available for download at `http://kaonws.sourceforge.net`.

8.1 General Architecture

The ontologies and algorithms described in this work are implemented in the ***KA**rlsruhe **S**emantic **W**eb **S**ervices Framework* (KASWS).[1] As shown in Figure 8.1, the prototype consists of two components: the KASWS client residing at the provider and requester side, and the KASWS server that can be operated by a requester, provider or a third party depending on the concrete scenario.

The KASWS client facilitates the specification of Web service offers as well as requests and enables the automated execution of the selected Web services, their monitoring and contract evaluation. For specifying the client's business process, a standard business process designer (e.g. ActiveBPEL Designer®, ORACLE Business Process Manager®) is used. This design is augmented by a business process execution engine and a bid specification tool. This tool enables creating goal as well as utility function policies that can be used within the business process designer to specify the client's desired binding. In addition, the tool allows managing the ontologies on the server, e.g. by specify mappings between ontologies or adding

[1] More information about the KASWS prototype can be found on `http://kasws.sourceforge.net`.

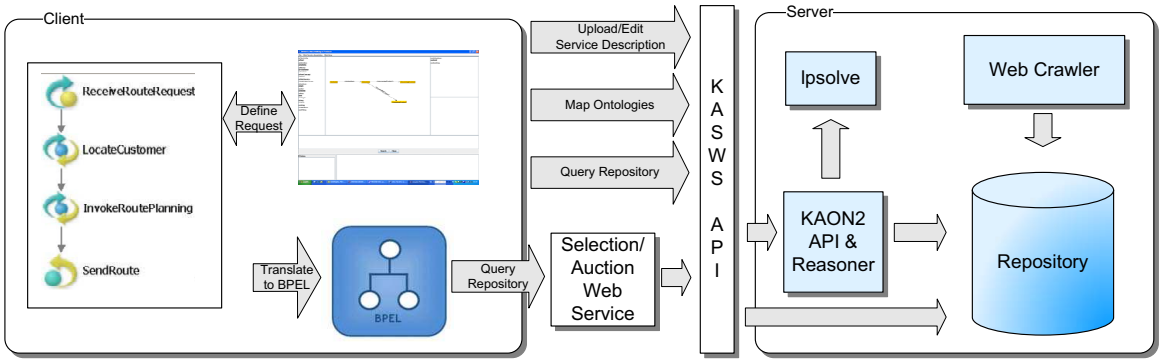

Figure 8.1: General Architecture.

matching rules. The business process and the policy specification components are discussed in Section 8.2 and 8.3, respectively.

The KASWS server provides a repository for Web service offers and requests together with algorithms for determining matches and allocations. This component thus represents a market platform that enables flexible binding of services in a service-oriented architecture by connecting service providers and requesters. In order to perform the matching, an ontology reasoner is required for comparing the attribute values defined in offers and requests. For our implementation, we use the KAON2[2] API and reasoner. To determine the allocation, a linear programming solver is added that can be invoked by the reasoner on demand. In the prototype we use the open source solver lpsolve[3]. The server component can be accessed either via Web service interfaces or via the client tool. A more detailed description of the component can be found in Section 8.4.

8.2 Business Process Component

As discussed in Section 2.1.3, WS-BPEL [OAS06a] has become a quasi-standard for specifying and executing service-based business processes. We thus rely on existing WS-BPEL engines instead of taking care of the business process execution ourself. Therefore, a standard WS-BPEL engine and a WS-BPEL process designer have been added to the client tool. A WS-BPEL specification defines which services should be executed in which order. However, currently WS-BPEL only supports two forms of binding mechanisms: "Binding by Inclusion" and "Binding by Reference" (cf. Section 2.1.2). Since "Binding by Constraint" is not supported, dynamic binding mechanisms cannot be realized [PA05]. In order to enable dynamic service selection at runtime, we propose modeling a dynamic invocation task in the WS-BPEL process as two subsequent process tasks, where the first task is responsible for determining (e.g. selecting) the Web service that is executed in the second task. For binding a Web service during the runtime of the process, we utilize the distinction between ports and port types. This feature allows us to dynamically re-assign end points as long as the service candidates have an identical interface, i.e. port type.

[2] Available for download at `http://kaon2.semanticweb.org/`.
[3] Available for download at `http://sourceforge.net/projects/lpsolve`

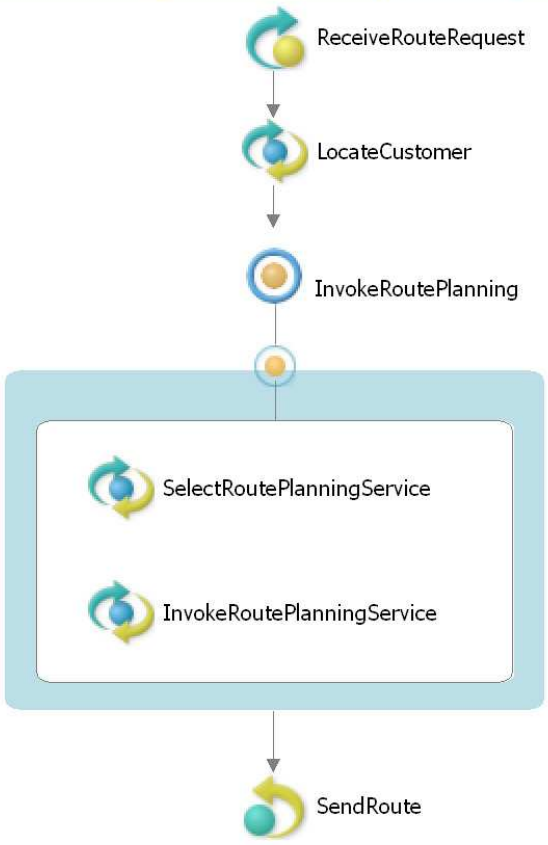

Figure 8.2: Extended WS-BPEL process for dynamic Web service binding.

This approach is exemplified in Figure 8.2 by means of the route planning example (Example 4.3). Here the route planning service should be chosen dynamically, since the required Web service depends on the location of the customer. In contrast to the process used in Example 4.3, the 'InvokeRoutePlanning' task is modeled using two subtasks in Figure 8.2: 'SelectRoutePlanningService' and 'InvokeRoutePlanningService'. Both tasks are invocation-tasks, where the former sends a *Request* to the server component to get the address of a suitable service and the latter uses this received address and invokes the actual Web service. We realize the *Binding by Constraint* paradigm by specifying a SPARQL query referring to a *Request* (such as Query R39 or R40), which is then passed to the server. In this case, SPARQL provides a standardized language for identifying suitable services without referring to a concrete name or identifier. Parts of the query are generated at development time of the process, while others can be added dynamically at runtime.

To illustrate this approach, Listing 8.1 shows an excerpt from the WS-BPEL process that implements the business process shown in Figure 8.2. In lines 2-4, the user's request is received, the contained *clientId* is passed to the location service, where the current country of the user is determined (lines 8-10). Then the SPARQL query, which statically refers to the *Request ns1:RequestOperatorX* in the knowledge base, is extended by the context *location* (lines 12-15) that has been determined by the location service. That means a SPARQL-request similar to those defined in (R39)/(R40) is dynamically constructed during the process execution and stored in the variable *routeRequest*. This variable is then passed to the server by means of the provided Web service interfaces. This invocation takes place in lines 18-20. The

```
1  ...
2  <receive name="receiveRoute" partnerLink="customer"
3      portType="client:Process" operation="initiate"
4      variable="routeRequest" createInstance="yes"/>
5  <assign name="Assign_Query">
6      <copy> <from variable="routeRequest" part="user"/>
7          <to variable="clientID"/>
8  <invoke name="LocationCheck" partnerLink="LocationServicePLT"
9      portType="ns1:LocationService" operation="executeQuery"
10     inputVariable="clientID" outputVariable="location"/>
11 <assign name="Assign_Query">
12     <copy> <from expression='"concat(string(\"SELECT ?O , ?U WHERE {
13             EVALUATE mmp(ns1:RequestOperatorX, ?O,\" ),
14             bpws:getVariableData('location','Country'),string(\",?U ) . }\"))"'/>
15         <to variable="requestQuery" part="queryMessage"/>
16     </copy>
17 </assign>
18 <invoke name="ServiceInvoke" partnerLink="SelectionServicePLT"
19     portType="ns1:SelectionService" operation="executeQuery"
20     inputVariable="requestQuery" outputVariable="responseQuery"/>
21 <assign name="Assign_Port">
22     <copy> <from variable="responseQuery" part="queryResult"/>
23         <to partnerLink="RoutePlanner"/>
24     </copy>
25 </assign>
26 <invoke name="RoutePlanningInvoke" partnerLink="RoutePlanner"
27     portType="ns1:RoutePlanningService" operation="requestRoute"
28     inputVariable="routeRequest" outputVariable="responseRoute"/>
29 ...
```

Listing 8.1: Flexible binding in WS-BPEL.

server internally executes the contracting process and returns the concluded contract containing the port of the chosen Web service. This port is assigned to the port type of the following partner link (lines 21-25) and the route planning service is invoked by passing the original user request containing the start and end point (lines 26-28) to the selected Web service.

This approach works as long as all Web services being considered have the same interface, viz., port type definition. In case service candidates have different interfaces, dynamic selection requires complex interface mappings, which are currently not supported by our prototype. In [PA05], an approach for dynamic binding of services using reflection is presented. However, this technique cannot be used directly for WS-BPEL. A general discussion of mapping mechanisms between ontological and WSDL interface descriptions (often called grounding mechanisms) can be found in [KRMF06].

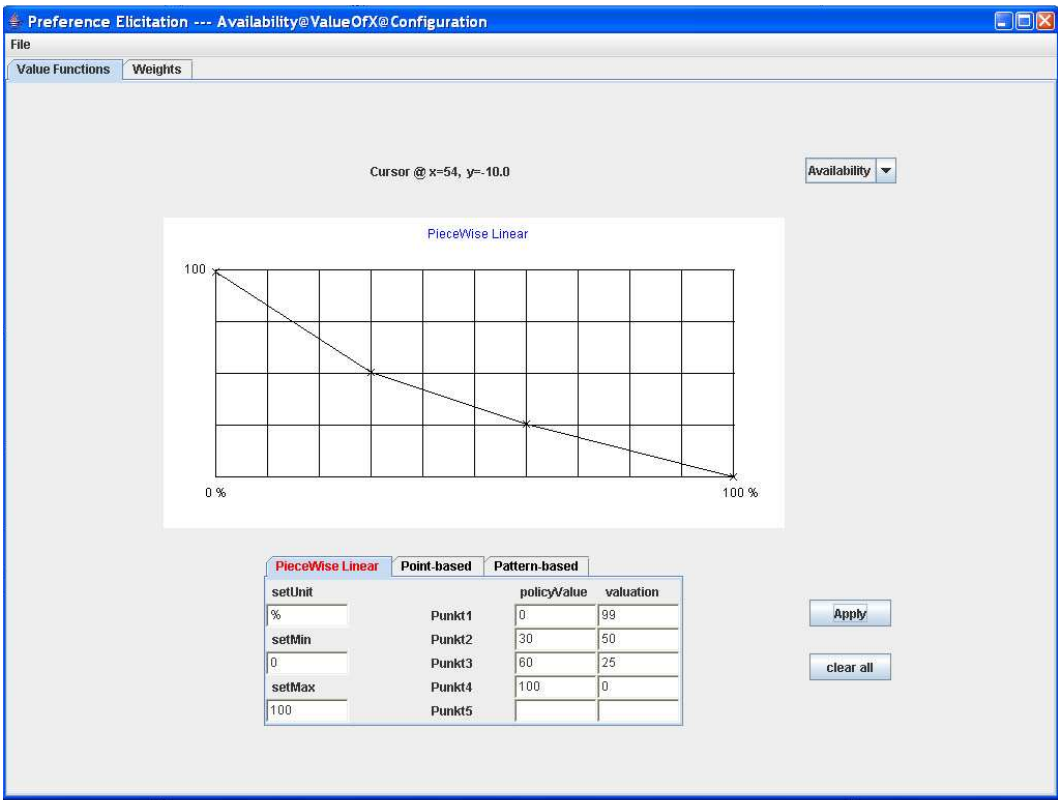

Figure 8.3: Screenshot of the visual policy editor.

8.3 Bid Specification Component

As discussed in the last section, in order to enable dynamic Web service binding, a query is specified within the process description that defines the policies of the requester explicitly (Query (R40)) or via a reference to a *Request* in the repository (Query (R39)). Therefore, means for the specification of *Requests* as well as *Offers*, and the corresponding queries are required. These are provided by the bid specification component. It allows specifying bids and transferring them to the server. In this context, the domain ontologies that are used can be changed and mapped to alternative ontologies. For specification of mappings between ontology constructs, the approach presented in [HM05] is used. The specification of *Requests* and *Offers* involves the modeling of policies. This is supported by a simple visual editor that enables the definition of utility functions (namely *PointBasedFunctions*, *PiecewiseLinearFunctions* or *PatternBasedFunctions*) over attributes of a Web service and to define weights for the different attributes, capturing their relative importance. Figure 8.3 shows a screenshot of the visual policy editor. Specified policies can be stored remotely on the server or directly added to the query.

To support arbitrary domain ontologies, SPARQL queries can be defined using the editor shown in Figure 8.4. The ontology concepts on the left are used to assemble a query in the central panel. The query can be composed by a drag-and-drop mechanism. Policies can be added to all variables that are connected to variables typed as *CPO:Attributes*. They are indicated by a blue circle symbol labeled with a 'P'. In addition, SPARQL filter conditions can be added using the input field below the central panel. Once a query is generated, it can either be stored on the server as a

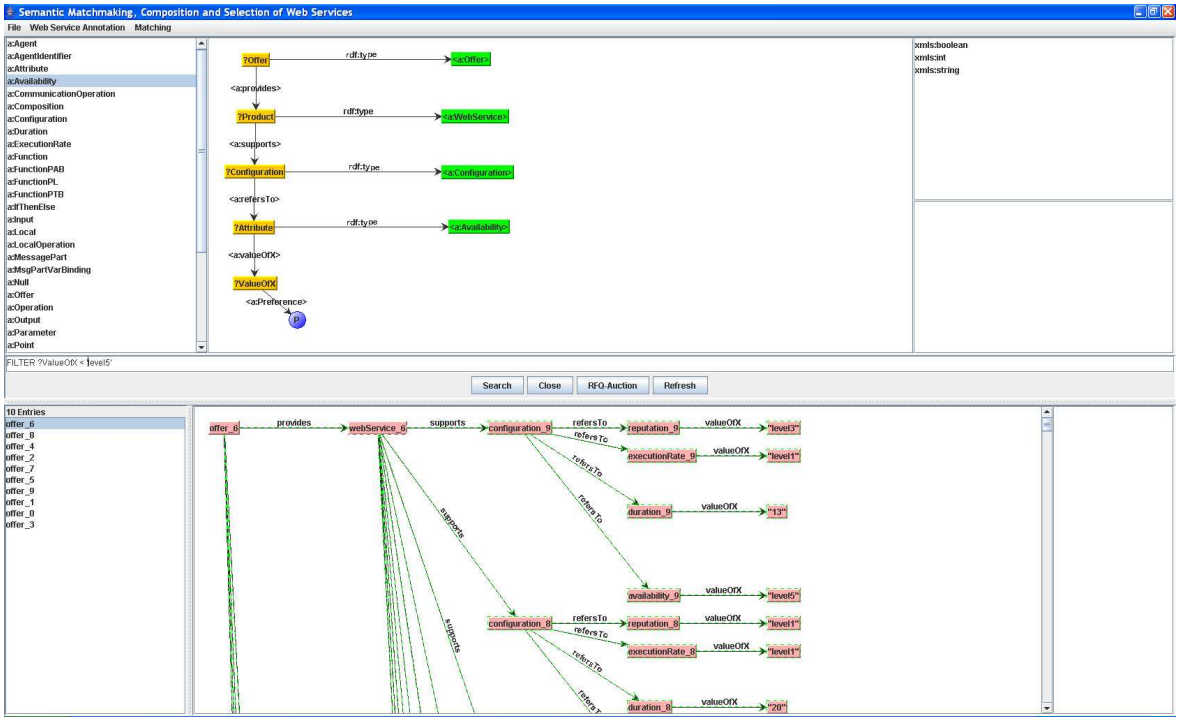

Figure 8.4: Screenshot of the SPARQL query editor.

Request / *Offer*, it can be serialized as SPARQL for inclusion in the business process, or it can be used directly for initiating a Web service selection or auction process. The buttons 'Search' and 'RFQ-Auction' are intended for the latter. A Web service selection results in a ranked list of Web service offers, which is shown on the left panel below the 'Search'-button. Each offer can be selected for closer examination. The offer is displayed with its attributes in the central lower panel. If no suitable service is contained in the result list, a subset of offers can be selected as participants in a procurement auction as described in Section 7.1.2.

8.4 Web Service Market Platform

The server component provides a market platform where providers and requesters can place their bids in order to conclude a contract. The bids are stored in a repository. The repository can be seen as a description logic knowledge base. It can be accessed via the inference engine KAON2. KAON2 is chosen because it supports a suitable description logic fragment ($\mathcal{SHIQ}(\mathbf{D})$) as well as the DL-safe subset of SWRL. $\mathcal{SHIQ}(\mathbf{D})$ corresponds to the logic fragment required for expressing OWL-DL ontologies ($\mathcal{SHOIN}(\mathbf{D})$) without the construct of nominals (i.e. enumerated classes), but including qualified number restrictions. KAON2 thus supports the formalism required for our bid and contract descriptions. In addition, KAON2 has been optimized for query answering [MS06], which is the main functionality required for implementing the Web service market.

A single Web service market platform may support several different allocation mechanisms. In order to support this feature, bids stored in the repository are assigned to *TradingComponents*, which virtually segment the bids into distinct markets. We therefore extend the ontology by the following axioms:

(A79) $\mathit{TradingComponent} \sqsubseteq \mathit{DnS{:}SituationDescription} \sqcap$
$=_1 \mathit{hasIdentifier.XSD{:}String} \sqcap \exists \mathit{DOLCE{:}part.Bid}$

(A80) $\mathit{hasIdentifier} \sqsubseteq \mathit{DOLCE{:}locatedIn}$

According to Axiom (A79), a *TradingComponent* contains at least one *Bid* and is identified uniquely by a string. This allows an allocation algorithm to consider only a subset of bids assigned to a certain *TradingComponent*. The approach can not only be used to provide different allocation mechanisms based on a single repository, it also provides means for distributing a single market instance into independent submarkets. As shown in [LS06], such an approach could considerably improve scalability of a complex allocation mechanism such as a multi-attribute combinatorial exchange.

The allocation algorithms are also (partly) modeled in the knowledge base. For example, Rules (R34) - (R38) specify how the ranking should be determined in a Web service selection using KAON2. Since usually description logic reasoners are not designed for handling numbers efficiently, calculations are executed using built-in predicates such as *swrlb:add* or *swrlb:divide*. Rule (R38) even requires a linear programming solver to calculate the ranking. This is realized by providing the built-in *optLP*, which executes the semantic matching of attribute values and then internally invokes a linear solver. In our current implementation we use the open-source solver lpsolve Version 5.5.

For settings in which the KASWS server cannot be accessed via the bid specification tool, an additional Web service interface is provided. This interface enables accessing the server via the SOAP protocol and thus allows a direct access from a BPEL execution engine, which is utilized in the business process specified above (Listing 8.1). Different allocation mechanisms might however require different operations in the Web service interface. The most simple interface provides only two operations: (*i*) *executeUpdate* which is used for adding/deleting *Offers* and *Requests* and (*ii*) *executeQuery* which is used for issuing a SPARQL query to the repository. This two operations are exemplified by the excerpt from the WSDL document shown in Listing 8.2. The interface (i.e. port type) is defined in lines 2-11. It contains the two operations *executeQuery* and *executeUpdate* and defines the corresponding inputs and outputs. In addition, the concrete SOAP serialization of the messages is defined in lines 12-25.

Due to the lack of a publicly available UDDI repositories, a Web crawler is added to the server tool that automatically collects WSDL files from the Web and transforms HTML forms to offer descriptions.

8.5 Application Example

In order to show how the prototype can be used in a real world scenario, we outline two applications for a mobile and grid setting.

```
1  ...
2  <portType name="SelectionService">
3      <operation name="executeQuery">
4          <input message="tns:executeQuery"></input>
5          <output message="tns:executeQueryResponse"></output>
6      </operation>
7      <operation name="executeUpdate">
8          <input message="tns:executeUpdate"></input>
9          <output message="tns:executeUpdateResponse"></output>
10     </operation>
11 </portType>
12 <binding name="SelectionServicePortBinding" type="tns:SelectionService">
13     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
           document"></soap:binding>
14     <operation name="executeQuery">
15         <soap:operation soapAction=""></soap:operation>
16         <input><soap:body use="literal"></soap:body></input>
17         <output><soap:body use="literal"></soap:body></output>
18     </operation>
19     <operation name="executeUpdate">
20         <soap:operation soapAction=""></soap:operation>
21         <input><soap:body use="literal"></soap:body></input>
22         <output><soap:body use="literal"></soap:body></output>
23     </operation>
24 </binding>
25 ...
```

Listing 8.2: Web service interface (WSDL) of the server component.

8.5.1 Automated Web Service Selection for Mobile Applications

In order to illustrate the application of the system, we come back to our initial scenario described in Example 4.3, where a mobile phone user sends a route request to the network operator. Depending on the location of the customer the operator selects dynamically an appropriate route planning service, invokes this service, and finally forwards the route to the customer. By means of our prototype, the scenario can be implemented as follows:

1. First, the network operator implements its business process using the BPEL designer. This process might correspond to the process depicted in Figure 8.2. In this context, each dynamic invocation of a Web service has to consist of two invocations: one invocation of the selection service and one of the actual Web service.

2. In a second step, the network operator has to define preferences and conditions for Web services that should be selected dynamically. This can be done by means of the bid specification tool. It supports the network operator in defining the request including policies. This request is then serialized as a SPARQL-query and the query-string is used as input of the service selection invocation. In doing so, the customer's location derived from the previous Web service invocation has to be added as context to the query.

3. Once the process has been finalized, it is serialized using the standard WS-BPEL XML syntax, which is understood by standard WS-BPEL execution engines. The serialization corresponding to Figure 8.2 is shown in Listing 8.1.

4. This process specification is then deployed on the network operator's Web server, which is augmented by a WS-BPEL engine. Thus, the WS-BPEL process itself is exposed as a Web service to the customers. The business process is now usable by customers.

5. Once a customer sends a route request with the correct input data, the execution of business process is triggered. That means, in the first step the location of the customer is determined, the location is added as context to the query and then the query is send to the server using the *executeQuery*-operation defined in the WSDL file (compare Listing 8.2).

6. Within the server component the query is forwarded to the KAON2-reasoner, which executes the query on the repository. In order to rank the returned offers, a linear program might be generated and solved by using the lpsolve-libraries. The Web service ranked first is returned to the requester via the output variable of the selection Web service (denoted by *executeQueryResponse* in Listing 8.2). In enterprise and grid scenarios, a contract formation algorithm as outlined in Section 7.1.3 might be executed. In such cases, a *ContractDescription* instance is returned as result from the server.

7. Once the result of the selection service is received, the port address of the route planning service to be invoked can be assigned to the subsequent invocation operation.

8. Then the actual route planning Web service is invoked by the WS-BPEL execution engine, which yields a route that can be returned to the customer and displayed on the mobile device.

It is straightforward to extend this small example to a broader application scenario, where a lot of other services can be provided to the customer, such as event, cinema, traffic, weather, music or stock quote services. For example, compare the services listed at service repositories such as WebRPC[4] or programmableweb[5]. WebRPC also provides programmatic access to quality data about latency, availability and customer ratings of Web services that can be directly used in the decision making process. Therefore, the network operator simply has to define the corresponding policies over these aspects.

The route planning example illustrates the application of the prototype within a mobile scenario using the Web service selection algorithm. In a similar manner other allocation algorithms can also be plugged in. This is realized by reformulating the query that is sent to the server or by using an alternative Web service interface. For example, in the case of an auction, it might be more convenient to send *Request* and *Offer* instances to the server, where the allocation is then determined by (successively) querying the knowledge base. This is addressed by the following implementation of a grid service market.

