Mass Customization of Cloud Services

Engineering, Negotiation and Optimization

Zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften

(Dr. rer. pol.)

von der Fakultät für Wirtschaftswissenschaften des Karlsruher Instituts für Technologie (KIT)

genehmigte Dissertation

von

Dipl.-Wi.-Ing. Steffen Haak

Tag der mündlichen Prüfung: 20.02.2013
Referent: Prof. Dr. Christof Weinhardt
Korreferent: Prof. Dr. Rudi Studer

Karlsruhe, 2013

Abstract

The fundamental shift from mass production to mass customization has been fostered by the rise of information technology (IT). Yet, it has primarily affected traditional manufacturing goods. Despite IT's role as enabler for mass customization, IT services – and in particular Cloud computing services – are currently not offered in a mass customization fashion. Consumers either are faced with the effort of building a service customized to their needs themselves or can select a more or less appropriate off-the-shelf Software-as-a-Service offer.

Several challenges hinder the entry of mass customization principles to providers of Cloud computing services. Firstly, provider and consumer need to find a way of communicating functional and non-functional requirements. This requires a simple negotiation mechanism helping both parties to find an agreement on non-functional aspects like Quality-of-Service (QoS) and price. Secondly, the service engineering on provider side needs to be automated. In order to do so, knowledge on service resources and offered functionalities has to be available as externalized knowledge, so that it can be used for an automated service configuration process. Thirdly, most Cloud computing service offers can be provided through a vast set of different service configurations, providing the same functionality, yet differing in quality and price. Finding the optimal choice within such a set with respect to both consumer preferences and accruing costs can be computationally complex, requiring adequate and efficient optimization techniques.

The work at hand addresses these challenges through technical and economic contributions. On the technical side, a framework for a semantic description of service functionalities, resources, dependency and interoperability information as well as meta information on quality and cost is provided. Further, concept and implementation of an ontology update mechanism, keeping the externalized knowledge up to date, are contributed. A service engineering algorithm enables Cloud providers to use this knowledge to derive viable service configurations for specific consumer requests.

On the economic side, three different multi-attributive negotiation mechanisms are compared. Two are stylized versions of existing mechanisms, one is newly introduced. All mechanisms are individual rational, budget balanced and benefit from a simple bidding language. Based on this comparison, an optimization model and various optimization techniques, which become necessary in such a negotiation scenario, are developed to cope with the challenge of computational complexity