[4] `http://www.webrpc.com/`

[5] `http://www.programmableweb.com/`

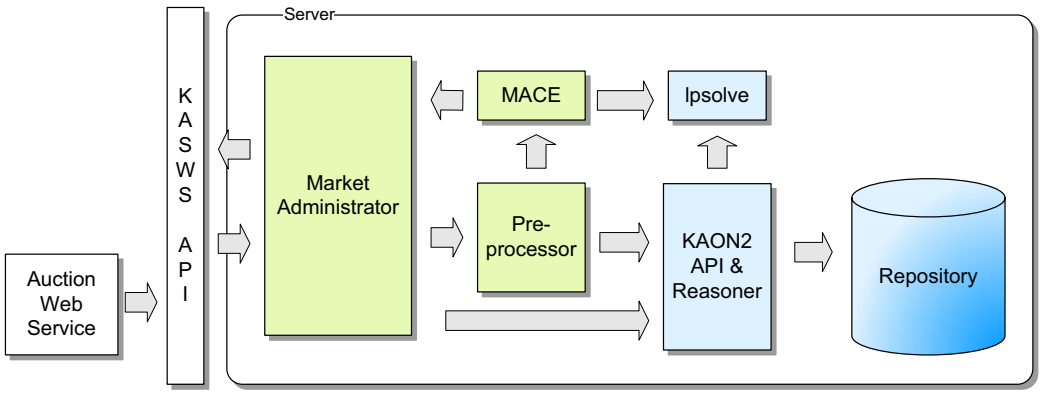

Figure 8.5: Prototype extended by auction components.

8.5.2 An Ontology-based Exchange for Grid Services

In order to adapt the allocation mechanism for grid services, two major extensions to the service repository are required: on the one hand, we have to be able to deal with limited resources which require more complex allocation mechanisms supporting combinatorial bids, and on the other hand, a contracting phase that leads to legally enforceable agreements. The former is realized by generalizing the Local Allocation Problem (Definition 5.11) to the multi-attribute combinatorial double auction (MACE) presented in [SNVW06]. This mechanism meets the requirements by allowing simultaneous trading of providers as well as requesters, and supports combinatorial as well as multi-attributive bid descriptions. However, existing implementations of this mechanism rely on pure syntactic matching of bids and thus do not support the interoperability requirement.

The problem can be addressed by an additional preprocessing step where the Multi-attribute Matching Problem (Definition 5.10) is calculated for each offer/request-combination using the matching rules contained in the repository. Thus, semantic matching is automatically used where appropriate. Based on these results, the original auction implementation can be used. Since a combinatorial double auction is usually computationally demanding and exhibits an exponential runtime with respect to the offers and requests in the market, a splitting approach is used where the overall market is divided into sub-markets that can be solved independently (compare the *TradingComponents* introduced Section 8.4).

In the following, we briefly sketch how the system is realized using the prototype described above. Figure 8.5 refines the architectural overview shown in Figure 8.1 by adding the components required for implementing the combinatorial auction. These components are the *Market Administrator*, *Preprocessor*, and the original implementation of a combinatorial auction called *MACE*. For a detailed description of the system the interested reader is referred to [LS06].

Market Administrator: The Market Administrator receives *Bid* instances from the Web service interface and adds them to the repository. In this context, each instance is assigned to exactly one *TradingComponent*. As defined by Axiom (A79), a *TradingComponent* clusters *Bid* instances into disjoint sets, which are subsequently handled by independent auction instances. Thereby, the number of bids per auction can be reduced. To avoid loosing matches, the clustering

algorithm has to make sure that all offers and requests that could potentially be matched are assigned to the same *TradingComponent*. An algorithm which splits the auction according to input and output of a service is presented in [LS06].

As discussed in Chapter 4, grid scenarios usually require a contract formation step, where agreements are transformed into legally enforceable contracts. This functionality is also provided by the Market Administrator, based on the results received from the auction component.

Preprocessor: Once a new *Bid* comes into the system, the preprocessor executes two tasks: (*i*) *semantic preprocessing* to solve the MMP for all relevant *Offer*/*Request*-combinations using Query R39, where semantic matches between attribute values are considered. (*ii*) Subsequently, the results are used to generate the bids according to the bidding syntax of the MACE system. We call this second step *syntactic preprocessing*. By transforming OWL-syntax into the bidding language of MACE, the formal semantics is lost. However, since matching of attribute values is already carried out in the semantic preprocessing step and solving the Combinatorial Auction Problem (as defined in Section 5.3.2) does not require logical reasoning any more, this does not have any disadvantages.

MACE: MACE is an implementation of the multi-attribute combinatorial double auction and solves an extended version of the Combinatorial Auction Problem. It supports only the datatype matching operators '$\leq$'/'$<$' or '$=$'/'$\neq$' and therefore a semantic as well as syntactic preprocessing step is required in order to ensure interoperability and Web-compliance.

Since the prototype returns formal contracts to the winning parties and most WS-BPEL engines provide monitoring information about the process execution (e.g. the ORACLE Business Process Manager®), which also contains information about each Web service invocation, a compliance checking algorithm as outlined in Section 7.2 can be applied. This provides some insight into the performance of the different providers that could be used, e.g., for future service selections.

After presenting a prototype that implements a semantic Web service market in this chapter, we discuss in the next chapter whether our approach adequately addresses the requirements postulated in Chapter 4.

Chapter 9

Discussion and Evaluation

In this chapter, we revisit the design requirements specified in Chapter 4, and discuss how they are addressed by our system. This step can be seen as a part of the evaluation phase in the market engineering process. In a nutshell, as described in Section 1.2, the recipe we use in this work for developing a Web service market is as follows:

1. Identify a suitable language for the market and express the required vocabulary with the chosen language.

2. Define contracting and contract monitoring mechanisms for the specified vocabulary and the chosen language.

In the first part of this chapter (Section 9.1), we focus on requirements that can be addressed by choosing an adequate representation language. Subsequently, in Section 9.2 we assess whether the vocabulary defined in the ontology provides the required expressiveness for context-dependent, multi-attributive and combinatorial bids as well as legally reliable contracts. Having addressed language and vocabulary requirements, we focus on the design requirements of the contracting mechanisms in Section 9.3. We adopt a more quantitative view, evaluating the communication efficiency, the computational tractability and the optimality of our approach by means of a simulation.

Evaluation results have been partly published in conference proceedings. The results of the performance evaluation without efficient optimization techniques are taken from [LA07] and the results using efficient optimization techniques from [LAGS07].

9.1 Choice of Language

A major contribution of this work is the development of an ontology framework for expressing Web service offers, requests and contracts. In the following, we discuss how the use of an ontology language in a service-oriented architecture facilitates Web compliance and interoperability. In fact, ontologies can be seen as the main enabler of these features.

(*R1*) Web Compliance

Throughout this work, the ontology language OWL-DL and the DL-safe fragment of the rule language SWRL is used. As discussed in Section 2.4.2, both logical formalisms are specifically designed for the Web and thus augment the facilities for expressing the meaning of terms provided by XML. OWL documents can thus be exchanged between computers having different operating systems and running different applications. By using URIs (Universal Resource Identifier) ontologies enable the unique identification of resources in the Web. URIs are one of the essential design principles of the World Wide Web. Another important feature inherited from XML is the concept of namespaces for denoting unambiguous information spaces. URIs and namespaces support disambiguation of resources having the same name and thus the modularization of ontologies into distinct modules. As there are many different domains and applications with possibly overlapping vocabularies, modularization is essential for ontologies on the Web. It enables reuse of ontologies and avoids over-commitment (see Section 4.2.1 on page 42). Since SWRL rules can be serialized with OWL/XML or OWL/RDF syntax, they are also compliant to existing Web standards.

Regarding implementation, our platform resides on top of existing Web service infrastructure and thus reuses existing W3C/OASIS standards and recommendations. Most notably, WS-BPEL is used for specifying the client process, which identifies individual services via WSDL. In order to enable dynamic assignment of ports to a WSDL port type, we rely on WS-Addressing [W3C05]. For conveying messages between participants in the system, traditional SOAP messages are used on top of the HTTP protocol. Therefore, no proprietary extensions are introduced and the system can be seen as fully Web-compliant in a sense that only existing standards and recommendations are used.

(*R6*) Interoperability

Ontologies can be seen as state-of-the-art for enabling interoperability among heterogeneous information systems over the Web. By combining XML-based syntax with a logical formalism that unambiguously captures its semantics, semantically equivalent terms can be determined automatically even if they are represented by different terms. Examples of how inferencing can be used in the matching process of services are given in Examples 2.1 and 7.1.1. This matching technique particularly addresses the common case, in which providers and requesters define their services on different levels of abstractions. Utilizing concept hierarchies in the matching process allows us to overcome this problem. Moreover, policies can be defined on an abstract level, which is especially important if not all possible alternatives are known a priori. For example, a requester could use the following axiom $\mathit{HighSecEncryption} \sqsubseteq \mathit{EncryptionMethod} \sqcap \exists \mathit{keylength}.\{> 1024\mathit{bit}\}$ to specify that she generally accepts high security encryption methods. Given this definition, all encryption methods with a shorter key length are considered inappropriate without requiring the requester to know the exact encryption mechanism beforehand. Such a mechanism is important for providers since it allows them to add new features to the service and still be "discoverable" by customers.

Another important feature of a language consisting of OWL-DL and DL-safe rules is that it allows the specification of complex mappings between ontology con-

structs (possible from different source ontologies). To some extent, simple mappings can also be determined automatically [Ehr06]. For example, using the mapping system presented in [HM05], a participant joining the Web service market may align her local ontology to ontologies in the market and, thereby enabling the matching of her request/offer to the ones already available in the market.

9.2 Expressiveness of Vocabulary

The ontology language OWL-DL and the rule language (DL-safe) SWRL has been used to define an ontology framework for Web based markets. This framework provides an expressive vocabulary for specifying Web service offers, requests and contracts. In the following, we investigate whether the framework supports multi-attribute combinatorial bids containing context-dependent preferences. In addition, we discuss how automation and a high degree of flexibility is enabled by our ontology framework.

(*R2*) Multi-attribute Descriptions

Configurability of products makes high demands on the bidding language in terms of the exponential amount of possibilities that have to be captured by a bid. We address this problem in Chapter 5 by introducing a multi-attribute Web service model, in which the *CPO:Attributes* capture the different dimensions by which a service can be differentiated. In addition, prices are expressed via functions over these attributes, which enables efficient representation of highly configurable services. One of the major contributions of this work is to specify a vocabulary that enables expressing this model with OWL and DL-safe rules. This is realized by the Core Policy Ontology and the Core Ontology of Bids introduced in Chapter 6. These modules allow the market participants to express their bids without enumerating all acceptable trades. A detailed discussion how this approach features communication efficiency can be found in Section 9.3.

(*R3*) Combinatorial Requests and Offers

Some allocation mechanisms require the specification of sub- and superadditive valuations over combinations of Web services. The Core Bidding Ontology provides the vocabulary for expressing such combinatorial offers and requests. This is realized by the concept *COB:BundleBid* in the ontology, which comprises *COB:ANDBids*, *COB:ORBids* and *COB:XORBids*. *COB:ANDBids* are the ontological primitives for modeling complementarities (i.e. superadditivity) and *COB:XORBids* for modeling substitutes (i.e. subadditivity). Hence, we can express arbitrary OR/XOR-formulae of the form $(b_1 \vee b_2)$ or $(b_1 \oplus b_2)$, where b_1 and b_2 may represent *COB:AtomicBids*, *COB:ANDBids*, or recursively again *COB:ORBids* or *COB:XORBids*. While XOR-bids alone can represent all valuations [Nis00, Prop. 3.2], for some cases an exponential number of contained *AtomicBids* is required. For example, additive valuation on m items requires at least 2^m *COB:AtomicBid* instances, whereas a representation of m *COB:AtomicBids* is possible using m *COB:ORBids*. Thus, a combination of OR- and XOR-bids is required

for a succinct representation of bids. By featuring arbitrary OR/XOR-formulae, this is provided by the Core Ontology of Bids.

(*R4*) Context Dependency

As discussed in Chapter 4, especially in mobile scenarios preferences of a user may depend heavily on the current context, which could include the current time, location, activity, etc. Although these properties can be seen as typical attributes in the decision problem, ontologically we distinguish between characteristics of the Web service and characteristics of the environment in which the decision is taken. While the former are modeled as *CPO:Attributes* forming a *CPO:Configuration*, the latter are modeled as *CPO:ContextDimensions* forming a *CPO:Context*. This distinction is important, since the *CPO:Context* is usually given, while the *CPO:Configuration* can be freely chosen. Preferential dependency between *CPO:Attributes* and *CPO:ContextDimensions* can be modeled in the Core Policy Ontology by defining that certain *Preferences* are only applicable in a certain *Context*. This means scoring and pricing functions can still be modeled additively, while still capturing their dependency on the context.

(*R7*) Automation

Automation requires machines making decisions without human intervention. As a first condition, the machine has to be able to interpret the different decision alternatives and criteria, which in turn requires a machine-understandable description of them. In our approach, this is provided by an underlying ontology language. As a second condition, the machine needs information on how to make decisions in a certain situation. In our approach this information is provided by means of *CPO:PolicyDescriptions*. They formalize the guidelines of the system owner in a machine-interpretable format and thus enable autonomous decisions by the system. We introduced the notion of goal policies which capture hard constraints that have to be fulfilled and thus restrict the decision space of the system. Moreover, we introduced utility function policies as a means for representing preferences. By means of utility function policies the machine is able to arrange the alternatives in a preference structure, which enables the machine to make optimal decisions on behalf of the user.

(*R8*) Flexibility

Due to the open nature of the Web service environment, flexibility is required in order to adapt the system to changing Web service types, participants, or other environmental conditions. Such adaption might require rather small changes, such as introducing new *CPO:Attributes*, or might involve a considerable redesign of a mechanisms, e.g. changing from Web service selection to an auction-based approach. Our system facilitates both types of changes through declarative matching and allocation rules. By leveraging declarative languages, we define the parts that may change frequently using declarative rules. Consequently, important parts of the algorithms are defined by means of DL-safe rules that can be easily changed during runtime of the system. For example, Rules (R3), (R4) and (R29) declaratively

define how certain *CPO:Attributes* should be matched. Once new *Attributes* are introduced, a corresponding matching rule is added or, alternatively, the concept of *Attribute* hierarchies is used to define the type of *Attribute* and thus also the way how it should be matched. For example, Rule (R30) defines the matching rule for *QoSAttributes*. New quality of service attributes can be introduced as specialization of *QoSAttribute* and, in doing this, they are automatically matched with the corresponding match predicate. Similarly, declarative allocation rules are defined that specify how requests and offers are allocated to each other. For example, in case of Web service selection, the allocation rule can be expressed directly in the query (Query (R39)) formalizing the Local Allocation Problem.

9.3 Design of Mechanisms

After having discussed whether the ontology language and the vocabulary expressed via this language meets our requirements, we now focus on the design of the mechanism in terms of computational tractability, communication efficiency and optimality. In contrast to the previous requirements, they are not amendable to a qualitative analysis. Instead, we do a quantitative analysis using the indicators *storage space*, *performance* and *completeness of the results*. Although a theoretical analysis of the time and space complexity are a first indication for practical applicability of the mechanism, to assess communication efficiency and computational tractability an absolute measure of performance and required storage space is needed that also includes possible side-effects that discloses potential software (development) problems. The latter is important since an efficient algorithm is easily implemented in an inefficient way, which may lead to considerable performance problems. In order to judge the optimality of the results, we have to investigate whether the allocation algorithm is complete in a sense that all relevant matches are considered, and whether the allocation optimizes the goal of the market.

In order to measure the compactness of our bid representation, the performance of our matching algorithms and the completeness of the results, a simulation is conducted in the following. The concrete setup of the simulation is described in Section 9.3.1, before we present insights into the results of the simulation. The compactness of the bid representation is discussed in Section 9.3.2, the performance of service selection in 9.3.3 and the completeness of the results in Section 9.3.4. Finally, the results are interpreted and discussed with respect to the requirements in Section 9.3.5.

9.3.1 Simulation Setup

Testing the system by means of a simulation first requires to generate realistic test data, which in our case breaks down to service descriptions and bids involving these descriptions. Reviewing related literature on Web service selection algorithms reveals a lack of suitable service descriptions, especially, descriptions involving multiple attributes. In fact, we found only two approaches that have been quantitatively evaluated. The first work presented in [LNZ04] is evaluated by means of a hypothetical phone service market. They consider two providers which differ in price (specified in Dollar), transaction support (“yes”/“no”), time out period (microsecond), compensation rate (percent), penalty rate (percent), execution du-

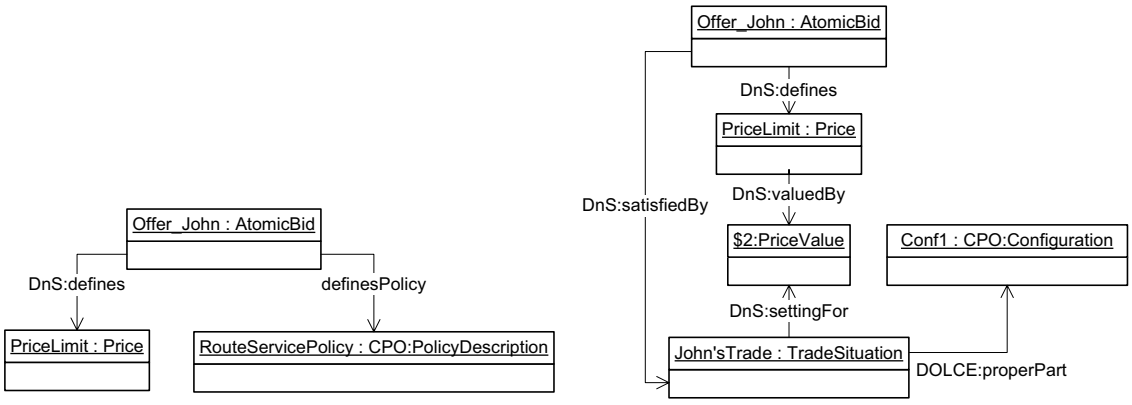

Figure 9.1: Comparing policy-based and enumeration-based representation of *Bids*.

ration (micro-second) and reputation (rank value in $[0;5]$). The concrete attribute values of their services are shown in the table below (Provider "ABC" and "BTT"). The second work containing a quantitative analysis of the presented selection algorithm is [WVKT06]. They use also the same dataset, but add additional hypothetic providers, which are denoted by "A1" and "A2" and are shown in row 3 and 4 of Table 9.1. Both approaches do not consider customizable services, but focus only on selecting one provider for a given request without considering different configurations. Due to the small size of the dataset, obviously time and space complexity have not been an issue.

name	price	transaction	time out	compensation	penalty	execution	reputation
ABC	25	"yes"	60	0.5	0.5	100	2.0
BTT	40	"yes"	200	0.8	0.1	40	2.5
A1	28	"yes"	140	0.2	0.8	200	3.0
A2	55	"yes"	180	0.6	0.4	170	4.0

Table 9.1: Experiment dataset obtained from [LNZ04, WVKT06].

In order to adopt the dataset to our requirements, we allow one provider to offer several configurations within one single offer. This is realized by means of the Core Ontology of Bids. Hereby two important scenarios can be distinguished:

Enumeration-based Approach: This is the approach taken in most current Web service description frameworks. It follows the idea that all acceptable configurations are enumerated in the bid. Using the Core Ontology of Bids, this scenario can be modeled by referencing the acceptable *COB:TradeSituation* instances directly from the *COB:BidDescription*. This approach is illustrated in the right diagram of Figure 9.1. For each combination of *COB:Bid* and *COB:Configuration* the corresponding *COB:PriceValue* is introduced. For the enumeration-based approach, the following adaption of Query (R39) can be used to get all offers and configurations ranked according to the requester's utility.

(R47) *PREFIX ex: < http://example.org/ns# >*
SELECT ?O, ?C, ?U WHERE {
?O rdf:type COB:Offer ; DnS:satisfiedBy ?TO .
?TO DOLCE:properPart ?CO ; DnS:settingFor ?PO .
ex:RequestUserAnnika DnS:satisfiedBy ?TR .
?TR DOLCE:properPart ?CR ; DnS:settingFor ?PR .
?CO compare ?CR . EVALUATE ?U := swrlb:subtract(?U, ?PR, ?PO) .
}ORDER BY DESC(?U)

Policy-based Approach: Policy-based bid specification is illustrated on the left hand side of Figure 9.1. According to Definition 5.4 and 5.5, no explicit modeling of trades is required here. Instead, we introduce a *PolicyDescription* which implicitly defines the *PriceValue* by means of Rules (R14)-(R19). For modeling policies we consider the worst case scenario, in which all *Functions* are represented by *PointBasedFunctions*. *PiecewiseLinearFunctions* and *PatternBasedFunctions* can be seen as more efficient in terms of compactness as well as performance. For retrieving a set of offers ordered according to the requester's preferences we can directly use Query (R39).

In a first simulation, we gradually increase the number of configurations provided by the offers and measure the storage capacity that is required to represent them. For each scenario, instances of offers, requests and contexts are randomly generated using uniform distributions and stored in the knowledge base. Thereby, we are able to address the following question:

Question A: Does the policy-based bid representation provide more efficient means for representing highly configurable bids than an enumeration-based approach?

However, communication efficiency that can be realized through compact bid representation is not the only goal the system should meet. We also have to provide means to efficiently query *Bids* from the repository and thereby execute the contracting mechanisms. As the way bids are represented might also considerably influence the performance of matching and allocation, in a second step the query answering time for the Web service selection algorithm triggered either by Query (R47) or (R39) is measured. Therefore, we send queries to the repository and measure the time that elapses until we receive the ranked list of configurations and offers. We take the average of 20 queries. To avoid network delays, the simulation is performed on a single computer. Thereby, we are able to address the following questions:

Question B.1: How does the evaluation of bids scale if bids are represented by enumeration of trades or by means of policies?

Question B.2: Can we improve policy-based bid evaluation in terms of performance by replacing matching algorithm [V1] with [V2] or [V3]?

Question B.3: How does the performance of preference- and context-aware selection strategies compare to suboptimal strategies? How expensive is optimality?

In order to evaluate how semantic technologies influence the optimality of the approach, we measure the completeness of the Web service selection algorithm and investigate how the completeness influences the utility of the participants. In contrast to the dataset shown in Table 9.1, we generated normalized prices and scores in the range $[0,1]$. Thus, the maximal utility that can be realized by a requester in a selection is a utility of 1 and the worst a value of -1. Recall, for providers we assume indifference between the different configurations, i.e. the difference between the reservation price and internal costs is identical for all configurations. Since the importance of semantic matching depends on the structure of the ontology, we introduce an additional attribute 'ServiceType'. The attribute values are modeled using three different ontologies. Each ontology has a different structure in terms of number of concepts and depth of hierarchy. This allows us to compare the completeness of our selection algorithm using syntactic and semantic matching and to draw conclusions how semantic technologies can improve the results in a Web service market.

Question C.1: Does semantic matching improve the completeness of the results? In which cases is it particularly important?

Question C.2: How does the completeness of the results influence the utility that can be realized by the market participants?

In the remainder of this section, the compactness of the bid representation is evaluated by addressing Questions A (Section 9.3.2), the performance aspects by answering Questions B (Section 9.3.3) and the completeness of the results by discussing Questions C (Section 9.3.4). Finally, these evaluation results are discussed with respect to the requirements communication efficiency, computational tractability and optimality in Section 9.3.5.

9.3.2 Compactness of Bid Representation

In this section, we investigate whether the policy-based modeling approach facilitates compact representation of bids and therefore improves communication efficiency in the market. We compare the policy-based bid specification with a baseline approach that enumerates all provided or requested trades in terms of storage capacity required for representing bids. We evaluate settings that differ in the number of offers in the repository and in the number of configurations per offer. The difference between the two scenarios is illustrated in Figure 9.2.

Question A: Does the policy-based bid representation provide more efficient means for representing highly configurable bids than an enumeration-based approach?

Figure 9.2 shows the number of bytes that are required to represent a multi-attribute offer using the Core Ontology of Bids.[1] Hereby, the dotted lines represents the required bytes using the policy-based approach and the solid lines the required bytes

[1] The detailed results are available in the appendix (Table A.1).

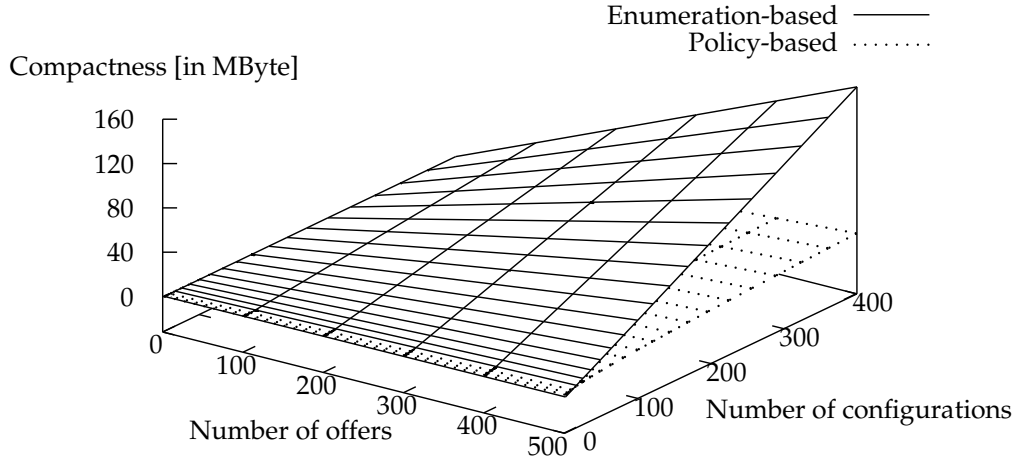

Figure 9.2: Compactness of bid representation.

using the enumeration-based approach. The number of offers in the repository is increased step-by-step from 1 to 500 and the number of possible configurations per offer from 1 to 400. Obviously, already with two configurations the policy-based approach is more compact than the enumeration-based approach for expressing the same information. If we further increase the number of configurations per offer, the number of axioms required for the enumeration-based approach increases linearly, which leads to an overall space complexity of $\mathcal{O}(|O||C|)$ for the enumeration-based approach. Because the set C of configurations is defined as $C = \prod_j A_j$ (Section 5.1), the number of configurations grows exponentially with the number of attributes. We thus get a space complexity of $\mathcal{O}(|O|n^{|A|})$, where n represents the maximal number of attribute values of an attribute ($n = \max_j |A_j|$). For the policy-based approach, in contrast, storage size increases linearly with the number of attribute values $\sum_j |A_j|$ and thus this approach exhibits a logarithmic space complexity with respect to the number of configurations. In addition, no *PriceValue* instance for each Offer and *Configuration* instance has to be introduced. The policy-based approach leads therefore to an overall space complexity of $O(|O| + log(|C|))$ and $O(|O| + n|A|)$. Note that this is the worst case complexity which holds only for discrete attributes. Continuous attributes can be specified even more efficiently with the policy-based approach using *PiecewiseLinearFunctions* or *PatternBasedFunctions*, whereas a representation using enumeration is not possible at all.

Considering the absolute number of bytes required for representing bids, the problem of enumeration-based bid representation for large scenarios becomes clear. For example, assuming a scenario where 1000 providers offer a service with 100 configurations the size of the repository amounts to 77 MBytes (Table A.1). Using policy-based bid generation only about 24 MBytes are required, which makes the scenario much easier to handle. If *PointBasedFunctions* are replaced, e.g., by *PatternBasedFunctions* a further improvement even below 7 MBytes is possible. Therefore, one can conclude that policy-based bid specification considerably improves scalability of bid representation in the presence of highly configurable products.

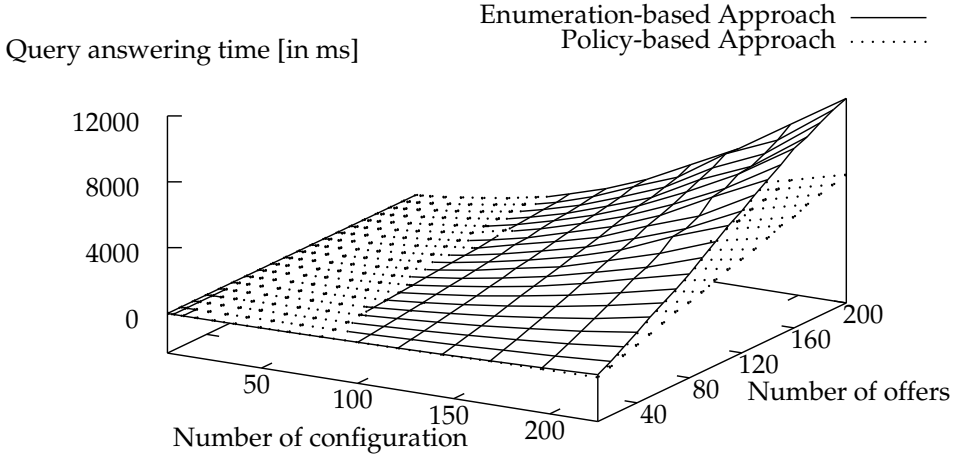

Figure 9.3: Comparing the evaluation performance of enumeration-based and policy-based bid representation.

9.3.3 Performance

In the second step, we investigate the performance of bid evaluation. As discussed above, this is realized by measuring the time between sending Query (R39)/(R47) and receiving the result set (without considering the communication overhead). In the following, we first compare the query answering time of the enumeration-based (Query (R47)) and the policy-based approach (Query (R39)).

Question B.1: How does the evaluation of bids scale if bids are represented by enumeration of trades or by means of policies, respectively?

Generally, the performance of the system depends on the following factors:

- the *number of Offers* that have to be evaluated for each *Request*;
- the *number of Configurations* that are contained in the Offers;
- and the *type of Attributes* that constitute the *Configuration* (e.g. the datatype or structure of the ontology describing the *AttributeValue*, the *match*-predicate, etc.).

Since the same *match*-rules are used for matching of attribute values in both scenarios, the third factor does not influence the comparison between the enumeration- and policy-based approach. We thus restrict our following analysis of the two scenarios on a changing number of *Offers* and *Configurations* and use only "flat" matching rules for comparing the attributes. Performance evaluations of query answering with more complex matching rules is a complementary question and falls back to the performance of standard description logic operations such as subsumption checking, which have already been elaborated, e.g., in [MS06] for KAON2.

Since the time complexity of query answering is completely predefined by the ontology/rule language as well as the corresponding reasoning algorithms (implemented in KAON2), theoretically no difference between the complexity of the two

approaches exists. However, as KAON2 is based on the principle of "graceful degradation" [MSS05] meaning that a user only pays the performance penalty for features used, in practice their can be considerable performance differences. We have thus conducted a simulation to measure the performance in practice. Figure 9.3 illustrates the average query answering time depending on the number of *COB:Offers* and the number of *CPO:Configurations* per *COB:Offer*. The complete dataset is listed in the appendix (Table A.2). For simplicity, we assumed that all provider offer all possible configurations.

As depicted in Figure 9.3, while query answering in case of enumeration is extremely fast for small scenarios (i.e. less than 100 configurations or 50 offers), the lookup time of prices in the knowledge base increases considerably with an increasing size of the A-box. For example, the service selection with 200 *Offer* instances each referring to 20 *Configuration* instances can be done within 194 ms. However, executing the selection with 225 configurations per offer requires already 10 sec. and, according to Table A.2, over 6 minutes are required for a setting with 800 offers and 600 configurations. This is clearly too long for service election at runtime.

In case of policy-based descriptions, similar performance characteristics can be identified, however, on a lower level. In small scenarios, the policy-based approach is outperformed by the enumeration-based approach, while for medium and large scenarios the former performs considerably better. Considering 200 offers each with 20 configurations the selection can be done in 301ms, whereas in the case of 800 offers and 600 configurations the selection can be done within 2 minutes. This is an improvement of 4 minutes compared to the enumeration-based approach. As a reason for the slowdown, the exponential increase of *CBO:TradeSituation* instances (in particular *CBO:Price* and *CBO:PriceValue* instances) in the knowledge base can be identified. As discussed in Section 9.3.2, this is not the case for the policy-based approach. However, in small scenarios the evaluation of Rules (R14) and (R19) is more expensive than looking up the right *PriceValue* instance in the repository. As discussed in [LA07], this issue can be addressed with intelligent caching algorithms that, e.g., explicitly store price information of frequently queried configurations. Using such a strategy, the costly policy evaluations have to be done only once for the first query.

Although the policy-based approach does provide a considerably improved scalability compared to the enumeration-based approach, it might still not be sufficient for dynamic service selection at runtime. For example, a mobile phone user usually does not want to wait for 2 minutes until a suitable route planner is found and invoked. Therefore, in Section 7.1.1, we introduced two alternative matching variants that are applicable if not a full ranking of all configurations is required but the selection is restricted to the best provider with the best configuration

Question B.2: Can we improve policy-based bid evaluation in terms of performance by replacing matching algorithm [V1] with [V2] or [V3]?

Up to now, we used the matching variant [V1] introduced in Rule (R34) for solving the Multi-attribute Matching Problem. While featuring a full ranking of configurations, this approach requires to represent all configurations as instances in the knowledge base and therefore does not support continuous attributes. In addition, searching this configuration space is exponential in the number of attributes, which leads to the unfavorable runtime discussed above.

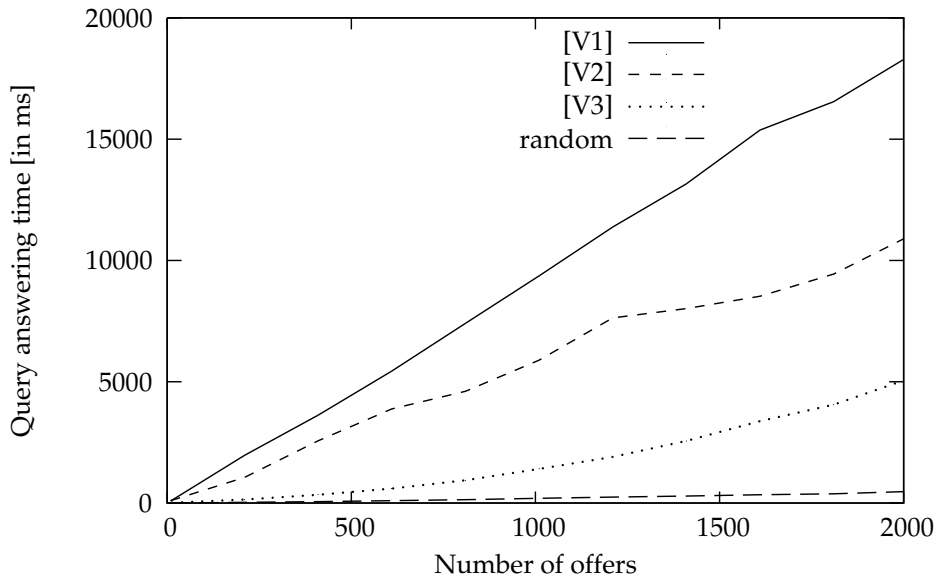

Figure 9.4: Performance with 100 configurations and varying number of offers.

Therefore, in Section 7.1.1 two alternative matchmaking variants are introduced, which address the problem of the exponential search space by restricting the allowed utility function policies to an additive form. As discussed in Section 5.1, although preferential independency does often not hold perfectly between the different attributes, additive functions do still provide a good approximation [RN03]. Variant [V2] utilizes the additive structure by optimizing each attribute separately and then aggregates the optimal values. Thereby, the search space can be reduced from $\prod_{l=1,\ldots,n} |A_l|$ in case all configurations have to be considered to $\sum_{l=1,\ldots,n} |A_l|$ in case the attributes are optimized. Variant [V3] further improves the optimization through the use of an efficient linear programming solver. We distinguish between [V2] and [V3] in order to find out which share of the expected improvement in performance is due to the additive structure of the policies and which part is due to the advanced tool support.

In Figures 9.4 - 9.6, the three variants of the matching predicate are evaluated for varying number of offers in the knowledge base and varying number of configurations per offer. In addition, they are compared to a baseline algorithm which randomly selects an offer and configuration from the repository. Figure 9.4 shows the change in runtime of a query if we stepwise increase the number of offers in the repository. In this scenario we assume all offers to provide 100 configurations. In Figure 9.5, a larger setting is considered where each offer provides 900 configurations. Finally, in Figure 9.6, we assume a constant number of offers in the repository and gradually increase the number of configurations per offer. A detailed list of the simulation results can be found in Table A.3.

In the first setting (Figure 9.4) with 2000 offers in the repository, [V2] reduces the runtime compared to [V1] from 18 to 11 seconds and in the second setting (Figure 9.5) from 477 to 41 second. [V3] further reduces the runtime in this setting to 5 and 13 seconds, respectively. While in the first setting where we deal only with 100 configurations the slowdown is only modest, [V1] does not scale when moving to more complex settings. This is in line with the experiment discussed in the

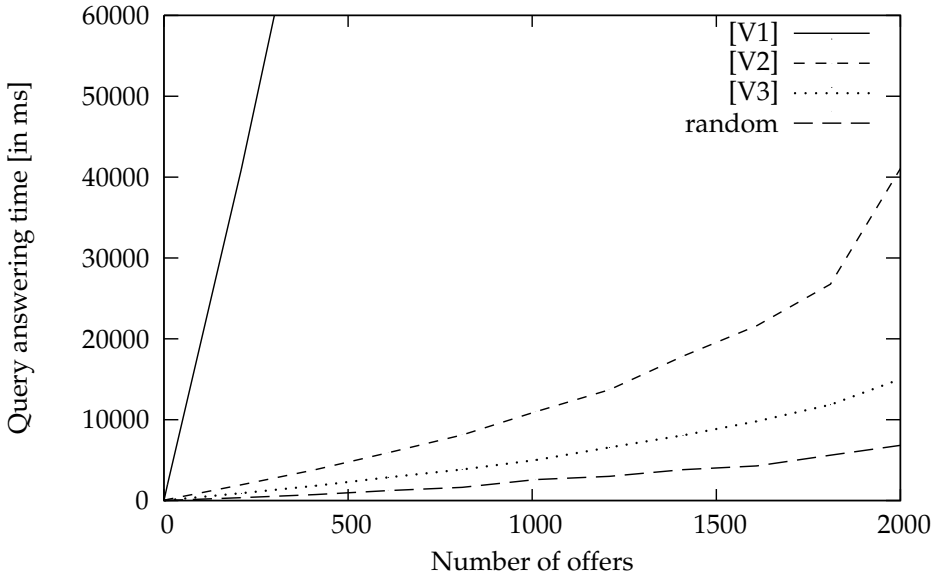

Figure 9.5: Performance with 900 configurations and varying number of offers.

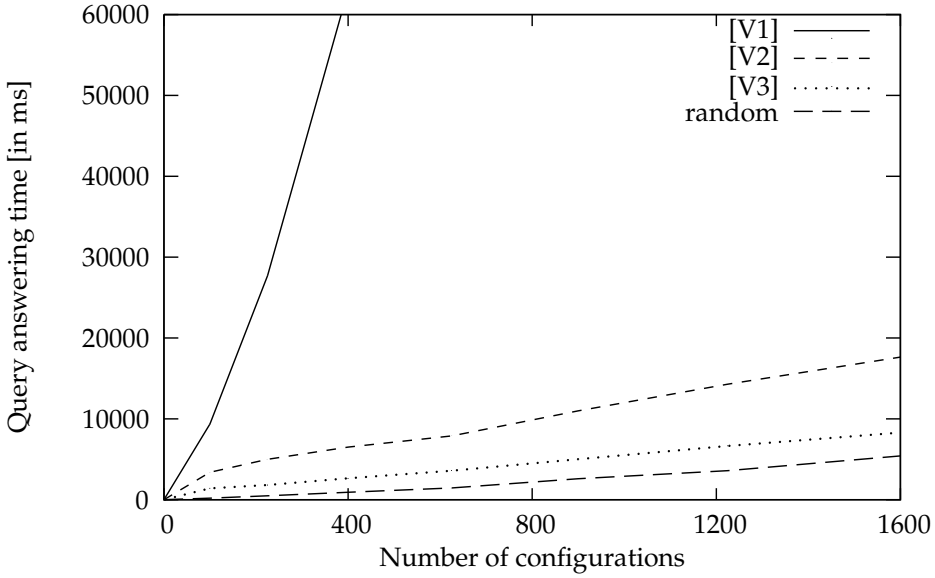

Figure 9.6: Performance with 1010 offers and varying number of configurations.

previous section. Regarding [V2] and [V3], the results indicate a considerable improvement of performance, especially, when moving from [V1] to [V2]. This means that the assumption of additive policies seems to considerably simply and accelerate the service selection process. Using an optimized linear programming solver further improves the performance, but also requires to augment the description logic reasoner with an additional tool that is invoked via a built-in predicate of the reasoner. The dominance of variant [V2] and [V3] over [V1] can be recorded almost for all settings, except very small settings in which the number of offers is below 10 and the number of configurations below 50. In this case, initializing the solver could be more time-consuming than directly finding the best provider using mechanism [V1].

Question B.3: How does the performance of preference- and context-aware selection strategies compare to suboptimal strategies? How expensive is optimality?

In order to investigate the additional time required for optimal selection of services, we compare the approaches that result in an optimal selection [V1], [V2] and [V3] with a baseline algorithm that randomly selects an offer and corresponding configuration from the knowledge base. While the random algorithm considerably outperforms all optimal algorithms in all settings, it may also lead to a huge utility loss for the participants. Assuming a uniform distribution of the prices and scoring values in $[0,1]$ and a reasonable number of offers (> 50), an optimal algorithm leads to a utility of almost 1 while the random algorithm results only in an average utility of 0 [LAGS07].

Comparing the fastest matching approach [V3] to the random algorithm, the cost of optimality is rather moderate. In particular, considering the third setting with 1000 offers and 1600 configurations per offer, there is only a slowdown by 35% when moving from random selection to [V3] (Figure 9.6). Comparing this number to smaller settings, we can observe much greater slowdowns which, at first glance, seems contradictory. However, this observation can be explained by the fact that for large-scale scenarios (more that 1000 offer per configuration) query answering becomes the predominant factor compared to the optimization. Since query answering is required for both algorithms, variant [V3] and random selection converge.

9.3.4 Completeness of Results

Obviously, the allocation determined in the market should not be chosen randomly, but should serve a certain goal. Goal of the Web service selection algorithm is the optimization of the requester's utility. By solving the Local Selection Problem always the service is selected that provides a utility maximizing configuration for a given request. Hence, our selection algorithms can be considered as sound. However, the set of offers that are considered in the LSP depends on the way attribute values are matched. Obviously, if more matching offers are found there is a higher chance of finding a better offer. As discussed, in a heterogeneous environment syntactic matching is often not sufficient to ensure finding all relevant offers. Therefore, we evaluate in this section how the semantic matching can improve the completeness of the matching results, while ensuring soundness of the matching approach. In doing this, we introduce the attribute 'ServiceType' which captures the main functional-

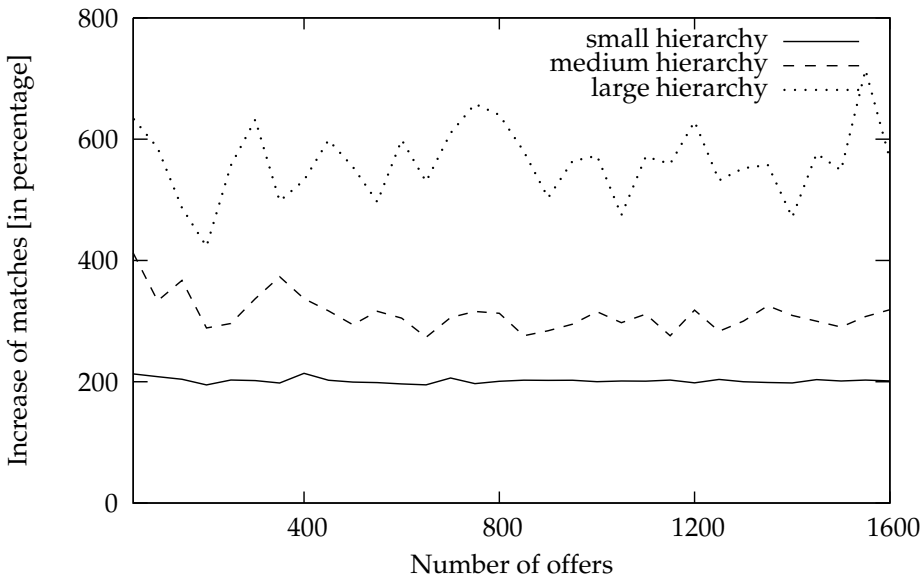

Figure 9.7: Increase in number of matches enabled by semantic matching.

ity of a service described via a service taxonomy (e.g. as given in the MIT process handbook [MCP+99]).

Question C.1: Does semantic matching improve the completeness of the results? In which cases is it particularly important?

In order to find out whether semantic technologies can improve the matching in the market, we introduce a matching rule that uses subsumption checking for comparing the attribute values of 'ServiceType' (compare Rule (R4)). This approach is still sound since all service types that are more specific than a requested service type are also relevant to the request. We compare the completeness of the results derived by the semantic matching approach with a matching algorithm that does not utilize the semantic information contained in the hierarchy, i.e. string matching of attribute values. The results are shown in Figure 9.7. It shows the average percentage of additional matches depending on the number of offers in the repository and on the structure of the ontology used for describing the service type taxonomy. We used the three different ontologies:

- An ontology from which a *small hierarchy* of service types can be inferred. The hierarchy contains 3 concepts and a depth of 1 subclass relation.
- An ontology from which a *medium hierarchy* of service types can be inferred. The hierarchy contains 30 concepts and a maximal depth of 3 subclass relations.
- In ontology from which a *large hierarchy* of service types can be inferred. The hierarchy contains 197 concepts and a maximal depth of 5 subclass relations.

The results indicate that the larger an ontology is the more additional matches can be found by means of the semantic matching approach. While the absolute number of additional matches increases with the number of offers in the repository (Table

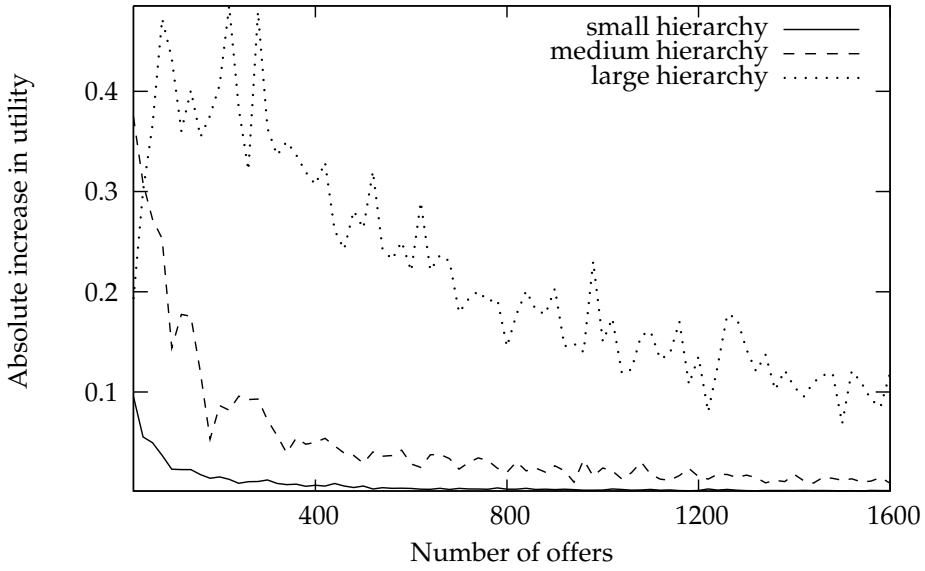

Figure 9.8: Absolute gain in utility through the use semantic matching.

A.4), the relative increase, as percentage of matches in total, seems to be independent of the repository size. For small hierarchies the number of matches increases by 200% in average, for medium hierarchies by about 300%, and for large hierarchies by over 500%. Of course, no negative numbers can be observed, since no matches are lost when moving from the syntactic to the semantic matching approach.

However, the question is how important are these additional matches for a requester. Do they really improve the allocation?

Question C.2: How does the completeness of the results influence the utility that can be realized by the market participants?

In order to investigate this question, we measure the utility of the requester in the different settings (Table A.5). The requester's utility is given as the surplus realized with the utility maximizing configuration and offer. The surplus is defined as the difference between score and price. As discussed above, in our simulation we generate prices for offers and requests in the range $[0,1]$ using a uniform distribution. Thereby the utility is measured in the range $[-1,1]$. Figure 9.8 shows the absolute gain in utility when moving from the syntactic to the semantic matching algorithm. The results indicate that more additional matches (as in the case of large hierarchies) also lead to a higher utility gain. This behavior is also expected since the probability that a better offer is found increases with the number of additional matches. However, this behavior can be observed mainly in settings where the ratio of offers per possible service type is rather low. In such sparse repositories semantic matching is more important than in settings where already syntactic matching leads to a substantial number of suitable offers. This is intuitively clear since in the latter cases the probability is also rather high that a very good offer is contained in this set. Very good offers can hardly be improved by finding additional matches, whereas in cases with only a few matches, additional matches often lead to a considerable improvement in utility. This means that semantic matching is especially important if service

differentiation can be very high, while the number of provided services is rather low.

9.3.5 Discussion of Simulation Results

In general, the results of the evaluation with respect to compactness and performance are promising since we are able to reduce the storage capacity from $\mathcal{O}(|O|n^{|A|})$ to $\mathcal{O}(|O| + n|A|)$ by means of utility function policies and the fastest approach allows ranking up to 2000 offers with a reasonable number of configurations below 15 seconds. Considering the fact that all these 2000 offers fulfill the mandatory conditions that can be defined in the FILTER clause of Query (R39), such a scenario can be considered as very large. Thus, the policy-based bid representation presented in this work meet the requirements communication efficiency and computational tractability.

In this context, it is important to note that we analyzed a worst-case scenario, in which all offers provide all possible configurations and all attributes of a configuration are discrete. Optimization on discrete attributes is more time consuming than on continuous attributes, because techniques like differentiation are not applicable. Therefore, we expect better performance in a real-world use case. For example in our mobile scenario, only up to 20 different route planning providers might be available, whereas the number of possible configurations may easily exceed 1000. However, it is unlikely that all of them are offered by all providers.

As a further result of our performance study, it becomes clear that providing only expressive means for modeling preferences as done in [GP03, OVSH06] is not sufficient. It also must be ensured that the preferences allow for an implementation of efficient matching/allocation algorithms. Comparing the results for [V2] and [V3], we can identify the absence of the additivity assumption as the major source of complexity (improvement from [V1] to [V2]). Using an efficient implementation for solving the optimization problem provides a relatively minor improvement in performance (improvement from [V2] to [V3]). Therefore, in many cases, especially if service selection has to be done at runtime, restricting the expressiveness of the bidding language is a viable way to considerably increase performance. In addition, additive functions are easier to specify for a user and even if preferential independency does not hold exactly, they often provide a good approximation [RN03].

As we rely on purely "flat" matching predicates in our evaluation, we factored out the overhead introduced by complex matching rules (such as complex DL-inference predicates). However, since the execution of these complex matching rules is done in typically rather small domain ontologies (e.g. the Service Category Ontology or the Geographic Ontology shown in Figure 7.1), we do not expect a huge slowdown. In particular, the complexity of matching is independent from the number of offers and configurations. Analyzing the performance loss that is caused by more complex reasoning tasks depends on the axioms used in the ontology and is beyond the scope of our evaluation. For a detailed discussion of this aspect see [MS06].

A further conclusion from the evaluation is that in some very demanding settings, selection algorithms are not feasible at all. Therefore, in such cases reducing the set of offers is crucial. This can be realized by adding additional mandatory conditions through FILTER conditions (e.g. derived from goal policies). A way to

Approach	(R1) Web-compliance	(R2) Multi-attribute	(R3) Combinatorial	(R4) Context	(R5) Comm. Efficiency	(R6) Interoperability	(R7) Automation	(R8) Flexibility	(R9) Optimality	(R10) Tractability	(R11) Legal Reliability
OWL-DL/DL-Safe-Rules	✓	-	-	-	✓	✓	✓	-	-	-	-
WS-Standards	✓	-	-	-	-	✓	-	-	-	-	-
Core Policy Ontology	-	✓	-	✓	✓	-	✓	-	-	✓	-
Core Ontology of Bids	-	✓	✓	✓	-	-	✓	-	-	✓	-
Core Contract Ontology	-	-	-	-	-	-	✓	-	-	-	✓
Declarative Matching Rules	-	-	-	-	-	-	-	✓	-	-	✓
Semantic Matching	-	✓	-	-	-	✓	-	-	✓	-	-
Integration of LP-techniques	-	-	-	-	-	-	-	-	✓	✓	-

Table 9.2: Requirements and the approaches to address them.

realize this is to automatically exclude all attribute values leading to a score below a certain threshold. If the number of offers is too high, the threshold can be increased automatically.

Regarding the completeness of the results, the evaluation has shown that semantic technologies can considerably improve matching in a sense that the completeness (i.e. high recall) can be increased, while retaining sound results (i.e. high precision). Although this is an observation that is valid for all settings, the evaluation has shown that additional matches do not necessarily lead to a higher utility for the participants. Particularly in settings with highly differentiated services and a rather low number of available offers semantic matching considerably improves the results. Therefore, one can argue that with increasingly customizable service offers there will be also an increasing need for semantic technologies.

9.4 Conclusion

In this chapter, we discussed whether the design, embodiment and implementation of our semantic Web service market meets the requirements stemming from a grid, mobile and enterprise services scenario. Table 9.2 summarizes again how we addressed the individual requirements.

Web compliance is addressed by the use of languages on top of current Web standards, namely, OWL and SWRL. Moreover, our platform also resides on top of existing Web service standards, in particular WSDL, SOAP, and WS-BPEL. This approach allows seamless integration with current Web and Web service technologies. Based on the ontology language, a framework consisting of several ontology modules has been presented in order to define a common vocabulary for the semantic Web ser-

vice market. The vocabulary for *multi-attribute descriptions* is provided by the Core Policy Ontology and the Core Ontology of Bids. By featuring utility function policies the Core Policy Ontology enables the *efficient representation* of highly configurable offers and requests. In addition, the Core Ontology of Bids provides means for expressing *combinatorial bids* and the Core Policy Ontology features *context-dependent* policy specifications, i.e. policies can be defined as applicable only under certain conditions. In order to provide the *interoperability* required in heterogeneous open markets, we rely on standardized languages and realize the matching of bids by means of a semantic matching approach that is featured by the underlying ontology language. Besides handling interoperability, semantic matching is also important to address the *optimality* requirement by ensuring complete results. *Automation* of the contracting process is supported by formal, machine-interpretable descriptions of the market information and by the concept of policies, which enable a human user to define guidelines how the system should behave. Furthermore, we have shown how the ontology Core Ontology of Bids can be used to design a contracting mechanism featuring automated selection of Web services and the Core Contract Ontology to automated contract execution and monitoring by connecting contractual information with the business interactions they govern. The *flexibility* required in a distributed, heterogeneous environment is facilitated by declarative matching rules, which can be used to adapt the contracting mechanism to different domains and applications. By combining this declarative logic-based matching rules with optimization methods, such as linear programming, we are able to select the optimal service in a reasonable time; thereby, providing a *computationally tractable* mechanism. This mechanism can be seen as the basis for more complex allocation mechanisms such as auctions. In order to provide *legal reliability* for the market participants, we proposed to augment contracts formalized by means of the Core Contract Ontology with a manually closed umbrella agreement. This approach features some degree of automated legal reasoning, while ensuring full legal reliability.

In the following chapter, we discuss how other approaches address these requirements and how our approach compares to them.

Part IV

Finale

Chapter 10

Related Work

In this chapter, we provide an overview of related work and discuss how it compares to the approach taken in this work. We classify related work according to two broad categories. In Section 10.1, we first concentrate on the knowledge representation within Web service markets. In this context, we analyze to which extend other approaches meet the language-specific requirements outlined in Section 4.2.1. In Section 10.2, we address related approaches to contracting and compliance monitoring mechanisms and analyze them with respect to the mechanism-specific requirements introduced in Section 4.2.2.

10.1 Knowledge Representation in Web Service Markets

In this section, we present various existing approaches applied in electronic markets for modeling buyer preferences, seller offerings and electronic contracts. The approaches are discussed with respect to the requirements (*R1*) Web-compliance, (*R2*) Multi-attribute Descriptions, (*R3*) Combinatorial Requests and Offers, (*R4*) Context dependency, (*R5*) Communication Efficiency, and (*R6*) Interoperability. Table 10.2 summarizes the different streams of work in terms of which of the requirements they support (indicated by check marks).

10.1.1 Electronic Data Interchange

One of the first attempts to exchange order information within electronic markets were the Electronic Data Interchange (EDI) protocols (e.g. EDIFACT [DIN98], X12 [Ame06]), which serialize request, offer and agreement information according to a predefined format agreed upon by both communication parties. Thus, EDI could potentially be used to efficiently describe multi-attributive, combinatorial requests and offers with preference and pricing functions. However, these pairwise agreements were rarely based on any standards and turned out to be effort-intensive, highly domain-dependent and inflexible. Thus, EDI is not addressing the interoperability and Web-compliance requirements.

Approach	(R1) Web-compliance	(R2) Multi-attribute	(R3) Combinatorial	(R4) Context	(R5) Efficiency	(R6) Interoperability
EDI/EDIFACT	-	✓	✓	✓	(✓)	-
XML-based Languages	✓	✓	✓	✓	-	(✓)
Semantic Web Service	✓	-	(✓)	-	-	✓
Semantic Policy Specification	✓	✓	(✓)	✓	(✓)	(✓)
Bidding Languages for CA	-	✓	✓	(✓)	✓	-
Product/Service Catalogs	✓	-	-	-	-	✓
Our Approach	✓	✓	✓	✓	✓	✓

Table 10.1: Analyzing related work with respect to language-specific requirements.

10.1.2 XML-based Policy Languages

More recent approaches, such as WS-Policy [W3C06b], EPAL [IBM03] and the XACML-based Web Service Policy Language (WSPL) [MAPG03], use XML (eXtensible Markup Language) [W3C04a] as a domain-independent syntax, to define constraints on attributes of configurable trading objects within the context of Web service agreements. However, they are not suitable for our purposes, because they only allow the expression of attribute value pairs and thus cannot be used to express seller pricing and buyer scoring functions. Therefore, multi-attribute bids cannot be expressed in an efficient way. In addition, the meaning of XML annotations is defined in a natural language specification, which is not amenable to machine interpretation and supports ambiguous interpretation. Therefore, such approaches require extensive standardization efforts, which is also a major problem for ebXML [OAS06c].

WS-Agreement [Glo06] and the Web Service Offering Language (WSOL) [Tos05] are further XML-based specification that can be used to express different valuations for configurations. However, since they support only discrete attributes and no functional relations, an exponential number of price attachments are required to model configurable services. Hence, they do not provide the required efficiency. An approach to extend WS-Agreement for expressing continuous functions is presented in [SY05]. However, the XML annotations still lack formal semantics and therefore do not provide the required interoperability. This drawback can be observed also for other approaches. Other rather informal policy languages are discussed in [AAFP03] and specifically for Web service policies in [KL04].

10.1.3 Semantic Web Services

As discussed in Chapter 2 and 6, a way to enable machine interpretation of bids and contracts is to specify them using an ontology. Such an ontology consists of a set of vocabulary terms, with a well-defined semantics provided by logical axioms constraining the interpretation and ensuring the well-formed use of the vocabulary terms. As already mentioned in Section 3.2.1, the Semantic Web Services community has already invested a lot of effort in developing ontologies for describing the functionality and other characteristics of Web services in order to support automated discovery, composition, mediation, etc. Since such vocabularies are also required in our approach for representing values of attributes such as *Input*, *Output* or *Precondition* and *Effect*, they can be considered as orthogonal work. In fact, our framework considers these ontologies as core ontologies for describing Web services. Hence, they can be "plugged-in" as modules in the second layer of the framework (compare 6.1). The most prominent approaches are OWL-S [DAM02], WSMO [DKL+05] and SWASDL [W3C07a] (former called WSDL-S [POSV04]). Since these approaches are not directly related, we refrain from an exhaustive discussion and highlight only those parts that are most interesting for our work. Bits and pieces are taken from the projects' Web pages and publications [SPAS03, POSV04, DKL+05, dBLPF05] as well as from additional literature [LLP+07, ABK+04, Het06, MOGS04].

Web Ontology Language for Services (OWL-S)

The Web Ontology Language for Services (OWL-S) [DAM02] defines an upper ontology for services, which itself is an ontology specified in OWL. Thus, the OWL language together with the OWL-S vocabulary defines the OWL-S Web service specification language [LLP+07]. The OWL-S vocabulary consists of four major elements: a *Service* as a central point of reference referring to a *ServiceProfile*, a *ServiceModel* and a *Service Grounding*. These elements are discussed below.

Service Profile The OWL-S profile describes the intended purpose of services in terms of their functionality as well as non-functional aspects. The functionality is described in OWL-S by the *information transformation* (captured by *Input* and *Output*) and the *state change* provoked by the service execution. The latter is described using *Preconditions* and *Effects*. Non-functional properties are captured by human-readable metadata, such as *serviceName*, *textDescription* or by *Parameters*. *Parameters* are not bounded in their meaning and can therefore be used as a main extension anchor within the OWL-S framework. Moreover, in order to support the discovery and matching of services the *ServiceProfile* can be arranged according to a hierarchy.

Service Model The service model describes how a service works by means of a *ProcessModel*. Generally speaking, the *ProcessModel* provides additional information for service requesters about how to interoperate with the service. It represents the dataflow within a composite service by defining the corresponding *Inputs* and *Outputs*. A composite process is constructed from atomic process by means of control constructs, such as *split* or *sequence*. These constructs roughly correspond to the primitives provided by the Ontology of Plans in our framework.

Service Grounding The service grounding bridges the gap between the semantic description and the concrete interface of a service provided by WSDL. It shows how to technically access a service regarding transportation protocol, message format, addressing and object serialization. Essentially, OWL-S atomic processes are mapped to WSDL operations whereas the input and output parameters in OWL-S are equivalent to WSDL message part definitions.

Although by providing the concept of *Parameters* as means for modeling non-functional properties, OWL-S does not provide native support for defining constraints over *Parameters*. When considering OWL-S as a module for our ontology framework, the concepts *ServiceProfile*, *Input*, *Output*, *Precondition*, *Effect*, *Parameter* and *ProcessModel* can be seen as subclass of *CPO:Attribute*. This enables the definition of goal as well as utility function policies (and thus fine-grained preferences) for these aspects and allows us to use them within the Web service selection process. Advantages of using our approach in the selection process compared to the current native OWL-S matchmakers is discussed in Section 10.2.1.

Web Service Modeling Ontology (WSMO)

The objective of the Web Service Modeling Ontology (WSMO) is to provide means for fully describing a Web service in order to automate all tasks dealing with the intra- and inter-enterprise integration of Web services [LLP+07]. The four main elements that constitute WSMO are identified below. WSMO defines them by means of the Meta Object Facility (MOF) [DKL+05].

Ontologies *Ontologies* provide the terminology used by *Services*, *Goals* and *Mediators*. In a first step, WSMO defines ontologies by means of a general epistemological model capturing the different existing ontology languages, such as RDF [W3C04b], OWL [W3C04c], F-Logic [KLW95], etc. Concrete logical languages for the general model are defined by the Web Service Modeling Language (WSML) [dBLPF05, dBLK+05]. WSML consists of different variants differing in their logical expressiveness and the underlying logical paradigms ranging from description logics to logic programming (and combinations of both).

Services A *Service* describes the *Interfaces* and *Capabilities* using *Ontologies*. *Interfaces* capture the *Choreography*, i.e. how to communicate with the service to make use of the functionality, and the *Orchestration*, i.e. how the service makes use of other services. The *Capabilities* are described by *Preconditions*, *Assumptions*, *Postconditions* end *Effects*. As for all other WSMO elements, non-functional properties can be attached to *Services* in order to capture, e.g., quality of service properties.

Goals *Goals* capture the requesters' view and define the requirements in terms of *Capabilities* and *Interfaces* a suitable service has to meet.

Mediators Finally, *Mediators* handle the heterogeneity of the environment by providing data, protocol and process mediation. The presence of *Mediators* can be seen as a major conceptual advantage of WSMO compared to OWL-S.

Considering the versions of WSML that are compatible with OWL-DL (i.e. WSML-Core, WSML-DL), we can also make use of WSMO as module to define Web service bids and contracts. Although non-functional attributes, which essentially refer to Dublin-Core metadata [WKLW98] and a recently added quality of service ontology [TF06], can be used for representing non-functional properties in WSMO, the current support for modeling, attaching and reasoning with quality of service descriptions is rather limited [TFJ+06]. One reason for this limitation is the fact that WSML does only allow the attachment of one single value to each property, which is obviously not sufficient to support the efficient representation of configurable offers, i.e. neither goal nor utility function policies can be represented.

Semantic Annotations for WSDL (SAWSDL/WSDL-S)

In contrast to the previous approaches, the Semantic Annotations for WSDL (SAWSDL) specification [W3C07a] (aka WSDL-S) does not aim towards a fully-fledged framework for describing Web services, but rather adds XML annotation tags to the WSDL XML-schema in order to enable references to a formal model. In fact, this models can be OWL-S and WSMO [Kop05], but also more informal models such as UML/OCL, for example. In addition, SWASDL provides a schema mapping extension which can be used within XML schema definitions to point to an external mapping specification language document, like XSLT. Hence, SAWSDL handles structural differences between the XML Schema elements and their related semantic model concepts.

However, as the previous approaches SAWSDL does not provide the necessary means for expressing constraints or preferences over multi-attribute service descriptions. In addition, SAWSDL alone does not solve the interoperability problem since it requires an additional semantic model to describe the services.

Others

Besides the approaches presented above, several other semantic Web service approaches have been proposed in recent years. For example, the Semantic Web Service Framework (SWSF) [BBB+05] can be seen as an extension and refinement of OWL-S, which uses a richer logical formalism (full first-order logic) for defining the ontology and the Process Specification Language (PSL) for formally specifying the behavioral aspects. Other approaches that capture behavioral aspects of a service are presented in [AS06, Aga07, BCG+05].

A problem with the above approaches is the fact that the ontologies only describe the services themselves and do not capture the environment in which they are used. Thus, they cannot be used to express context-dependent requests. Additional work that considers the role of context within service-oriented systems focuses on context representation [CCMP06, SSSC06] or the usage of context information for service discovery [ETW04, KK04, DLV06]. Martin [Mar06] specifically discusses the requirements and possible solutions for context modeling with a focus on OWL-S.

Since the Web service ontologies presented above are not capable of expressing constraints on attributes, they are also not sufficient for representing multi-attribute requests and offers in an efficient way. This problem is addressed by approaches that augment the Web service ontologies with ontology-based policy languages. For

example, [KFJ04] provides an approach for extending OWL-S with the policy language REI in order to model goal policies. Similar, approaches exist also for WSMO (see the discussions in [ABK+04, OLPL04]). We thus focus on semantic policy specifications in the following.

10.1.4 Semantic Policy Specifications

As we discussed in the last section, the state of the art regarding semantic Web services currently does not directly support constraints on multiple attributes. Therefore, the ontology-based policy languages such as KAoS [UBJ+04], REI [Kag04] and the work presented by Kolovsiki et al. [KPKH05] can be used to extend the service ontologies in order to allow the definition of multi-attribute policies for representing constraints on attributes. While KAoS and the approach by Kolovski et al. are based mainly on OWL-DL, REI uses OWL-Lite only as syntax for exchanging policies and performs reasoning based on a logic programming approach. How KAoS policies are used for managing semantic Web services is outlined in [UBJ+04]. Since pure OWL-DL is not fully sufficient to cope with the situation where one value depends on other parts of the ontology, they extend the logic by so called *role-value maps* (see also [BCM+03]). In REI (as well as in our work) this issue is addressed by using a rule language. For a detailed comparison of KAoS and REI the interested reader is referred to [TBJ+03]. The most recent approach by Kolovsiki et al. [KPKH05] defines the semantics of the WS-Policy specification by means of an OWL-DL ontology. This allows them to use a standard OWL reasoner for policy management and enforcement. However, due to the open world assumption of OWL their results sometimes are counterintuitive. Therefore, they plan to extend their approach with default logic [BH95] and ensure decidability with restricting default rules to named entities [KP06]. This roughly corresponds to the usage of DL-safe rules in our approach.

The major disadvantage of all these approaches in our context is their limitation in terms of expressing fine-grained preferences. Using these approaches policy evaluation always leads either to true or false, which is insufficient for efficiently encoding scoring or pricing functions.

More expressivity in this context is provided by rule languages, such as SweetDeal [GP03], DR-NEGOTIATE [SABG05, Gov05] and RBSLA [Pas06]. All three are rule-based approaches that use defeasible reasoning (i.e. Courteous Logic Programs or defeasible logic) to specify contracts or agent strategies, respectively. Similar to our approach they feature automatic reasoning based on a formal logic. However, there are some issues regarding the use of a (pure) logic programming paradigm. Often such languages do not provide full-fledged declarative semantics and thus combining rules from different sources becomes highly problematic. In fact, manual integration of the different logic programs might be required. Since in our setting, requests, offers and contracts have to be integrated from different sources, this is a major drawback. From an expressivity point of view, pure logic programming does not support equality reasoning required for expressing integrity constraints as well as number restrictions and lack existential quantification. Another problem is the interoperability of rules within the market. Although RuleML is available as a standard syntax for exchanging rules, the semantics of the syntax is not yet standardized which hinders interoperability. In addition, while the underlying rule language might be capable of expressing utility-based policies, they do not provide the

required policy specific modeling primitives directly, rather the rules for interpreting such policies have to be added manually by the user. In the DR-NEGOTIATE approach qualitative preferences are expressed via defeasible rules and priorities among them. While such an approach is suitable for ranking of alternatives, it is not possible to assess the absolute suitability of an alternative, which is important in case the best alternative is still not good enough.

Most similar to our approach is the work presented in [OVSH06], where WS-Agreement is extended with an ontological model and preferences are expressed via a rule-language. However, like the previous approaches they use a non-standard rule language and do not elaborate on the structure of the preference rules, which may narrow the applicability of the language for many service allocation algorithms (see discussion in Section 10.2.1 below). Concerning context-dependent offers and requests all rule-based approaches (and also description logics with role-value maps) are sufficiently expressive.

10.1.5 Market Bidding Languages

A separate stream of work has focussed on developing highly expressive bidding languages for describing various kinds of attribute dependencies and valuations, particularly in the context of (combinatorial) auctions (e.g. [Nis00], [BH01], [SNVW06]). The strength of these languages is to encode multi-attribute, combinatorial bids in a highly efficient way. In this context, two major approaches to bidding languages can be distinguished [BH01]: (*i*) the family of $\mathcal{L}_B$ languages where logical combinations of bids are associated to prices and (*ii*) the family of $\mathcal{L}_G$ languages where prices are attached directly to logical combinations of goods. The former approach is used in [Nis00, San02, SNVW06] and has the advantage that a user can specify her preferences completely in one bid, whereas the latter may require more than one bit. However, $\mathcal{L}_G$ could be more efficient in the presence of many disjunctions (e.g. $(b_1 \vee b_2) \wedge (b_3 \vee b_4) \wedge \ldots$). The language variant $\mathcal{L}_B^{or^*}$ efficiently captures all possible preferences [Nis00]. As discussed in Section 5.2.2, our approach allows arbitrary XOR/OR-formulae and thus also supports the $\mathcal{L}_B^{or^*}$ language.

However, the approaches discussed above assume a closed environment and therefore, even if they do use XML-based bidding languages [BK05], they do not deal with interoperability issues in the Web. While they do not explicitly consider context dependency, it is usually possible to model context-dependent offers and requests as preferentially dependent attributes.

10.1.6 Product and Service Catalogs

To address interoperability in B2B settings often standardized product and service taxonomies are used, such as the UN/SPSC,[1] the CPV,[2] or the MIT Process Handbook [MCP+99]. However, while these taxonomies are suitable for pure functional descriptions, their static hierarchical structure makes them inappropriate for multi-attribute descriptions. Using hierarchies each new service configuration would require a new node in the hierarchy. Since the number of configurations grows ex-

[1] United Nations Standard Products and Services Code (`http://www.unspsc.org`)

[2] Common Procurement Vocabulary (`http://simap.eu.int/nomen/nomenclature\_standards\_en.html`)

ponentially with the number of attributes (compare discussion in Section 5.1 and Section 9.3.2), such an approach leads to an exponential hierarchy size. Therefore, hierarchies might be applicable to describe individual attributes, but not the entire multi-attribute trading object.

10.1.7 Discussion

Recapitulating, two major streams of work can be identified. First, approaches coming from the area of market engineering propose highly expressive bidding languages for complex electronic markets such as combinatorial auctions. However, they lack support for interoperability in a Web environment. Second, there are approaches addressing interoperability in heterogeneous and dynamic environments such as the Web. These approaches typically do not focus on market-specific requirements. Our work unifies these two aspects in one coherent framework. We draw from utility theory to express scoring and pricing functions of market participants and thereby enable compact representation of context-dependent requests and offers. Furthermore, we show how these functions can be expressed declaratively with a standardized and Web-compliant ontology formalism.

10.2 Contracting and Contract Monitoring Mechanisms

In this section, we discuss contracting and contract monitoring mechanisms with respect to the mechanism-specific requirements (*R8*) Flexibility, (*R9*) Optimality, (*R10*) Tractability, (*R11*) Legal Reliability. In this context, we review work from several rather disjoint communities. First, we address approaches coming from the Web service and Semantic Web community dealing mainly with matching and ranking of Web services. Second, we examine approaches coming from the area of product configuration. They mainly discuss matching of request and offers with respect to differentiated products. As a third approach, we review Web service selection strategies that involve reputation and trust issues. Finally, some approaches from the area of legal information systems are discussed with a focus on contract management. Since all mechanisms discussed below also require a description language, some of the technologies introduced in the last section are revisited.

10.2.1 (Semantic) Web Service Selection

A first approach to make existing Web services discoverable and thus reusable was the Universal Description, Discovery and Integration (UDDI) repository. It can be used by providers to register their services with a name, an WSDL and additional service descriptions and by requesters to browse and find registered services. However, the used description language lacks expressiveness and formality (e.g. some parts rely on natural language descriptions) and thus the UDDI approach turned out to be not suitable for automated service contracting.

As discussed above, the goal of semantic service annotations is to provide description with a high degree of expressive and formality. Several proposals for semantically enabled matching in electronic markets have been presented

Approach	(*R7*) Automation	(*R8*) Flexibility	(*R9*) Optimality	(*R10*) Tractability	(*R11*) Legal Reliability
(Semantic) WS Selection	✓	-	(✓)	✓	-
Product Configuration	✓	-	(✓)	✓	-
Social Service Selection	-	-	(✓)	✓	-
Market-based WS Allocation	✓	(✓)	✓	-	-
WS Contract Management	(✓)	-	-	✓	✓
Our Approach	(✓)	✓	✓	✓	✓

Table 10.2: Analyzing related work with respect to mechanism-specific requirements.

[NSDM03, LH03, TBP03, GMP04]. The approaches improve semantic interoperability by utilizing logical descriptions contained in Web service requests and offers. Depending on the logic used and on the domain, different notions of match can be defined. For example, [PKPS02] presents four different levels of match for OWL descriptions, namely *exact*, *plugin*, *subsumes* and *fail*. The Web services are ranked depending on the type of match that can be realized, i.e. services with an exact match are ranked first, services with a fail are ranked last. Other notions of match for Web services are presented in [GMP04, RDNDS$^+$07]. However, all current approaches use the same notion of match for different ontologies and domains. This approach requires to model ontologies in a way that they fit to the matching approach used. Since in most scenarios, already existing domain ontologies should be used, this rather inflexible approach is problematic. As outlined in Section 7.1.1, in our work we address this problem by means of customizable matching predicates for each attribute that can be seamlessly changed, added or removed. This enables us to add domain ontologies (with the corresponding matching rules) during runtime of the system. In addition, approaches relying purely on logical matching do typically lead only to a coarse preference structure over the different alternatives and thus to insufficient rankings. Therefore, one can argue that pure logical matching is not sufficient and has to be augmented by "value reasoning" [SRT05, KFS06].

Matchmaking approaches that are based on information retrieval techniques utilizing ontological distance measures as done in [KFKS05, SMSA$^+$05, BK06a] provide such a more quantitative view. Here rankings are calculated by measuring similarity between concepts or property values within an ontology. Klusch and colleagues [KFKS05] extend the approach also with the logic-based matchmaking algorithm introduced by [PKPS02]. This is similar to our approach, since similarity measures can be seen as special type of preference functions that can be implemented by defining the appropriate matching rules. However, our selection approach is not limited to

similarity-based preferences, but instead preferences and pricing functions can be explicitly stated, which is not possible using the former approaches.

Specifying the valuation of certain alternatives explicitly is possible in the system presented by Balke and Wagner [BW03]. They use SQL-based preference queries introduced in [Kie02] for their Web service selection algorithm. While this work does not utilize semantic service annotations, in a recent extension it has been shown how such preference queries can be formulated for SPARQL [SPT06]. This approach is similar to our preference queries defined in Section 7.1.2. However, it meets our requirements only partially since no policies can be expressed. Policies are required to state information about preferences (e.g. in which context they should be used), to specify multi-attribute offers (e.g. by means of pricing policies), and for more complex allocation mechanisms such as a double auction.

In order to represent such information declaratively some kind of rule language is required (such as in our case DL-safe SWRL rules). As discussed in Section 10.1.4, prominent examples are SweetDeal [GP03] and the work by Oldham and colleagues [OVSH06]. Although these approaches show how preferences can be formally represented, they currently lack a formal selection model (as we present in Section 5.3), which allows to judge the computational tractability of the proposed algorithms. Optimal and efficient algorithms for the Local Selection Problem (LSP) and Global Selection Problem (GSP) are presented in [LNZ04, BF05] and [ZBN+04, VAG+04, CSM+04, YL05, AP05], respectively. For the GSP also AI planning algorithms exists that rely on (qualitative) preferential information [BFM06]. However, both communities focus purely on algorithm aspects and abstracts from the representation issues. Like these approaches, our work utilizes efficient optimization techniques for Web service selection (limited to LSP), but also augments them with the required service description and matching models.

Just recently several WSMO based approaches for Web service selection by means of user preferences have been presented [VHPA06, WVKT06]. There are several major differences to our work. First, the semantic matching and the selection algorithm are separated. Semantic matching is only executed with respect to functional properties in order to determine valid services. The ranking for these services is then determined based only on "flat" quality of service criteria. Second, the degree of flexibility is lower since metrics are hard-coded and statically assigned to quality of service attributes at development time. In addition, they do not focus on highly configurable services, but consider only the problem of selecting between different providers. The latter is addressed in product configuration which is discussed below.

From a preference modeling point of view, most similar to our work is the approach presented by [KKR04, KKRKS07]. They use fuzzy sets for expressing soft constraints over attribute values which corresponds directly to the utility function policies used in this work. The major drawback of their approach is the lack of expressivity for modeling attribute values. While we use standard description logic for expressing attribute values, they have invented a new, proprietary logic which provides very limited expressivity restricted to type hierarchies.

10.2.2 Product Configuration

Product configuration describes the problem a user faces when choosing values for the parameters of a configurable product. Examples for typical configurable products are computer systems, customizable cars, made-to-order factory parts, or – as discussed in Section 2.1.2 – Web services. A model capturing configuration problems is composed of the description of all the attributes characterizing it, the allowed values for these attributes and the constraints expressing incompatible values [MF89]. A wide range of different techniques can be used to solve different forms of configuration problems. A detailed survey of models, techniques and tools is presented in [Stu97, SW98]. Since the basic problem formulation corresponds to a constraint satisfaction problem, this is one of the most prominent methods. Finding a service that adheres to the goal policies introduced in Section 5.1 can be seen as a combinatorial problem that corresponds to the definition of a *Constraint Satisfaction Problem* [Tsa93]. The corresponding problem formulation is given by the following tuple (L, A, Φ). Similar approaches for Web service selection have been discussed in [MDRCD+03] for Enterprise services and in [LYFA02, LF04] for Grid services.

For applying constraint satisfaction approaches to configuration problems the need for logic-based product descriptions has been widely acknowledge by the community [SW98]. Several approaches that utilize description logic for matching of products have been presented (e.g. [MIP+98, FFJ+03, JM03]) to improve interoperability between heterogeneous descriptions. The problem of the constraint satisfaction mechanisms so far is that they only support, so called, hard constraints which correspond to goal policies in our framework. Since this is clearly insufficient in many cases, there is work about adding fine-grained preferences to constraint problems. The most prominent approach are semiring-based constraint satisfaction problems [BMR97]. They add an objective function to the problem which is used to rank the admissible results. This objective function can be compared to our scoring policies. Pricing policies are not explicitly considered and the available notions of match are typically restricted to a rather small set of operators, such as '$<$','$>$' or '$=$'. In addition, to the best of our knowledge there is no approach that combines a semiring-based constraint satisfaction formulation with an expressive description logic such as $\mathcal{SHOIN}$.

10.2.3 Social Service Selection

Social service selection [SH05, Chp. 20] can be seen as a special case where the decisive factors in the service selection process are social attributes that rely on notions such as trust and reputation. Technically, this means the same algorithms can be applied as presented in previous sections. The problem here is the determination of the values for trust and reputation. In this context, algorithms are required that aggregate the experiences of users in a distributed and fair way. In addition, mechanisms have to be found that give incentives to submit ratings for service executions. Reputation and trust based service selection approaches are presented in [SS04a, MS04, JF05, LHVA05].

Since reputation and trust determination involves judging the quality of an Web service execution, it mostly involves human interaction which contradicts Requirement (*R7*). In fact, the contract monitoring approach presented in Section 7.2 can be

used to assess (at least partially) the quality of a Web service execution and to come up with provider ratings in an automated fashion.

Another complementary approach to social service selection are recommender techniques, such as collaborative filtering, that log customer data in order to utilize this data for future decisions. For example, the market operator could store which service is selected by which of the customers and then use this data to select services for incoming requests from similar customers.

In this work, service selection is done based only on the preference data of a requester without considering "community data". However, additional social attributes, such as reputation or user ratings, can be easily added to the decision problem formulated in Definition 5.10.

10.2.4 Market-based Web Service Allocation

All selection approaches discussed up to now have involved some kind of optimization that ensures that the allocation meets the desired goal of the market designer in an optimal way. For example, the Web service selection algorithm LSP presented in Section 7.1.1 ensures that the services is selected that maximizes the requester's utility. Hereby, we assume that the defined scoring policies represent the requesters real utility function. Considering the economic properties of this Hit-and-Take-mechanism some problems become evident: the mechanism is not incentive compatible in a sense that providers have an incentive to reveal their true valuation of providing a service to the requester. That means it can be advantageous for a provider to strategically over- or underprice the service.

To address this problem market mechanisms can be introduced where prices are dynamically determined based on supply and demand. As discussed in Section 2.3.2, such dynamic pricing mechanism can lead to economically efficient allocations [Hur73]. Thereby, they make sure that a provided services is awarded to the requester who has the highest valuation of the service. This is particularly relevant for grid services since not all requesters might get the best service due to resource limitations. An overview of market mechanisms for grid services can be found in [BAGS02, BAV05]. Hereby, an important goal is the realization of an incentive compatible mechanism, where the optimal strategy for all participants is to reveal their true valuations (cf. Section 2.3). A detailed discussion of market mechanisms with respect to their economic properties can be found in mechanism design literature [Hur73, Par01, Jac02]. Since these dynamic pricing mechanisms have to be tailored towards a concrete scenario, we refrain from introducing concrete mechanisms but rather exemplify how the Local Allocation Problem can be extended to dynamic pricing mechanisms (see Section 7.1.2). In this context, a RFQ-auction is introduced that realizes a simple auction protocol based on the selection infrastructure. As an example for more complex auction mechanisms, the approach is extended to a combinatorial multi-attribute double auction by means of the MACE system [SNVW06, LS06]. However, based on the infrastructure presented in this work also other market mechanisms can be implemented. This makes them applicable in an open and heterogeneous environment such as the Web.

10.2.5 Web Service Contract Management

Automated management of contracts requires a formal, machine-interpretable descriptions [MG05]. Thereby, automation of management tasks like contract formation, monitoring and execution is enabled. Throughout Section 10.1, we have discussed several languages that strive for formalization of Web service contracts. For example, these include ontologies that define a formal semantics for WS-Agreement constructs [OVSH06, JW05] or languages based on proprietary rule formalisms [GP03, Gov05]. They all enable closing machine-interpretable contracts in an automated fashion. However, they are not capable of fully expressing a real-world contract and thus fail in providing legally reliable contracts. Our approach is different in that we do not strive for full automation, but augment an automatically closed contract with an umbrella contract that provides the legal basis for the automation. An alternative would be to transform formal agreements to natural language contracts by means of a template-based approach as presented in [HF05]. The problem here is that the formality of the contract is lost and thus contract monitoring and execution process cannot be automated. Another major difference is that the approaches mentioned above focus mainly on the contracting phase. While providing a machine-interpretable language, they do not provide a formal language for representing Web service monitoring information and no specific customizable contract monitoring algorithms. An approach for automated verification, validation and consistency checks of RBSLA contracts is presented in [PBD05]. As discussed in the language part, while being considerably more expressive than our contract representation, RBSLA is based on a combination of proprietary rule languages and requires several different reasoners to perform this task. An approach for checking the compliance of a business process with respect to a contract is presented in [GMS06].

However, contract management is only a part of integrated quality of service management. Work that complements our approach deals with monitoring of Web service executions, which involves measuring and predicting certain service levels. That means besides contract management, tasks such as admission control, resource management, resource monitoring, system diagnostics and system adaption are required. An integrated approach to quality of service management is presented in [CSDS03, LDK04, WWC+05]. For ensuring certain quality levels certain Web service standards are available. For example, WS-Reliability [OAS04] can be used to provide protocol guarantees, such as the guarantee that the order of messages is preserved. First commercial tools such as the Oracle® Web Service Manager[3] support end-to-end monitoring of business processes. This is also utilized by our approach to capture the required monitoring information.

10.2.6 Discussion

A wide range of different Web service contracting algorithms have been presented in recent years. Most of them provide means for optimizing the selection with respect to a certain goal. However, some approaches lack support for user preferences and purely rely on (semantic or syntactic) distance measures such as different notions of match or concept similarity. For most approaches the selection can be done auto-

[3] Available at `http://www.oracle.com/technology/products/webservices_manager`

matically in a computationally tractable manner. For social selection approaches, in contrast, feedback about service invocations is required, which in most cases has to be determined manually. All Web service selection approaches mentioned above do not address the flexibility requirement adequately, since they use fixed algorithms for matching as well as allocation. In addition, legal reliability is not in their focus and their optimization goals do not address economic objectives such as incentive compatibility. This is provided by market-based allocation mechanisms, which in turn require considerable additional time for determining the allocation (especially if several bidding rounds are involved). Finally, there are several approaches that strive for automated conclusion of contracts. However, legal reliability cannot be reached in a fully automated fashion. This is also the case in our work. Therefore, we use a semi-automated approach where an automatically closed contract is augmented with an umbrella contract. The flexibility is provided by customizable matching and allocation rules that seamlessly combine efficient optimization techniques with semantics-based matchmaking. Thereby, we enable efficient calculation of optimal allocations.

Chapter 11

Conclusions and Outlook

In this thesis, we outlined the entire development process of an ontology-based Web service market infrastructure from the requirements analysis to the evaluation stage. By combining techniques from the area of computer science, economics and law in a novel way, this work proposed a market infrastructure for enabling automated contracting and contract monitoring. Thereby, we contributed to the state of the art in designing electronic markets by showing how semantic technologies can be utilized for solving problems such as interoperability or insufficient flexibility. In order to develop such a system in a structured way, we provided a coherent methodology that integrates the engineering of service-oriented architectures, markets, and ontologies.

In Section 11.1, we briefly summarize the contents of this work and accentuate the major contributions. Subsequently, open research questions are reviewed and an outlook on how they could be addressed in future work is given in Section 11.2.

11.1 Summary of Contributions

Service-oriented architectures require coordination mechanisms bringing together service requesters and providers. Throughout the work such a coordination mechanism is referred to as Web service market. The trend towards increasingly adaptive as well as autonomous service-oriented architectures requires an infrastructure that enables a high degree of automation within Web services markets. This affects mainly the contracting process that includes matching of service offers and requests, determining suitable allocations and closing contracts.

In order to support the automation of the contracting process, this work investigated two main research questions: (*i*) Can semantic technologies be used to express policies such that they enable the specification of offers, requests and contracts in Web service markets? (*ii*) Can we automate the contracting process in the market based on these semantic descriptions? The two research questions were addressed by developing an ontology framework and a mechanism that enables the automation of the contracting and contract monitoring process. In this context, we pointed out the need for semantic technologies and showed the benefits that can be realized in Web service markets.

In general, we focused mainly on two aspects: the *language* used within the market and the *mechanisms* that provide the market functionality. In Part II of this work, we discussed the design of these two aspects. In Chapter 4, we introduced three

general scenarios and analyzed them with respect to the requirements they impose on the market infrastructure. Thereby, six language-specific and five mechanism-specific requirements were derived. In Chapter 5, we presented an implementation-independent conceptual design with a formal policy model that facilitates the expression of bids and contracts in the market. Based on these communication primitives, we outlined a contracting mechanism that comprises the matching, allocation, acceptance and (partly) the settlement phase.

In the embodiment phase, this conceptual design is then implemented by means of a specific ontology language (Chapter 6). In order to feature interoperability and Web-compliance, our technology of choice are ontologies. They provide expressive, declarative and formal means for representing policies and the communication primitives based on them. Regarding the ontology-specific aspects, the following main contribution can be identified:

- We introduced an expressive ontology framework for representing the major communication primitives required in electronic markets. The framework is based on a clean and highly axiomatized foundational ontology and makes extensive use of typical ontology design patterns. This allows us to ensure high quality and to avoid typical shortcomings that can be identified in naively built ontologies, such as conceptual ambiguity or poor axiomatization [Obe05]. Moreover, we showed how the framework can be expressed using a standard ontology language that is suitable for the Web.

- Major contributions of this work are the core ontologies that are newly added to the framework. As a central part, the Core Policy Ontology takes up the idea of utility function policies presented in [KW04] and shows how they can be formally represented in a declarative way. To the best of our knowledge, this is the first work heading towards this goal. Since utility function policies are modeled as direct extensions of traditional goal policies, goal policies can also be expressed using the ontology. The declarative nature of the policies enables their integration and thus facilitates management tasks, such as checking for consistency and automatically enforcing policies.

- Based on the Core Policy Ontology, primitives for expressing offers and requests are provided by the Core Ontology of Bids. The Core Ontology of Bids supports customizable services by providing a compact way for describing configurable service offers. Compared to the state of the art where configurations are usually described by enumerations, we were able to considerably increase communication efficiency (Section 9.3.2) as well as the computational tractability of the contracting process (Section 9.3.3) by means of utility function policies.

- The Core Contract Ontology extends the Core Policy Ontology by adding primitives for formally expressing Web service contracts. The advantages of the underlying ontologies also carry over to the Core Contract Ontology. In particular, the formal representation of contracts is a necessary precondition for realizing automated contract execution and monitoring.

By leveraging the formal nature of the ontological descriptions, contracting in the market can be widely automatized. Chapter 7 provides novel algorithms for

automated matching, allocation, contract formation and contract monitoring. These algorithms rely on semantic technologies to realize a powerful, but still efficient market infrastructure. In the following, we briefly summarize the mechanism-specific contributions:

- The work illustrated how existing semantic Web service annotation frameworks can be aligned with a well-known multi-attribute decision making approach based on techniques from economic fields, such as decision theory and operations research. While this allows us to use standard decision theoretic approaches, it also enables leveraging matching algorithms based on description logic, which provide improved interoperability and thus increase the quality of the results (Section 9.3.4). By providing a seamless integration of efficient optimization techniques into this framework, we were also able to ensure the required performance of the contracting process (Section 9.3.3).

- Another important problem inherent to existing approaches is the hard-coding of matching and allocation algorithms. Since different aspects of services often have to be matched in a different way, such hard-coded solutions do not allow adding of attributes during runtime. We approached this problem by means of customizable matching predicates that can be added as well as changed at the runtime of the systems. Similarly, we allow the adaption of allocation mechanisms in a flexible manner. Thus, our infrastructure can be easily adapted to new services and application scenarios.

- We provided the desired automation by relying on the concept of formal policies, where guidelines are used to specify declaratively how the system should behave. Based on these guidelines, the system can allocate services optimally without human interventions. In contrast to related approaches, this work addressed the context-dependency of decisions. We handled this issue by associating policies to certain environmental conditions, such as time or location.

- After an agreement between market participants is reached, a legally enforceable contract has to be closed. We support this step by providing a semi-automated contracting process. A formal contract expressed via the Core Contract Ontology is augmented by a manually closed umbrella agreement. Without such an umbrella contract legal reliability cannot be guaranteed. Since automated contracting may lead to a heterogeneous set of individual contracts, we provided sophisticated contract management mechanisms that enable contract execution and monitoring.

In Part III, our approach to address the research questions was implemented and evaluated. In a first step, we introduced the KASWS prototype in Chapter 8, which implements the contracting process based on the presented ontology framework. The prototype utilizes WS-BPEL as a workflow language. In this work, we showed how automated contracting of Web services can be realized using a standard WS-BPEL execution engine. In addition, tools for defining requests and offers, and a server component for hosting the market platform was presented. Based on this implementation, we discussed the market infrastructure with respect to the requirements in Chapter 9. In this context, an evaluation of the communication

efficiency, computational tractability and the optimality of our approach was conducted. Thereby, we showed that a computationally tractable infrastructure can be realized by means of standard linear programming solvers assuming additive pricing and scoring policies. In addition, the results indicate that with increasing customization possibilities there will be also an increasing need for semantic technologies to ensure reasonable results.

11.2 Future Work

There are several directions in which our approach can be extended. First, more sophisticated allocation mechanisms than presented in this work can be envisioned. Such mechanisms might address non-linear pricing or handle service selection for an entire workflow. Extensions to the selection algorithms are discussed in Section 11.2.1. Second, while introducing an automated contracting mechanism allows the determination of Web service bindings for given requests and offers, specifying these requests and offers is cumbersome and often cannot be done manually. Therefore, solutions are required that automatically generate requests and offers based on system information (Section 11.2.2). Third, the Core Contract Ontology is very restricted in the normative positions that can be expressed. In fact, the Core Contract Ontology currently supports only obligations. Although obligations are by far the most important normative position, also other positions such as permissions might be necessary in some settings. In addition, our systems lacks means for reasoning over temporal aspects of a contract. Possible extensions to the contract representation are discussed in Section 11.2.3. Regarding implementation, the assumption of having the same interface descriptions can be relaxed to apply our prototype in a broader context. In the following, this is discussed in Section 11.2.4.

11.2.1 Extensions to the Selection Algorithm

In this work, we presented a basic Web service selection algorithm and showed how this algorithm can be used for implementing a simple negotiation protocol. As already discussed in Section 5.3.2, this simple negotiation protocol can be extended in several ways. For example, it would be interesting to consider the extension of our approach to algorithms dealing with non-linear pricing [BF05], multi-unit negotiation schemes [EWL06], homogeneity constraints [BK05], and the Global Selection Problem [ZBN+04, SBM+04, AVMM04, YL05, JMG05]. However, the question is how these algorithms can be implemented in the presented Web service infrastructure.

In [AL05], an approach is provided that can be used to aggregate service properties along the workflow. This is required to solve the Global Selection Problem. In this context, aggregation rules are specified using DL-safe rules that determine how a certain attribute is aggregated based on workflow constructs such as split, choice, etc. By adapting the global selection algorithms mentioned above to the specific setting, services could be selected based on the aggregated information. However, the specification of aggregation rules for various different attributes and the computational complexity of the optimization algorithms are a major issue in this context.

11.2.2 Automated Bidding

An important assumption underlying our work was that market participants know their bids exactly. This does not only include the specific Web service they require/provide at some future point in time, but also the price and quality. This assumption is very much simplifying reality and obstructs the actual use of Web service markets. To establish a prospering Web service market rules how to conduct the bidding are required. Since the demand and supply situation can be extremely volatile, manual bidding is to slow to accommodate abrupt demand/supply shifts. Therefore, automated approaches have to be developed that leverage current or estimated system information such as predicted workload. An initial approach is presented in [NLS06], where requests and offers are generated automatically based on workload information as well as bidding policies. In this context, an important future topic is also the development of policy management tools that support the entire policy lifecycle.

11.2.3 Expressive Contract Representation

Obligations are often not sufficient to fully formalize Web service contracts. For example, contracts typically also state rights, permissions, duties, etc. A detailed overview of a wide range of different normative positions can be found in [Ser01]. Although these normative position can often be expressed by complex modal logics in a formal way, an interesting further research question would be whether we are able to avoid such complex, often undecidable logics by capturing the semantics of additional normative positions in our framework. This would require to introduce new types of *CPO:PolicyDescriptions* in addition to *CCO:Obligations* and to extend the *DnS:satisfies*-relation for handling these normative positions. However, whether such an extension is possible for normative positions beyond obligations is the subject to further research.

Another interesting further research question is the incorporation of dynamic aspects into the contract descriptions. While our approach allows us to express the fact that certain tasks have to be executed in a certain order, we currently do not support sophisticated reasoning over such dynamic aspects. However, this is required for verification of a correct execution of the temporal regulations contained in a contract. A possible solution for this problem using temporal logic is briefly discussed in the section.

11.2.4 Extending the Prototype

Our prototype can be extended in several ways: First, a major limitation of our current prototype is the fact that all alternative Web services are required to have an identical interface. Unless there are no standardized Web service interfaces as proposed by [GMN05], interface and protocol mappings are required. Solutions for these problems are currently under investigation in many projects [PA05, KRMF06, HBM+07] and an integration of these solutions in our prototype should be considered. Second, our prototype can be extended by allowing the specification of policies on temporal attributes of a Web services. How values of temporal attributes, such as the execution order of certain tasks, can be modeled using

π-calculus is extensively discussed in [Aga07]. By combining a temporal logic, such as μ-calculus [Koz83], with the utility function policies defined in this work, pricing and scoring policies as well as contractual obligations can be specified also for such temporal attributes. Thereby, one can express, e.g., that services are preferred that return route planning information, before credit card information has to be disclosed.

In summery, this work provided a Web service market infrastructure and showed how semantic technologies can be leveraged to automate the contracting process. This is a first step towards realizing the vision of applications being assembled and reconfigured automatically according to the users' needs. Although additional research questions have to be addressed to fully reach this goal, the presented ontology framework and contracting mechanism provide a solid basis for initial deployment as well as further research. In particular, the infrastructure paves the way for implementing more sophisticated market mechanisms in an open and heterogenous environment such as the Web.

Part V

Appendix

Appendix A

Detailed Evaluation Results

On the following pages, detailed evaluation results are listed. Table A.1 shows the number of bytes required for storing a configurable bid with an increasing number of configurations per bid. The table compares the compactness of the policy-based bid representation with an enumeration-based approach. Table A.2 compares the two approaches in terms of performance. This is done for several settings with varying number of offers in the knowledge base and and varying number of configurations per offer. The third table (Table A.3) compares the three different optimal variants for solving the MMP introduced in Section 7.1.1 with a random baseline algorithm. Table A.4 lists the relative increase in the number of matches and Table A.5 the gain in utility that can be realized by introducing semantic matching.

	Enumeration-based approach		**Policy-based approach**	
Number of configurations	Number of offers			
	1	1000	1	1000
1	13584	884249	16713	4730290
4	18551	3180842	18802	6842185
9	26830	7007536	20892	8954157
16	38428	12387458	22980	11066050
25	53336	19302521	25070	13177972
36	71612	27756265	27161	15289955
49	93138	37745240	29249	17401855
64	118002	49272716	31334	19513784
81	146233	62337494	33436	21625528
100	177642	76937326	35512	23737525
121	212641	93161460	37614	25863421
144	251193	110930077	39728	27989455
169	292753	130246219	41827	30115456
196	337890	151110492	43922	32241324
225	386319	173517589	46035	34367403

Table A.1: Comparison of *Bid*-compactness measured in bytes.

Number of configurations	Enumeration-based approach						Policy-based approach					
	Number of offers						Number of offers					
	1	200	400	600	800	1000	1	200	400	600	800	1000
1	1						3	12	15	20	25	32
20	6	194	363	575	780	976	30	301	577	903	1224	1473
100	9	2042	4242	6828	8666	11414	16	1774	3552	5356	7168	9466
225	40	10352	20859	31380	42586	54592	49	5373	11014	17630	23130	29358
400	126	32615	66689	99168	132298	169301	107	12360	24753	38161	52519	65582
625	313	81734	166531	273764	356475	414763	204	24176	47834	214233	105613	132147
900	652	185013	355631	527661	740778	961524	362	45043	87550	130187	181685	245228
1225	1163	352069	726405	1047356	1484515	1913432	605	70275	146705	219234	297666	391032

Table A.2: Performance of enumeration- and policy-based bid representation measured in ms.

Number of offers	Number of requests	Variant [V1]	[V2]	[V3]	random
10	100	88.40	11.00	98.70	2
10	225	275.75	14.50	96.38	2
10	400	578.78	18.89	50.00	6
10	625	1070.75	35.00	115.75	12
10	900	2032.50	49.00	136.75	12
10	1225	2805.75	54.75	153.75	16
10	1600	4388.50	73.00	175.25	125
210	100	1964.90	143.40	1055.70	28
210	225	5477.67	238.00	1024.33	59
210	400	12549.30	389.60	1278.80	130
210	625	23402.50	575.75	1550.75	229
210	900	40933.50	909.25	1943.75	354
210	1225	57545.00	1004.40	2456.00	566
210	1600	91913.50	1435.50	2807.25	667
410	100	3621.30	340.70	2565.10	58
410	225	10926.10	526.20	1882.30	143
410	400	24445.20	827.50	2315.10	285
410	625	46159.60	1219.40	2954.20	492
410	900	83385.00	1817.80	3804.80	732
410	1225	114123.60	2064.60	4364.20	1087
410	1600	195892.60	2910.00	6142.80	1414
610	100	5440.50	596.90	3880.00	97
610	225	16541.60	875.60	2867.40	239
610	400	36551.70	1335.20	3541.20	477
610	625	68247.80	1923.80	4530.40	872
610	900	126494.75	2869.00	5919.75	1231
610	1225	178032.00	3276.00	7176.67	1673
610	1600	303819.00	4830.00	9941.40	2392
810	100	7406.20	939.30	4604.70	137
810	225	22058.90	1283.50	3866.40	341
810	400	49532.40	1957.70	4927.40	659
810	625	92020.40	2727.80	6150.20	1167
810	900	162013.40	3844.20	8108.80	1631
810	1225	261356.60	5373.80	10242.40	2670
810	1600	392423.60	6261.40	13412.40	4244
1010	100	9363.70	1406.90	5889.00	194
1010	225	27704.60	1813.70	4997.20	485
1010	400	63007.90	2653.60	6505.30	916
1010	625	114931.20	3586.40	7887.60	1452
1010	900	209159.00	5008.40	10986.20	2594
1010	1225	334040.40	6650.60	14288.80	3590
1010	1600	499499.75	8301.25	17637.25	5415
1210	100	11378.70	1900.80	7631.70	244
1210	225	33705.20	2418.70	6223.80	609

Table A.3: Performance of matching variants measured in ms.

Number of offers	Number of requests	Variant [V1]	[V2]	[V3]	random
		...table continued...			
1210	400	76722.00	3462.00	8082.80	1114
1210	625	141408.80	4795.80	10310.00	1785
1210	900	250168.80	6570.60	13699.80	2992
1210	1225	385929.40	7657.80	18569.40	4411
1210	1600	615422.40	10829.00	27118.00	7675
1410	100	13158.20	2560.40	10016.30	288
1410	225	39189.00	3273.20	7584.80	697
1410	400	89581.80	4681.40	9707.60	1365
1410	625	170953.20	6253.80	13235.80	2493
1410	900	299713.60	8066.20	17863.00	3817
1410	1225	472858.60	9351.00	24232.60	6442
1410	1600	725616.60	12215.60	30870.00	8547
1610	100	15376.44	3376.33	12525.00	341
1610	225	44423.67	4114.11	9123.00	827
1610	400	105556.80	5741.60	12226.40	1561
1610	625	204100.80	7543.00	15905.60	2701
1610	900	356955.40	9797.20	21617.60	4281
1610	1225	564605.80	12520.40	32485.20	7684
1610	1600	829816.00	15782.00	41827.00	12243
1810	100	16546.00	4054.00	9440.00	381
1810	225	51097.10	5284.80	11168.80	983
1810	400	118398.20	7000.60	14366.00	1773
1810	625	230666.00	9164.00	19113.00	2930
1810	900	397408.80	11853.40	26760.00	5610
1810	1225	633159.60	14432.60	35638.40	8381
1810	1600	1018316.00	18928.00	57277.00	14999
2010	100	18377.00	5106.00	10967.00	474
2010	225	56024.40	6457.40	13028.40	1103
2010	400	129337.00	8346.00	17141.60	2224
2010	625	257530.40	10696.80	22923.60	3620
2010	900	477285.20	15189.20	41808.60	6893
2010	1225	750579.80	17367.40	47258.20	9628
2010	1600	1267707.20	23656.80	69836.40	17158

Table A.3: Performance of matching variants measured in ms.

Number of offers	Ontology Size small	medium	large
50	2.1294615	4.121233	6.3432
100	2.0834138	3.3342234	5.85673
150	2.040428	3.6711953	4.86775
200	1.9481369	2.8867044	4.23123
250	2.028849	2.9601085	5.567565
300	2.018622	3.3694522	6.32232
350	1.9792932	3.7323475	4.97856
400	2.139109	3.370374	5.3223
450	2.024841	3.1682165	5.986
500	1.9944267	2.9396975	5.542
550	1.9858284	3.162173	4.964
600	1.9641815	3.0488298	5.98987
650	1.9487522	2.7331111	5.334362
700	2.0612867	3.058414	6.189736
750	1.9686356	3.1586716	6.5806847
800	2.007206	3.1291995	6.445567
850	2.0255516	2.75824	5.823243
900	2.0224805	2.8404422	5.0331
950	2.0243948	2.949343	5.6380305
1000	2.0013342	3.1555436	5.7260814
1050	2.0125282	2.9738472	4.7477293
1100	2.0098119	3.1146703	5.713008
1150	2.0276752	2.7581966	5.592429
1200	1.9817696	3.1809587	6.2915382
1250	2.037736	2.8324652	5.3207836
1300	2.0000722	2.9998546	5.52149
1350	1.9874947	3.2531524	5.575553
1400	1.9797436	3.0936215	4.710363
1450	2.0347984	2.99901	5.7617445
1500	2.0126047	2.9012818	5.4873
1550	2.0272074	3.0749483	7.14308
1600	2.0129132	3.1874866	5.672242

Table A.4: Relative increase in number of matches.

Number of offers	Ontology Size small	medium	large
20	0.09555222	0.37464982	0.19251056
40	0.05517628	0.3081325	0.30062455
60	0.04938148	0.27226597	0.36664814
80	0.036708742	0.2529401	0.47242188
100	0.022990435	0.14397457	0.43292484
120	0.02256395	0.17738761	0.3604471
140	0.022581125	0.17552733	0.40090007
160	0.01760894	0.119190454	0.35450992
180	0.013973519	0.052918535	0.37577412
200	0.015289108	0.086436376	0.40634733
220	0.012868008	0.08185781	0.4852279
240	0.008909034	0.09583692	0.38248003
260	0.010487919	0.0924208	0.32111472
280	0.010691282	0.092999406	0.47933888
300	0.012235651	0.0719818	0.36299497
320	0.008779077	0.056526937	0.3361407
340	0.007585828	0.039164335	0.34986693
360	0.008028221	0.054182816	0.3363778
380	0.0059215548	0.047897566	0.31925362
400	0.0069562746	0.04975953	0.30695474
420	0.0061652004	0.05378554	0.32924053
440	0.008710342	0.0461867685	0.26146355
460	0.006203355	0.039130975	0.24230418
480	0.0046784864	0.036651656	0.28084144
500	0.0062209563	0.0290486815	0.2632665
520	0.0032276108	0.040161386	0.32026425
540	0.004538642	0.03591316	0.24324867
560	0.003889592	0.0366198	0.23201433
580	0.004039617	0.04195609	0.25004023
600	0.00377196525	0.028168267	0.22053334
620	0.00305063425	0.024819074	0.28980482
640	0.0029021874	0.037336957	0.22122073
660	0.0038956509	0.037800502	0.2365056
680	0.0027790456	0.03350959	0.23051052
700	0.0038645505	0.023034053	0.17912284
720	0.0032849847	0.029631373	0.19297531
740	0.0031958327	0.03426681	0.20062312
760	0.0030074283	0.03157547	0.19072086
780	0.0042936774	0.023953684	0.19240904
800	0.003054501	0.019866526	0.14516315
820	0.0030347258	0.030413609	0.17781146
840	0.0038933815	0.021336889	0.2008796
860	0.0027463883	0.02413283	0.1822948
880	0.0030268072	0.020077545	0.1774458

Table A.5: Absolute increase in utility.

Number of offers	Ontology Size small	medium	large
	...table continued...		
900	0.0026097656	0.02638042	0.20317516
920	0.0030189245	0.021451754	0.14562145
940	0.0026009171	0.0100156695	0.14834931
960	0.00204681 5	0.0322994485	0.13981244
980	0.0020820692	0.015991356	0.23037796
1000	0.0021654442	0.02420365	0.15034899
1020	0.0032640204	0.021291722	0.17357865
1040	0.002760285	0.013192313	0.118703626
1060	0.0020622672	0.01955185	0.12257241
1080	0.002252339	0.030172277	0.15654509
1100	0.0027919323	0.017989727	0.16042116
1120	0.002001287	0.013039539	0.13276814
1140	0.0022746518	0.011950105	0.13801081
1160	0.0016592741	0.016651746	0.1701133
1180	0.0014683396	0.023741398	0.10756383
1200	0.0017239705	0.015593271	0.13537155
1220	0.0032547847	0.013101587	0.080048814
1240	0.0021203817	0.017965656	0.12632816
1260	0.002794765	0.017708108	0.1773103
1280	0.0020168736	0.014791583	0.1738556
1300	0.0016111017	0.017072136	0.14256744
1320	0.0014701232	0.014631683	0.12065126
1340	0.0017422572	0.009344198	0.13710518
1360	0.0017253712	0.011862086	0.10211418
1380	0.0017002166	0.010610642	0.121615686
1400	0.0016830281	0.017689139	0.104814075
1420	0.0018417432	0.011392601	0.09553937
1440	0.0016883373	0.00914073	0.10963619
1460	0.0017317086	0.014893098	0.11682918
1480	0.0015989288	0.013872844	0.1184862
1500	0.0014090643	0.011929865	0.068410076
1520	0.0015754208	0.013377261	0.12113015
1540	0.001199834	0.010287596	0.10936712
1560	0.0018342689	0.011092854	0.09239143
1580	0.001652427	0.014645634	0.08503954
1600	0.0016681179	0.009022864	0.11921606

Table A.5: Absolute increase in utility.

References

[AAFP03] Issam Aib, Nazim Agoulmine, Mauro Sergio Fonseca, and Guy Pujolle. Analysis of Policy Management Models and Specification Languages. In Dominique Gaïti, Guy Pujolle, Ahmed Al-Naamany, Hadj Bourdoucen, and Lazhar Khriji, editors, *Network control and engineering for QoS, security and mobility II*, pages 26–50. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[ABdB+05] Vladimir Alexiev, Michael Breu, Jos de Bruijn, Dieter Fensel, Rubén Lara, and Holger Lausen, editors. *Information Integration with Ontologies: Experiences from an Industrial Showcase*. Wiley, February 2005.

[ABK+04] Sinuhé Arroyo, Christoph Bussler, Jacek Kopecký, Rubén Lara, Axel Polleres, and Michal Zaremba. Web Service Capabilities and Constraints in WSMO. In *W3C Workshop on Constraints and Capabilities for Web Services*, Redwood Shores, CA, USA, 2004.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services, Version 1.1, 2003.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services - Concepts, Architectures, Applications*. Springer, 2004.

[Act06] Active Endpoints. ActiveBPEL Engine Tutorial. Available from `http://www.active-endpoints.com/open-source-tutorial.htm`, 2006.

[ADR05] Ganand Anandalingam, Robert W. Day, and Srinivasa Raghavan. The Landscape of Electronic Market Design. *Management Science*, 51(3):316–327, March 2005.

[AG03] Samuil Angelov and Paul Grefen. The 4W Framework for B2B e-Contracting. *Int. Journal on Networking and Virtual Organisations*, 1(3), 2003.

[Aga07] Sudhir Agarwal. *Formal Description of Web Services for Expressive Matchmaking*. PhD thesis, Fakultät für Wirtschaftswissenschaften, Universität Karlsruhe (TH), May 2007.

[AL05] Sudhir Agarwal and Steffen Lamparter. User Preference based Automated Selection of Web Service Compositions. In Kunal Verma; Amit Sheth; Michal Zaremba; Christoph Bussler, editor, *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, Amsterdam, Netherlands, DEC 2005. IBM.

[AMBD04] Alain Abran, James W. Moore, Pierre Bourque, and Robert Dupuis, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society, February 2004. Available at `http://www.swebok.org/`.

[Ame06] American National Standards Institute (ANSI). ANSI ASC X12. `http://www.x12.org/`, 2006.

[AP05] Danilo Ardagna and Barbara Pernici. Global and Local QoS Guarantee in Web Service Selection. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops, BPM 2005 International Workshops*, pages 32–46, Nancy, France, September 2005.

[Ari08] Aristotle. Metaphysics. In W. D. Ross, editor, *The Works of Aristotle translated into English, Volume VIII*. Oxford University Press, Oxford, UK, 1908.

[AS06] Sudhir Agarwal and Rudi Studer. Automatic Matchmaking of Web Services. In *Proc. of IEEE International Conference on Web Services (ICWS 2006)*, pages 45–54, Chicago, USA, 2006. IEEE Computer Society.

[ATY00] Arne Andersson, Mattias Tenhunen, and Frederik Ygge. Integer Programming for Combinatorial Auction Winner Determination. In *Proc. of 4th Int. Conf. on MultiAgent Systems (ICMAS-2000)*, Washington, DC, USA, 2000. IEEE Computer Society.

[AVMM04] Rohit Aggarwal, Kunal Verma, John Miller, and William Milnor. Constraint Driven Web Service Composition in METEOR-S. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 23–30, Washington, DC, USA, 2004. IEEE Computer Society.

[AW00] Rakesh Agrawal and Edward L. Wimmers. A Framework for Expressing and Combining Preferences. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 297–306, New York, NY, USA, 2000. ACM Press.

[BAG03] Ziv Baida, Hans Akkermans, and Jaap Gordijn. Serviguration: Towards Online Configurability of Real-world Services. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, pages 111–118, New York, NY, USA, 2003. ACM Press.

[BAGS02] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *J. of Concurrency and Computation: Practice and Experience*, 14(13-15), 2002.

[Bak98] Yannis Bakos. The Emerging Role of Electronic Marketplaces on the Internet. *Commun. ACM*, 41(8):35–42, 1998.

[BAV05] Rajkumar Buyya, David Abramson, and Srikumar Venugopal. The Grid Economy. *Proceedings of the IEEE, Special Issue on Grid Computing*, 93(3):698–714, March 2005.

[BBB+05] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. Semantic Web Services Framework (SWSF). `http://www.w3.org/Submission/SWSF/`, September 2005. W3C Member Submission.

[BCG+05] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Services Composition based on Behavioral Descriptions. *International Journal of Cooperative Information Systems (IJCIS)*, 14(4):333Ű376, 2005.

[BCHS05] Stephan Bloehdorn, Philipp Cimiano, Andreas Hotho, and Steffen Staab. An Ontology-based Framework for Text Mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20(1):87–112, May 2005.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory Implemenation and Applications*. Cambridge University Press, 2003.

[BdVS02] Piero A. Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An Algebra for Composing Access Control Policies. *ACM Trans. Inf. Syst. Secur.*, 5(1):1–35, 2002.

[BF05] Piero A. Bonatti and Paola Festa. On Optimal Service Selection. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 530–538, New York, NY, USA, 2005. ACM Press.

[BFM06] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Planning with Qualitative Temporal Preferences. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 134–144, Lake District of the United Kingdom, 2006. AAAI Press.

[BG95] Fahiem Bacchus and Adam Grove. Graphical Models for Preference and Utility. In *Eleventh Conference on Uncertainty in Artificial Intelligence (UAIŠ95)*, page 3Ű10, San Francisco, CA, USA, 1995. Morgan Kaufmann.

[BGG+02] Stefano. Borgo, Aldo Gangemi, Nicloa Guarino, Claudio Masolo, and Alessandro Oltramari. Ontology RoadMap. WonderWeb Deliverable D15. `http://wonderweb.semanticweb.org`, Dec 2002.

[BH95] Franz Baader and Bernhard Hollunder. Embedding Defaults Into Terminological Knowledge Representation Formalisms. *J. Autom. Reasoning*, 14(1):149Ű180, 1995.

[BH01] Craig Boutilier and Holger H. Hoos. Bidding Languages for Combinatorial Auctions. In *International Joint Conference on Artificial Intelligence IJCAI'01*, pages 1211–1217, Seattle, Washington, USA, 2001.

[BK05] Martin Bichler and Jayant Kalagnanam. Configurable Offers and Winner Determination in Multi-attribute Auctions. *European Journal of Operational Research*, 160(2):380–394, January 2005.

[BK06a] Abraham Bernstein and Christoph Kiefer. Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In *21th Annual ACM Symposium on Applied Computing (SAC)*, pages 1684–1689, New York, NY, USA, 2006. ACM Press.

[BK06b] Peter Bruckner and Sigrid Knust. *Complex Scheduling*. Springer, 2006.

[BKK+02] Martin Bichler, Jayant Kalagnanam, Kaan Katircioglu, Alan J. King, Richard D. Lawrence, Ho Soo Lee, Grace Y. Lin, and Yingdong Lu. Applications of Flexible Pricing in Business-to-Business Electronic Commerce. *IBM Systems Journal*, 41(2), July 2002.

[BKS03] Martin Bichler, Gregory E. Kersten, and Stefan Strecker. Towards a Structured Design of Electronic Negotiations. *Group Decision and Negotiation*, 12(4):311– 335, 2003.

[BL84] Ronald J. Brachman and Hector J. Levesque. The Tractability of Subsumption in Frame-Based Description Languages. In Ronald J. Brachman, editor, *Proceedings of the National Conference on Artificial Intelligence (AAAI'84)*, pages 34–37, Austin, USA, 1984. AAAI Press.

[BLHL01] Tim Berners-Lee, Jim Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based Constraint Satisfaction and Optimization. *J. ACM*, 44(2):201–236, 1997.

[BMW+07] Jean-Sebastien Brunner, Li Ma, Chen Wang, Lei Zhang, Daniel C. Wolfson, Yue Pan, and Kavitha Srivinas. Explorations in the Use of Semantic Web Technologies for Product Information Management. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 747–756, Banff, Alberta, Canada, 2007. ACM Press.

[Boe88] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5):61–72, May 1988.

[Bor97] Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Enschede, Enschede, The Netherlands, 1997.

[Bro98] David C. Brown. Defining Configuring. *Artif. Intell. Eng. Des. Anal. Manuf.*, 12(4):301–305, 1998.

[BS85] Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2), 1985.

[BSD+04] Piero A. Bonatti, Nahid Shammehri, Claudiu Duma, Daniel Olmedilla, Wolfgang Nejdl, Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti, Paolo Coraggio, Grigoris Antoniou, Joachim Peer, and Norbert E. Fuchs. Rule-based Policy Specification: Sate of the Art and Future Work. Deliverable I2-D1, REWERSE Project (IST-2004-506779), August 2004.

[Bur05] Mike Burner. Service Orientation and Its Role in Your Connected Systems Strategy. Technical report, Microsoft Corporation, 2005.

[BVEL04] Saartje Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual modeling of OWL DL ontologies using UML. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213, Hiroshima, Japan, November 2004. Springer.

[BW03] Wolf-Tilo Balke and Matthias Wagner. Towards Personalized Selection of Web Services. In *Proceedings of the 12th International World Wide Web Conference*, Budapst, Hungary, 2003.

[CCC+04] Andrea Cali, Diego Calvanese, Simona Colucci, Tommaso Di Noia, and Francesco M. Donini. A Description Logic Based Approach for Matching User Profiles. In Volker Haarslev and Ralf Möller, editors, *Proc. of the 2004 Description Logic Workshop (DL 2004)*, volume 104 of *CEUR Workshop Proceedings*, Whistler, British Columbia, Canada, 2004. CEUR-WS.org.

[CCMP06] Cinzia Cappiello, Marco Comuzzi, Enrico Mussi, and Barbara Pernici. Context Management for Adaptive Information Systems. *Electr. Notes Theor. Comput. Sci.*, 146(1):69–84, 2006.

[CFLGP03] Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, Tools and Languages for Building Ontologies: Where is their Meeting Point? *Data Knowl. Eng.*, 46(1):41–64, 2003.

[Cha04] David A. Chappell. *Enterprise Service Bus. Theory in Practice.* O'Reilly Media, 2004.

[Che93] Yeon-Koo Che. Design Competition Through Multidimensional Auctions. *RAND J. Econom.*, 24:668Ű680, 1993.

[CM01] James Cole and Zoran Milosevic. Extending Support for Contracts in ebXML. In Maria E. Orlowska and Masatoshi Yoshikawa, editors, *ITVE '01: Proceedings of the workshop on Information technology for virtual enterprises*, pages 119–127, Queensland, Australia, 2001. IEEE Computer Society.

[CM02] Xiangping Chen and Prasant Mohapatra. Performance Evaluation of Service Differentiating Internet Servers. *IEEE Trans. Comput.*, 51(11):1368–1375, 2002.

[CP04] Li Chen and Pearl Pu. Survey of Preference Elicitation Methods. Technical report, Ecole Politechnique Federale de Lausanne (EPFL), Lausanne, Switzerland, 2004.

[CSDS03] Fabio Casati, Eric Shan, Umeshwar Dayal, and Ming-Chien Shan. Business-oriented Management of Web services. *Commun. ACM*, 46(10):55–60, 2003.

[CSM+04] Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[DAM02] DAML Services Coalition. DAML-S: Web service description for the semantic web. In *Proceedings of the First International Semantic Web Conference (ISWC)*, Sardinia, Italy, 2002.

[dBLK+05] Jos de Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel. The Web Service Modeling Language WSML. WSML Final Draft WSMO Deliverable D16, DERI, October 2005. `http://www.wsmo.org/TR/d16/d16.1`.

[dBLPF05] Jos de Bruijn, Rubén Lara, Axel Polleres, and Dieter Fensel. OWL DL vs. OWL flight: Conceptual Modeling and Reasoning for the Semantic Web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM Press.

[Ded02] Adenekan Dedeke. Self-Selection Strategies for Information Goods. *First Monday*, 7(3), 2002.

[DFK+04] Ying Ding, Dieter Fensel, Michel C. A. Klein, Borys Omelayenko, and Ellen Schulten. The Role of Ontologies in eCommerce. In *Handbook on Ontologies*, pages 593–616. Springer, 2004.

[DIN98] DIN. ISO 9735 – EDIFACT Version 4.0. UNCEC CEFACT. Information available from `http://www.unece.org/trade/untdid/welcome.htm`, 1998.

[DJP03] Rajdeep K Dash, Nicholas R. Jennings, and David C. Parkes. Computational-Mechanism Design: A Call to Arms. *IEEE Intelligent Systems*, 18(6):40–47, November 2003. Special Issue on Agents and Markets.

[DKL+05] Roman Dumitru, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel:. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77 – 106, 2005.

[DLP03] Asit Dan, Heiko Ludwig, and Giovanni Pacifici. Web Services Differentiation with Service Level Agreements. Ibm developersworks, IBM Corp., May 2003.

[DLV06] Christos Doulkeridis, Nikos Loutas, and Michalis Vazirgiannis. A System Architecture for Context-Aware Service Discovery. *Electronic Notes in Theoretical Computer Science. Proceedings of the First International Workshop on Context for Web Services (CWS 2005)*, 146(1):101–116, 2006.

[DS97] Aspassia Daskalopulu and Marek J. Sergot. The Representation of Legal Contracts. *AI and Society*, 11(1/2):6–17, 1997.

[dVV03] Sven de Vries and Rakesh V. Vohra. Combinatorial Auctions: A Survey. *INFORMS Journal on Computing*, 15(3):284–309, Summer 2003.

[Ehr06] Marc Ehrig. *Ontology Alignment - Bridging the Semantic Gap*. PhD thesis, Universität Karlsruhe (TH), Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe, 2006.

[EL04] Ahmed Elfatatry and Paul Layzell. Negotiating in Service-oriented Environments. *Commun. ACM*, 47(8):103–108, 2004.

[Erl06] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall/Pearson PTR, 2006.

[ETJ04] Cora Beatriz Excelente-Toledo and Nicholas R. Jennings. The Dynamic Selection of Coordination Mechanisms. *Journal of Autonomous Agents and Multi-Agent Systems*, 9(1-2):55–85, 2004.

[ETW04] Islam Elgedawy, Zahir Tari, and Michael Winikoff. Exact Functional Context Matching for Web Services. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 143–152, New York, NY, USA, 2004. ACM Press.

[EWL06] Yagil Engel, Michael P. Wellman, and Kevin M. Lochner. Bid Expressiveness and Clearing Algorithms in Multiattribute Double Auctions. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 110–119, New York, NY, USA, 2006. ACM Press.

[FBS99] Yuzo Fujishima, Kevin Leyton Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Int. Joint Conference of Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, July 1999. Morgan Kaufmann.

[FFJ+03] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker. Configuration Knowledge Representations for Semantic Web Applications. *Artif. Intell. Eng. Des. Anal. Manuf.*, 17(1):31–50, 2003.

[Fis70] Peter C. Fishburn. *Utility Theory for Decision Making.* Wiley, NewYork, 1970.

[FKL+05] Dieter Fensel, Uwe Keller, Holger Lausen, Axel Polleres, and Ioan Toma. What is Wrong with Web Services Discovery. In *W3C Workshop on Frameworks for Semantics in Web Services*, Innsbruck, Austria, June 2005. Available from `http://www.w3.org/2005/04/FSWS/Submissions/50/WWW_or_What_is_Wrong_with_Web_service_Discovery.pdf`.

[FMS06] Ian Foster, Tom Maguire, and David Snelling. OGSA WSRF Basic Profile 1.0. Technical report, Open Grid Forum, May 1 2006.

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. *Grid Today*, 1(6), July 22 2002. `http://www.gridtoday.com/02/0722/100136.html`.

[Gan04] Aldo Gangemi. Ontology Design Patterns. Technical Report 2004#1, Laboratory for Applied Ontology, Rome, Italy, 2004.

[GB01] Kannan Govindarajan and Arindam Banerji. HP Web Services Architecture Overview. In *W3C workshop on Web services*, San Jose, CA, USA, April 2001. W3C. Available at `http://www.w3.org/2001/03/WSWS-popa/paper36/`.

[GBCL04] Aldo Gangemi, Stefano Borgo, Carola Catenacci, and Jos Lehmann. Task Taxonomies for Knowledge Content. Deliverable d07, EU 6FP METOKIS Project, June 2004. Available from `http://metokis.salzburgresearch.at`.

[GBW+98] Frank Griffel, Marko Boger, Harald Weinreich, Winfried Lamersdorf, and Michael Merz. Electronic Contracting with COSMOS - How to Establish, Negotiate and Execute Electronic Contracts on the Internet. In C. Kobryn, C. Atkinson, and Z. Milosevic, editors, *2nd Int. Enterprise Distributed Object Computing Workshop (EDOC '98)*, page 10, La Jolla, CA, USA, November 1998.

[GCB04] Aldo Gangemi, Carola Catenacci, and Massimo Battaglia. The Inflammation Ontology Design Pattern: An Exercisein Building a Core Biomedical Ontology with Descriptions and Situations. In D. Pisanelli, editor, *Biomedical Ontologies*, Amsterdam, The Netherlands, 2004. IOS Press.

[Ger00] Ed Gerck. Overview of Certification Systems: X.509, PKIX, CA, PGP and SKIP. *The Bell*, 1(3):8, 2000.

[GF95] Michael Grüninger and Mark Fox. Methodology for the Design and Evaluation of Ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, Montréal, Québec, Canada, April 1995.

[GHA07] Stephan Grimm, Pascal Hitzler, and Andreas Abecker. Knowledge Representation and Ontologies. In Rudi Studer, Stephan Grimm, and Andreas Abecker, editors, *Semantic Web Services Ű Concepts, Technologies and Applications*, chapter 3, pages 37–88. Springer, 2007.

[GHVD03] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of 12th International Conference on the World Wide Web (WWW 2003)*, pages 48–57, Budapest, Hungary, 2003. ACM Press.

[GLA+04] Stephan Grimm, Steffen Lamparter, Andreas Abecker, Sudhir Agarwal, and Andreas Eberhart. Ontology Based Specification of Web Service Policies. In Peter Dadam and Manfred Reichert, editors, *INFORMATIK 2004 - Informatik verbindet, Band 2, Proceedings of Semantic Web Services and Dynamic Networks*, volume 51 of *LNI*, pages 579–583, Ulm, Germany, September 2004. GI.

[Glo06] Global Grid Forum. Grid Resource Allocation Agreement Protocol. Web Services Specification. Available from `http://www.ogf.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal\_sp\_tn\_jpver\_v2.pdf`, October 2006.

[GM03] Aldo Gangemi and Peter Mika. Understanding the Semantic Web through Descriptions and Situations. In *Confederated Int. Conf. DOA, CoopIS and ODBASE*, number 2888 in LNCS, Catania, Sicily, Italy, November 2003. Springer.

[GMN05] Martin Gaedke, Johannes Meinecke, and Martin Nussbaumer. Aspects of Service-Oriented Component Procurement in Web-Based Information Systems. *IJWIS*, 1(1):15–24, 2005.

[GMP04] Stephan Grimm, Boris Motik, and Chris Preist. Variance in e-Business Service Discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications, Workshop at ISWC 2004*, Hiroshima, Japan, 2004.

[GMS06] Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance Checking between Business Processes and Business Contracts. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 221–232, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[Gov05] Guido Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14:181–216, 2005.

[GP03] Benjamin Grosof and Terrence C. Poon. SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proceedings of the 12th World Wide Web Conference*, pages 340–349, Budapest, Hungary, May 2003.

[GP04] Asunción Gómez-Pérez. Ontology Evaluation. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, chapter Ontology Evaluation, pages 251–274. Springer, 2004.

[gri05] Sangyoon Oh and Geoffrey C. Fox. HHFR: A new architecture for Mobile Web Services: Principles and Implementations. Technical report, Indiana University, September 2005.

[Gru93] Tom R. Gruber. A translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GST04] Aldo Gangemi, Maria Teresa Sagri, and Daniela Tiscornia. A Constructive Framework for Legal Ontologies. Deliverable d07, EU 6FP METOKIS Project, Deliverable, 2004. Available from `http://metokis.salzburgresearch.at`.

[GST05] Aldo Gangemi, Maria Teresa Sagri, and Daniela Tiscornia. A Constructive Framework for Legal Ontologies. In Richard Benjamins, Pompeu Casanovas, Joost Breuker, and Aldo Gangemi, editors, *Law and the Semantic Web: Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications*. Springer, 2005.

[Gua97] Nicola Guarino. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In *SCIE '97: International Summer School on Information Extraction*, pages 139–170, London, UK, 1997. Springer-Verlag.

[Gua98] Nicola Guarino. Formal Ontologies and Information Systems, Preface. In *Proceedings of the 1st Conference on Formal Ontologies and Information Systems (FOIS'98)*, pages 3–15, Trento, Italy, 1998. IOS Press.

[Haa06] Peter Haase. *Semantic Technologies for Distributed Information Systems*. PhD thesis, University of Karlsruhe (TH), 2006.

[Hag96] Jaap C. Hage. A Theory of Legal Reasoning and a Logic to Match. *Artificial Intelligence and Law*, 4:199–273, 1996.

[HBM+07] R. Motahari-Nezhad Hamid, Boualem Benatallah, Axel Martens, Francisco Curbera, and Fabio Casati. Semi-automated Adaptation of Service Interactions. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 993–1002, New York, NY, USA, 2007. ACM Press.

[Hef01] Jeff Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. PhD thesis, University of Maryland, College Park, 2001.

[Hen03] Jim Hendler. On Beyond Ontology. In *Keynote talk, 2nd International Semantic Web Conference (ISWC'03)*, Sanibel Island, Florida, USA, 2003.

[Het06] Moritz Hetzer. A Semantic Bidding Language for Combinatorial Contracts. Master's thesis, University of Karlsruhe (TH) and Carnegie Mellon University, May 2006.

[HF05] Yigal Hoffner and Simon Field. Transforming Agreements into Contracts. *International Journal of Cooperative Information Systems*, 14(2-3):217–244, 2005.

[HM05] Peter Haase and Boris Motik. A Mapping System for the Integration of OWL-DL Ontologies. In Axel Hahn, Sven Abels, and Liane Haak, editors, *IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 9–16, Bremen, Germany, November 2005. ACM Press.

[Hoh13] Wesley Newcomb Hohfeld. Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. *Yale Law Journal*, 23, 1913.

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A Proposal for an OWL Rules Language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA, 2004. ACM Press.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Submission, avilable at http://www.w3.org/Submission/SWRL, May 2004.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, December 2003.

[HSTT00] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to Decide Query Containment under Constraints using a Description Logic. In *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, pages 326–343, Reunion Island, France, 2000.

[Hur73] Leonid Hurwicz. The Design of Mechanisms for Resource Allocation. *American Economic Review*, 69(2), 1973.

[IBM03] IBM Corporation. Enterprise privacy authorization language (EPAL 1.2). Available from `http://www.w3.org/Submission/EPAL`, November 2003. W3C Member Submission.

[Jac02] Matthew O. Jackson. *Mechanism Theory*. Eolss Publishers, Oxford ,UK, 2002.

[JF05] Radu Jurca and Boi Faltings. Reputation-based Service Level Agreements for Web Services. In *Service Oriented Computing (ICSOC - 2005)*, volume 3826 of *LNCS*, pages 396 – 409, Amsterdam, Netherlands, 2005.

[JM03] Ulrich Junker and Daniel Mailharro. The logic of ILOG (J)Configurator: Combining constraint programming with a description logic. In *Proc of the IJCAI-03 Configuration Workshop*, page 13Ű20, Acapulco, Mexico, 2003.

[JMG05] Michael C. Jaeger, Gero Mühl, and Sebastian Golze. QoS-aware Composition of Web Services: An Evaluation of Selection Algorithms. In Robert Meersman and Zahir Tari, editors, *Confederated International Conferences CoopIS, DOA, and ODBASE 2005*, volume 3760 of *LNCS*, pages 646–661, Agia Napa, Cyprus, 2005. Springer.

[JW05] Hai Jin and Hao Wu. Semantic-enabled Specification for Web Services Agreement. *International Journal of Web Services Practices*, 1(1–2):13–20, 2005.

[Kag04] Lalana Kagal. *A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments*. PhD thesis, University of Maryland Baltimore County, Baltimore MD 21250, November 2004.

[Kan72] Stig Kangar. Law and Logic. *Theoria*, 38:105–132, 1972.

[Kar72] R.M. Karp. Reducibility among Combinatorial Problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, page 85Ű103. Plenum Press, 1972.

[Kar03] Alan H. Karp. Rules of Engagement for Automated Negotiation. Technical Report HPL-2003-152, HP Laboratories Palo Alto, 2003.

[KC03] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, Januar 2003.

[KFJ04] Lalana Kagal, Tim Finin, and Anupam Joshi. Declarative Policies for Describing Web Service Capabilities and Constraints. In *W3C Workshop on Constraints and Capabilities for Web Services*, Oracle Conference Center, Redwood Shores, CA, USA, October 2004. W3C.

[KFKS05] Matthias Klusch, Benedikt Fries, Mahboob Khalid, and Katia Sycara. OWLS-MX: Hybrid semantic web service retrieval. In *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 2005. AAAI Press.

[KFS06] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

[Kha02] Rania Khalaf. Business Process with BPEL4WS: Learning BPEL4WS, Part 2. IBM developerWorks, available from `http://www.ibm.com/developerworks/webservices/library/ws-bpelcol2/`, August 2002.

[Kie02] Werner Kießling. Foundations of Preferences in Database Systems. In *28th International Conference on Very Large Data Bases (VLDB'02)*, pages 311–322, Hong Kong, China, 2002.

[KK04] Markus Keidl and Alfons Kemper. Towards Context-aware Adaptable Web Services. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65, New York, NY, USA, 2004. ACM Press.

[KKR04] Michael Klein and Birgitta König-Ries. Integrating Preferences into Service Requests to Automate Service Usage. In *First AKT Workshop on Semantic Web Services*, Milton Keynes, UK, Dezember 2004.

[KKRKS07] Ulrich Küster, Birgitta König-Ries, Michael Klein, and Mirco Stern. DIANE - A Matchmaking-Centered Framework for Automated Service Discovery, Composition, Binding and Invocation. In *Proceedings of the 16th International World Wide Web Conference (WWW2007)*, Banff, Alberta, Canada, May 2007.

[KKS06] John Kemp, Tapio Kaukonen, and Timo Skytta. *Mobile Web Services: Architecture and Implementation*. Wiley & Sons, 2006.

[KL04] Alexander Keller and Heiko Ludwig. Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen. *Praxis der Informationsverarbeitung und Kommunikation*, 27(2), June 2004. In German.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KNT00] Gregory E. Kersten, Sunil J. Noronha, and Jeffrey Teich. Are All E-Commerce Negotiations Auctions? In *COOP'2000: Fourth Int. Conference on the Design of Cooperative Systems*, Sophia-Antipolis, France, May 2000.

[Kop05] Jacek Kopeck. Aligning WSMO and WSDL-S. WSMO Working Draft WSMO Deliverable D30, DERI, August 2005. `http://www.wsmo.org/TR/d30/v0.1/`.

[Koz83] David Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 24:333–354, 1983.

[KP06] Vladimir Kolovski and Bijan Parsia. WS-Policy and Beyond: Application of OWL Defaults to Web Service Policies. In *2nd International Semantic Web Policy Workshop (SWPW'06). Workshop at the 5th International Semantic Web Conference (ISWC)*, November 2006.

[KPKH05] Vladimir Kolovski, Bijan Parsia, Yarden Katz, and James A. Hendler. Representing Web Service Policies in OWL-DL. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 461–475. Springer, 2005.

[KR76] Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. J. Wiley, New York, 1976.

[KRMF06] Jacek Kopecky, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic Web Services Grounding. In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW'06)*, Guadeloupe, French Caribbean, 2006.

[Kru03] Philippe Kruchten. *The Rational Unified Proces: An Introduction*. Addison-Wesley, 3rd edition, 2003.

[KW04] Jeffrey O. Kephart and William E. Walsh. An Artificial Intelligence Perspective on Autonomic Computing Policies. In *Proc. of 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, pages 3–12, New York, USA, June 2004.

[LA05] Steffen Lamparter and Sudhir Agarwal. Specification of Policies for Automatic Negotiations of Web Services. In Lalana Kagal, Tim Finin, and James Hendler, editors, *Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference*, pages 99–109, Galway, Ireland, November 2005.

[LA07] Steffen Lamparter and Anupriya Ankolekar. Automated Selection of Configurable Web Services. In *8. Internationale Tagung Wirtschaftsinformatik (WI2007)*, pages 441–460, Karlsruhe, Germany, 2007.

[LAGS07] Steffen Lamparter, Anupriya Ankolekar, Stephan Grimm, and Rudi Studer. Preference-based Selection of Highly Configurable Web Services. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 1013–1022, Banff, Canada, May 2007.

[Lan94] Thomas Langenohl. *Systemarchitekturen elektronischer Märkte*. PhD thesis, University of St. Gallen, St. Gallen, Switzerland, 1994.

[LAO+06] Steffen Lamparter, Anupriya Ankolekar, Daniel Oberle, Rudi Studer, and Christof Weinhardt. A Policy Framework for Trading Configurable Goods and Services in Open Electronic Markets. In *Proceedings of the 8th Int. Conference on Electronic Commerce (ICEC'06)*, pages 162–173, New Brunswick, Frediction, Canada, August 2006.

[LASW06] Steffen Lamparter, Anypriya Ankolekar, Rudi Studer, and Christof Weinhardt. Towards a Policy Framework for Open Electronic Markets. In Thomas Dreier, Rudi Studer, and Christof Weinhardt, editors, *Information Management and Market Engineering*, volume 4 of *Studies on eOrganisation and Market Engineering*, pages 11–28. Universitätsverlag Karlsruhe, 2006.

[LDK04] Heiko Ludwig, Asit Dan, and Robert Kearney. Cermona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 65–74, New York, NY, USA, 2004. ACM Press.

[LEO05] Steffen Lamparter, Andreas Eberhart, and Daniel Oberle. Approximating Service Utility from Policies and Value Function Patterns. In *6th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, pages 159–168, Stockholm, Sweden, June 2005. IEEE Computer Society.

[Ley05] Frank Leymann. The (Service) Bus: Services Penetrate Everyday Life. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *Third International Conference on Service-Oriented Computing (ICSOC 2005)*, Lecture Notes in Computer Science, pages 12–20, Amsterdam, The Netherlands, December 2005. Springer.

[LF04] Chuang Liu and Ian Foster. A Constraint Language Approach to Matchmaking. In *IEEE International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 7–14, March 2004.

[LGPJ97] Mariano Fernández López, Asunción Gómez-Pérez, and Natalia Juristo. METHONTOLOGY: From Ontological Art to Ontological Engineering. In *Workshop on Ontological Engineering. AAAI Spring Symposium*, Stanfort, USA, 1997.

[LGPSS99] Mariano Fernández López, Asunción Gómez-Pérez, Juan Pazos Sierra, and Alejandro Pazos Sierra. Building a Chemical Ontology Using Methontology and the Ontology Design Environment. *IEEE Intelligent Systems*, 14(1):37–46, 1999.

[LH03] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 331–339, Budapest, Hungary, 2003. ACM Press.

[LHVA05] Manfred Hauswirth Le-Hung Vu and Karl Aberer. QoS-based Service Selection and Ranking with Trust and Reputation Management. In *13th International Conference on Cooperative Information Systems (CoopIS 2005)*, Agia Napa, Cyprus, October 2005.

[Lin77] Lars Lindahl. Position and Change - A Study in Law and Logic. *Synthese Library*, 112, 1977.

[LKD+03] Heiko Ludwig, Alexander Keller, Asit Dan, Richard King, and Richard Franck. A Service Level Agreement Language for Dynamic Electronic Services. *Electronic Commerce Research*, 3(1-2):43–59, 2003.

[LL87] M. Lacroix and Pierre Lavency. Preferences; Putting More Knowledge into Queries. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, pages 217–225, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[LLM07] Steffen Lamparter, Stefan Luckner, and Sybille Mutschler. Formal Specification of Web Service Contracts for Automated Contracting and Monitoring. In *40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, page 63b, Waikoloa, Hawaii, USA, 2007. IEEE Computer Society.

[LLP+07] Holger Lausen, Rubeén Lara, Axel Pollers, Jos de Bruijn, and Dumitru Roman. Description. In Rudi Studer, Stephan Grimm, and Andreas Abecker, editors, *Semantic Web Services Ű Concepts, Technologies and Applications*, chapter 7, pages 157–186. Springer, 2007.

[LLSG04] Guoming Lai, Cuihong Li, Katia Sycara, and Joseph Andrew Giampapa. Literature Review on Multi-attribute Negotiations. Technical Report CMU-RI-TR-04-66, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2004.

[LML+05] Steffen Lamparter, Sibylle Mutschler, Stefan Luckner, Kendra Stockmar, and Carolina M. Laborde. A Modeling Perspective on Web Service Contracting. In *Proceedings of the EDOC Workshop on Contract Architectures and Languages (CoALa'05)*, Enschede, The Netherlands, September 2005. IEEE Computer Society.

[LNZ04] Y. Liu, A. H. Ngu, and L. Z. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. In *WWW Alt. '04: Proc. of 13th Int. WWW Conf. on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004.

[LS98] Markus A. Lindemann and Beat F. Schmid. Elements of a Reference Model for Electronic Markets. In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 4*, Hawaii, USA, 1998.

[LS06] Steffen Lamparter and Björn Schnizler. Trading Services in Ontology-driven Markets. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1679–1683, Dijon, France, 2006. ACM Press.

[Lud03] Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[LW66] Eugene L. Lawler and David E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(6):699 – 719, 1966.

[LWJ01] Alessio Lomuscio, Michael Wooldridge, and Nicholas R. Jennings. A Classification Scheme for Negotiation in Electronic Commerce. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, pages 19–33, London, UK, 2001. Springer-Verlag.

[LYFA02] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*, page 63, Washington, DC, USA, 2002. IEEE Computer Society.

[LZR03] Zhangxi Li, Huimin Zhao, and Sathya Ramanathan. Pricing Web Services for Optimizing Resource Allocation Ű An Implementation Scheme. In *Proc. of the Web2003, Seattle, WA*, 2003.

[MAPG03] Tim Moses, Anne Anderson, Seth Proctor, and Simon Godik. XACML Profile for Web Services. Available from `http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf`, September 2003. Oasis Working Draft.

[Mar06] David Martin. Putting Web Services in Context. *Electr. Notes Theor. Comput. Sci.*, 146(1):3–16, 2006.

[MBG99] Marco Casassa Mont, Adrian Baldwin, and Cheh Goh. POWER Prototype: Towards Integrated Policy-Based Management. Technical Report HPL-1999-126, HP Labs, October 1999.

[MBG+02a] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltramari, and Luc Schneider. The WonderWeb Library of Foundational Ontologies. WonderWeb Deliverable D17, August 2002. Available from `http://wonderweb.semanticweb.org`.

[MBG+02b] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltramari, and Luc Schneider. The WonderWeb Library of Foundational Ontologies: Preliminary Report. WonderWeb Deliverable D17, ISTC-CNR, Padova, Italy, August 2002.

[MBG+03] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. Ontology Library. WonderWeb Deliverable D18. `http://wonderweb.semanticweb.org`, 2003.

[MCP+99] Thomas W. Malone, Kevin Crowston, Brian T. Pentland, Jintae Lee, Chrysanthos Dellarocas, G. Wyner, John Quimby, Charles S. Osborn, Abraham Bernstein, George A. Herman, Mark Klein, and Elisa O'Donnell. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, 45(3):425–443, March 1999.

[MDRCD+03] Octavio Martín-Díaz, Antonio Ruiz-Cortés, Amador Durán, David Benavides, and Miguel Toro. Automating the Procurement of Web Services. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Service-Oriented Computing - ICSOC 2003*, volume 2910 of *LNCS*, pages 91–103, Trento, Italy, 2003.

[MF89] Sanjay Mittal and Felix Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1395 – 1401, Detroit, MI, USA, 1989.

[MG05] Zoran Milosevic and Guido Governatori. Special Issue on Contract Architectures and Languages – Guest Editors' Introduction. *International Journal of Cooperative Information Systems*, 14(2-3):73–76, 2005.

[MGP04] Craig McKenzie, Peter Gray, and Alun Preece. Extending SWRL to Express Fully-Quantified Constraints. In *RuleML 2004 Workshop at ISWC 2004*, Hiroshima, Japan, 2004. Springer.

[Mil95] Zoran Milosevic. *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Computer Science Dept. The University of Queensland, October 1995.

[Mil00] Paul R. Milgrom. An EconomistŠs Vision of the B-to-B Marketplace. Executive white paper, www.perfect.com, 2000.

[MIP+98] Deborah L. McGuinness, Charles Isbell, Matt Parker, Peter Patel-Schneider, Lori Alperin Resnick, and Chris Welty. A Description logic-based Configurator on the Web. *SIGART Bull.*, 9(2):20–22, 1998.

[MLM+06] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. OASIS Reference Model for Service Oriented Architectures 1.0. `http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf`, August 2006.

[MM87] R. Preston McAfee and John McMillan. Auctions and Bidding. *J. of Economic Literatur*, 25(2):699–738, June 1987.

[MM03] Daniel J. Mandell and Sheila McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 227–247, Sanibel Island, FL, USA, October 2003. Springer.

[MMW06] Jeffrey K. MacKie-Mason and Michael P. Wellman. Automated Markets and Trading Agents. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Computational Economics*, volume 2 of *Handbook of Computational Economics*, chapter 28, pages 1381–1431. Elsevier, 2006.

[MOGS04] Peter Mika, Daniel Oberle, Aldo Gangemi, and Marta Sabou. Foundations for service ontologies: Aligning OWL-S to DOLCE. In *Proc. of the 12th Int. Conf. on WWW*, New York, NY, USA, 2004. ACM.

[MOR01] James Bret Michael, Vanessa L. Ong, and Neil C. Rowe. Natural-Language Processing Support for Developing Policy-Governed Software System. In *Proceedings of the 39th International Conference on Technology for Object- Oriented Languages and Systems*, pages 263–274, Santa Barbara, California, USA, July 2001. IEEE Computer Society Press.

[Mot05] Boris Motik. On the Properties of Metamodeling in OWL. In *Proc. of the 4th Int. Semantic Web Conf. (ISWC 2005)*, pages 548–562, Galway, Ireland, November 2005.

[MS84] Thomas W. Malone and Stephen A. Smith. Tradeoffs in Designing Organizations: Implications for New Forms of Human Organizations and Computer Systems. Working papers no. 112. Working paper, Massachusetts Institute of

Technology (MIT), Sloan School of Management, April 1984. Available from `http://ideas.repec.org/p/mit/sloanp/2075.html`.

[MS04] E. Michael Maximilien and Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, September-October 2004.

[MS06] Boris Motik and Ulrike Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, pages 227–241, Phnom Penh, Cambodia, November 2006.

[MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[Mur05] Richard Murch. How Policy-Based Computing Can Automate Tasks and Reduce Costs. `http://www.phptr.com/articles/article.asp?p=361809&rl=1`, January 2005. Pearson Education, Prentice Hall PTR.

[MW82] Paul R. Milgrom and Robert J. Weber. A Theory of Auctions and Competitive Bidding. *Econometrica*, 50(5):1089–1122, 1982.

[MYB87] Thomas W. Malone, Joanne Yates, and Robert I. Benjamin. Electronic Markets and Electronic Hierarchies. *Commun. ACM*, 30(6):484–497, 1987.

[Nas50] John Nash. The Bargaining Problem. *Econometrica*, 18:155–162, 1950.

[Nas53] John Nash. Two-person Cooperative Games. *Econometrica*, 21:128–140, 1953.

[Neu04] Dirk Neumann. *Market Engineering - A Structured Design Process for Electronic Markets*. PhD thesis, Department of Economics and Business Engineering, University of Karlsruhe (TH), Karlsruhe, 2004.

[Nis00] Noam Nisan. Bidding and Alllocation in Combinatorial Auctions. In *Proceedings of the 2nd ACM conference on Electronic commerce (EC'00)*, pages 1–12, New York, NY, USA, 2000. ACM Press.

[NLS06] Dirk Neumann, Steffen Lamparter, and Björn Schnizler. Automated Bidding for Trading Grid Services. In *Proceedings of the European Conference on Information Systems (ECIS'06)*, Göteborg, Sweden, June 2006.

[NP01] Ian Niles and Adam Pease. Towards A Standard Upper Ontology. In *FOIS '01: Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 2–9, Ogunquit, Maine, USA, 2001. ACM Press.

[NSDM03] Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. A System for Principled Matchmaking in an Electronic Marketplace. In *WWW '03: Proceedings of the Twelfth International Conference on World Wide Web*, pages 321–330, Budapest, Hungary, 2003. ACM Press.

[OAS04] OASIS. WS-Reliability Version 1.1. `http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf`, September 2004.

[OAS06a] OASIS. Web Services Business Process Execution Language (WS-BPEL). `http://www.oasis-open.org/apps/org/workgroup/wsbpel/`, 2006.

[OAS06b] OASIS. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`, February 2006. OASIS Standard Specification.

[OAS06c] OASIS ebXML Joint Committee. Organization for the Advancement of Structured Information Standards, Enabling a global electronic market: ebXML. `http://www.ebxml.org`, 2006.

[Obe05] Daniel Oberle. *Semantic Management of Middleware*. PhD thesis, University of Karlsruhe(TH), 2005.

[OCM+07] Leo Obrst, Werner Ceusters, Inderjeet Mani, Steve Ray, and Barry Smith. The Evaluation of Ontologies. In Christopher J. O. Baker and Kei-Hoi Cheung, editors, *Semantic Web Revolutionizing Knowledge Discovery in the Life Sciences*, pages 139–158. Springer, 2007.

[OEH02] Justin O'Sullivan, David Edmond, and Arthur ter Hofstede. What's in a service?: Towards accurate description of non-functional service properties. *Distributed and Parallel Databases Journal - Special Issue on E-Services*, 12:117–133, 2002.

[OLG+06] Daniel Oberle, Steffen Lamparter, Stephan Grimm, Denny Vrandecic, Steffen Staab, and Aldo Gangemi. Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems. *Journal of Applied Ontology*, 2(2):163–202, 2006.

[OLPL04] Daniel Olmedilla, Rubén Lara, Axel Polleres, and Holger Lausen. Trust Negotiation for Semantic Web Services. In *1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, volume 3387 of *Lecture Notes in Computer Science*, pages 81–95, San Diego, CA, USA, jul 2004. Springer.

[OR05] Martin J. Osborne and Ariel Rubinstein. Bargaining and Markets. Levine's Bibliography 666156000000000515, UCLA Department of Economics, Feb 2005.

[OVSH06] Nicole Oldham, Kunal Verma, Amit Sheth, and Farshad Hakimpour. Semantic WS-Agreement Partner Selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, Edinburgh, Scotland, 2006. ACM Press.

[PA05] Cesare Pautasso and Gustavo Alonso. Flexible Binding for Reusable Composition of Web Services. In T. Gschwind, U. Aßmann, and O. Nierstrasz, editors, *Proc. of the 4th Workshop on Software Composition (SC 2005)*, volume 3628 of *LNCS*, pages 151–166, Edinburgh, Scotland, April 2005. Springer.

[Pap03] Michael P. Papazoglou. Service-oriented Computing: Concepts, Characteristics and Directions. In *Proc. of the 4th Int. Conference on Web Information Systems*, pages 3–12, Rome, Italy, December 2003.

[Par01] David Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, Univesity of Pennsylvania, May 2001.

[Pas06] Adrian Paschke. Verification, Validation and Integrity of Distributed and Interchanged Rule Based Policies and Contracts in the Semantic Web. In *International Semantic Web and Policy Workshop (SWPW'06) at ISWC'06*, Athens, Georgia, USA, 2006.

[PBD05] Adrian Paschke, Martin Bichler, and Jens Dietrich. ContractLog: An Approach to Rule Based Monitoring and Execution of Service Level Agreements. In *International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005)*, Galway, Ireland, 2005.

[PBS+06] Kosmas Petridis, Stephan Bloehdorn, Carsten Saathoff, Nikos Simou, Stamatia Dasiopoulou, Vassilis Tzouvaras, Siegfried Handschuh, Yannis Avrithis, Yiannis Kompatsiaris, and Steffen Staab. Knowledge Representation and Semantic Annotation of Multimedia Content. *IEEE Proceedings on Vision, Image and Signal Processing - Special issue on the Integration of Knowledge, Semantics and Digital Media Technology*, 153(3):255–262, June 2006.

[PG03] Michael P. Papazoglou and Dimitrios Georgakopoulos. Service-Oriented Computing. *Commun. ACM*, 46(10):24–28, 2003.

[PH06] Michael P. Papazoglou and Willem-Jan Van Den Heuvel. Service-oriented Design and Development Methodology. *International Journal of Web Engineering and Technology*, 2(4):412 – 442, 2006.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic Matching of Web Services Capabilities. In Ian Horrocks and James A. Hendler, editors, *The Semantic Web - ISWC 2002, First International Semantic Web Conference*, volume 2342 of *LNCS*, pages 333–347, Sardinia, Italy, 2002. Springer.

[PM04] Helena Sofia Andrade N. P. Pinto and João Pabãa Martins. Ontologies: How can They be Built? *Knowledge Information System*, 6(4):441–464, 2004.

[POSV04] Abhijit Patil, Swapna Oundhakar, Amit Sheth, and Kunal Verma. METEOR-S Web Service Annotation Framework. In *The 13th International World Wide Web Conference Proceedings*, pages 553–563, New York, NY, USA, May 2004. ACM.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Englewood Cliffs, N.J.: Prentice Hall, 1982.

[Ran03] Shuping Ran. A Model for Web Services Discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[RDNDS+07] Azzurra Ragone, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Simona Colucci, and Francesco Colasuonno. Fully Automated Web Services Discovery and Composition through Concept Covering and Concept Abduction. *International Journal of Web Services Research (JWSR)*, 4(3), 2007.

[RDS89] John A. Reinecke, Gary Dessler, and William F. Schoell. *Introduction to Business - A Contemporary View*. Allyn and Bacon, Boston, 1989.

[RFC99] Hypertext Transfer Protocol – HTTP/1.1 (IETF RFC 2616). `http://tools.ietf.org/html/rfc2616`, June 1999.

[RFC01] Terminology for Policy-Based Management. `http://www.rfc-archive.org/getrfc.php?rfc=3198`, November 2001.

[RFC05] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005.

[RN03] Stuart Russel and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall Series in Artificiall Intelligence, second edition, 2003.

[Rob04] Rick Robinson. Understand Enterprise Service Bus scenarios and solutions in Service-Oriented Architecture - The role of the Enterprise Service Bus. Technical report, IBM developerWorks, 2004. Available from `http://www-128.ibm.com/developerworks/webservices/library/ws-esbscen/`.

[Roy70] Winston Royce. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, 26:1–9, August 1970.

[RPH98] Michael H. Rothkopf, Alexander Pekec, and Ronald M. Harstad. Computationally Managable Combinatorial Auctions. *Management Science*, 44:1131 – 1147, 1998.

[RS01] Jean-Charles Rochet and Lars A. Stole. The Economics of Multidimensional Screening. Technical report, Graduate School of Business, University of Chicago, Chicago, IL., 2001.

[RWG01] Daniel M. Reeves, Michael P. Wellman, and Benjamin N. Grosof. Automated Negotiation from Declarative Contract Descriptions. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 51–58, New York, NY, USA, 2001. ACM Press.

[SABG05] Thomas Skylogiannis, Grigoris Antoniou, Nick Bassiliades, and Guido Governatori. DR-NEGOTIATE - A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 44–49, Washington, DC, USA, 2005. IEEE Computer Society.

[San99] Tuomas Sandholm. An Algorithm for Optimal Winner Determination in Combinatorial Auctions. In *Proceedings of the 16th Int. Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, 1999.

[San02] Tuomas Sandholm. Algorithm for Optimal Winner Determination in Combinatorial Auctions. *Artif. Intell.*, 135(1-2):1–54, 2002.

[SBF98] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.

[SBM+04] Quan Sheng, Boualem Benatallah, Zakaria Maamar, Marlon Dumas, and Anne Ngu. Enabling Personalized Composition and Adaptive Provisioning of Web Services. In *Proceedings 16th International Conference on Advanced Information Systems Engineering*, volume 3084 of *LNCS*, pages 322–337, Riga, Latvia, 2004.

[Sch93] Beat Schmid. Elektronische Märkte. *Wirtschaftsinformatik*, 35(2):465–480, 1993. In German.

[Sch02] Roy W. Schulte. Predicts 2003: Enterprise Service Buses Emerge. *Gartner Report*, December 9th 2002.

[Sch03] Luc Schneider. How to Build a Foundational Ontology: The Object-Centered High-level Reference Ontology OCHRE. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *KI 2003: Advances in Artificial Intelligence, 26th Annual German Conference on AI*, volume 2821 of *LNCS*, pages 120–134, Hamburg, Germany, September 2003. Springer.

[Sch05] Michael Schwind. Design of Combinatorial Auctions for Allocation and Procurement Processes. In *CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 391–395, Washington, DC, USA, 2005. IEEE Computer Society.

[SDBW03] Kristof Schneider, Christine Daun, Hermann Behrens, and Daniel Wagner. Vorgehensmodelle und Standards zur systematischen Entwicklung von Dienstleistungen. In Hans-Jörg Bullinger and August-Wilhelm Scheer, editors, *Service Engineering*, pages 113–138. Springer, 2nd edition, 2003. In German.

[Ser01] Marek Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.

[SGA07] Rudi Studer, Stephan Grimm, and Andreas Abecker, editors. *SemanticWeb Services – Concepts, Technologies and Applications*. Springer, 2007.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley and Sons, 2005.

[Slo94] Morris Sloman. Policy Driven Management For Distributed Systems. *Plenum Press Journal of Network and Systems Management*, 2(4):333–360, 1994.

[SMS+02] Akhil Sahai, Vijay Machiraju, Mehmet Saya, Aad van Moorsel, and Fabio Casati. Automated SLA Monitoring for Web Services. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*, 2002.

[SMSA+05] Tanveer Fathima Syeda-Mahmood, Gauri Shah, Rama Akkiraju, Anca-Andreea Ivan, and Richard Goodwin. Searching Service Repositories by Combining Semantic and Ontological Matching. In *IEEE International Conference on Web Services (ICWS 2005)*, pages 13–20, Orlando, FL, USA, 2005. IEEE Computer Society.

[SNVW06] Bjoern Schnizler, Dirk Neumann, Daniel Veit, and Christof Weinhardt. Trading Grid Services - A Multi-attribute Combinatorial Approach. *European Journal of Operational Research*, 2006.

[SPAS03] Katja Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1), 2003.

[SPT06] Wolf Siberski, Jeff Z. Pan, and U. Thaden. Querying the Semantic Web with Preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC)*, Athens, GA, USA, 2006.

[SR03] Amit P. Sheth and Cartic Ramakrishnan. Semantic (Web) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis. *IEEE Data Engineering Bulletin*, 26(4):40–48, 2003.

[SRT05] Amit P. Sheth, Cartic Ramakrishnan, and Christopher Thomas. Semantics for the Semantic Web: The Implicit, the Formal and the Powerful. *Int. J. Semantic Web Inf. Syst.*, 1(1):1–18, 2005.

[SS04a] Raghuram M. Sreenath and Munindar P. Singh. Agent-based Service Selection. *Journal of Web Semantics*, 1(3), 2004.

[SS04b] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer Verlag, 2004.

[SSA+01] York Sure, Rudi Studer, Hans Akkermans, Victor Iosif, Uwe Krohn, Thorsten Lau, Bernd Novotny, Hans-Peter Schnurr, and Fredrik Ygge. On-To-Knowledge Methodology – Employed and Evaluated Version. Technical report, On-To-Knowledge Project Deliverable D16, 2001. Available at `http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OTK-D16_v1-0.pdf`.

[SSSC06] Jaime Martín Serrano, Joan Serrat, John Strassner, and Ray Carroll. Policy-Based Management and Context Modelling Contributions for Supporting Services in Autonomic Systems. In Dominique Gaïti, Guy Pujolle, Ehab S. Al-Shaer, Kenneth L. Calvert, Simon A. Dobson, Guy Leduc, and Olli Martikainen, editors, *Autonomic Networking, First International IFIP TC6 Conference (AN 2006)*, volume 4195 of *Lecture Notes in Computer Science*, pages 172–187, Paris, France, September 2006. Springer.

[Sto04] Nenad Stojanovic. A Logic-Based Approach for Query Refinement. In *2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004)*, pages 477–480, Beijing, China, September 2004.

[Str02] John Strassner. How Policy Empowers Business-Driven Device Management. In *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, pages 214–217, Monterey, CA, USA, June 2002. IEEE Computer Society.

[Stu97] Markus Stumptner. An Overview of Knowledge-based Configuration. *AI Commun.*, 10(2):111–125, 1997.

[SW98] Daniel Sabin and Rainer Weigel. Product Configuration Frameworks – A Survey. *IEEE Intelligent Systems*, 13(4):42–49, 1998.

[SW03] Michael Ströbel and Christof Weinhardt. The Montreal Taxonomy for Electronic Negotiations. *Group Decision and Negotiation*, 12:143–164, 2003.

[SY05] Rizos Sakellariou and Viktor Yarmolenko. On the Flexibility of WS-Agreement for Job Submission. In *MGC '05: Proceedings of the 3rd International Workshop on Middleware for Grid Computing*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[TBJ+03] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In *Proceedings of the 2nd International Semantic Web Conference (ISWC'03)*, Sanibel Island, Florida, USA, 2003.

[TBP03] David Trastour, Claudio Bartolini, and Chris Preist. Semantic Web Support for the Business-to-Business e-Commerce Pre-contractual Lifecycle. *Computer Networks*, 42(5):661–673, 2003.

[TF06] Ioan Toma and Douglas Foxvog. Non-functional Properties in Web Services. WSML Working Draft (March 10, 2006) D28.4 v0.1, DERI Technical Report, March 2006.

[TFJ+06] Ioan Toma, Douglas Foxvog, Michael C. Jaeger, Dumitru Roman, Thomas Strang, and Dieter Fensel. Modeling QoS characteristics in WSMO. In Karl M. Göschka, Schahram Dustar, Frang Leymann, and Stefan Tai, editors, *Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 42 – 47. ACM, November 2006.

[Tos05] Vladimir Tosic. *Service Offerings for XML Web Services and Their Management Applications*. PhD thesis, Department of Systems and Computer Engineering, Carleton University, Canada, 2005.

[TPDW05] Valentina Tamma, Steve Phelps, Ian Dickinson, and Michael Wooldridge. Ontologies for Supporting Negotiation in e-Commerce. *Engineering applications of artificial intelligence*, 18(2):223–236, March 2005.

[TPP02] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. WSOL - Web Service Offerings Language. In *Web Services, E-Business, and the Semantic Web: CAiSE 2002 International Workshop, WES 2002. Revised Papers*, volume 2512 of *LNCS*, Toronto, Canada, May 27-28 2002.

[TPS06] Christoph Tempich, H. Sofia Pinto, and Steffen Staab. Ontology Engineering Revisited: an Iterative Case Study with DILIGENT. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *LNCS*, pages 110–124, Budva, Montenegro, JUN 2006. Springer.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[TT98] Yao-Hua Tan and Walter Thoen. A Logical Model of Directed Obligations and Permissions to Support Electronic Contracting. *Int. J. Electronic Commerce*, 3(2):87–104, 1998.

[UBJ+04] Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Stuart Aitken. KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[UK95] M. Uschold and M. King. Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95*, Montreal, Canada, 1995.

[VAG+04] Kunal Verma, Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Juhnyoung Lee. On Accommodating Inter Service Dependencies in Web Process Flow Composition. In *AAAI Spring Symposium on Semantic Web Services*, Stanford, CA, USA, 2004.

[Var97] Hall .R. Varian. Versioning Information Goods. In *Digital Information and Intellectual Property Conference*, Harvard University, USA, 1997.

[VHPA06] Le-Hung Vu, Manfred Hauswirth, Fabio Porto, and Karl Aberer. A Search Engine for QoS-enabled Discovery of Semantic Web Services. *International Journal of Business Process Integration and Management*, 1(4):244 – 255, 2006.

[vNM47] John von Neumann and Oskar Morgenstern. *Theory of Games and Economics Behavior*. Princeton University Press, Princeton, NJ, 1947.

[vR79] C. J. Keith van Rijsbergen. *Information Retrieval*. Butterworths, London, 2 edition, 1979.

[VSFGS06] Denny Vrandecic, Mari Carmen Suárez-Figueroa, Aldo Gangemi, and York Sure, editors. *Proceedings of the Int. Workshop on Evaluation of Ontologies (EON), in conjunction with the 15th WWW conference*, Edinburgh, United Kingdom, 2006.

[W3C01a] World Wide Web Consortium (W3C). DAML+OIL Web Ontology Language. `http://www.w3.org/TR/daml+oil-reference`, December 2001.

[W3C01b] World Wide Web Consortium (W3C). Web Services Definition Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001.

[W3C03] World Wide Web Consortium (W3C). SOAP Version 1.2 Part 1: Messaging Framework. `http://www.w3.org/TR/soap12-part1/`, June 2003. W3C Recommendation.

[W3C04a] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.1. `http://www.w3.org/TR/xml11`, February 2004. W3C Recommendation.

[W3C04b] World Wide Web Consortium (W3C). Resource Description Framework (RDF). http://www.w3.org/RDF/, 2004.

[W3C04c] World Wide Web Consortium (W3C). Web Ontology Language (OWL). `http://www.w3.org/2004/OWL/`, 2004.

[W3C04d] World Wide Web Consortium (W3C). Web Services Architecture Requirements. `http://www.w3.org/TR/wsa-reqs`, February 2004.

[W3C04e] World Wide Web Consortium (W3C). XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2/, October 2004. W3C Recommendation.

[W3C05] World Wide Web Consortium (W3C). Web Services Addressing (WS-Addressing) 1.0. `http://www.w3.org/Submission/ws-addressing/`, May 2005. W3C Recommendation.

[W3C06a] World Wide Web Consortium (W3C). SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, October 2006. Working Draft.

[W3C06b] World Wide Web Consortium (W3C). Web Services Policy Framework 1.5 (WS-Policy). `http://www.w3.org/2002/ws/policy/`, July 2006.

[W3C07a] World Wide Web Consortium (W3C). Semantic Annotations for WSDL and XML Schema. `http://www.w3.org/TR/sawsdl/`, January 2007. W3C Candidate Recommendation.

[W3C07b] World Wide Web Consortium (W3C). W3C XML Query (XQuery 1.0). `http://www.w3.org/XML/Query/`, January 2007. W3C Recommendation.

[WBCL02] Christopher Ward, Melissa J. Buco, Rong N. Chang, and Laura Z. Luan. A Generic SLA Semantic Model for the Execution Management of E-Business Outsourcing Contracts. In *EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies*, pages 363–376, London, UK, 2002. Springer-Verlag.

[WD92] Michael P. Wellman and Jon Doyle. Modular Utility Representation for Decision-Theoretic Planning. In *Proc. of 1st Int. Conf. on AI Planing Systems 1992 (AIPS-92)*, pages 236–242, June 1992.

[WHN03] Christof Weinhardt, Carsten Holtmann, and Dirk Neumann. Market-Engineering. *Wirtschaftsinformatik*, 45(6):635–640, 2003.

[WKLW98] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin Core Metadata for Resource Discovery. IETF RFC2413, September 1998.

[WPBB01] Rich Wolski, James S. Plank, John Brevik, and Todd Bryan. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. *Int. J. of High Performance Computing Applications*, 15(3), 2001.

[WTKD04] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility Functions in Autonomic Systems. In *International Conference on Autonomic Computing (ICAC-04)*, 2004.

[WVKT06] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A QoS-Aware Selection Model for Semantic Web Services. In Asit Dan and Winfried Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006, 4th International Conference*, volume 4294 of LNCS, pages 390–401, 2006.

[WWC+05] Guijun Wang, Changzhou Wang, Alice Chen, Haiqin Wang, Casey Fung, Stephen Uczekaj, Yi-Liang Chen, Wayne Guthmiller Guthmiller, and Joseph Lee. Service Level Management using QoS Monitoring, Diagnostics, and Adaptation for Networked Enterprise Systems. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, volume 0, pages 239–250, Enschede, The Netherlands, 2005. IEEE Computer Society.

[WWW01] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. A Parametrization of the Auction Design Space. *Games and Economic Behavior*, 35(1-2):304–338, April 2001.

[YL05] Tao Yu and Kwei-Jay Lin. Service Selection Algorithms for Web services with End-to-End QoS Constraints. *Information Systems and E-Business Management*, 3(2):103–126, 2005.

[Zad65] Lofti A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338Ű–353, 1965.

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[ZKG04] Olaf Zimmermann, Pal Krogdahl, and Clive Gee. Elements of Service-Oriented Analysis and Design - An Interdisciplinary Modeling Approach for SOA Projects. `http://www-128.ibm.com/developerworks/library/ws-soad1/`, June 2004. IBM developerWorks.

[ZWX06] Xiaobo Zhou, Jianbin Wei, and Cheng-Zhong Xu. Resource Allocation for Session-Based Two-Dimensional Service Differentiation on e-Commerce Servers. *IEEE Trans. Parallel Distrib. Syst.*, 17(8):838–850, 2006.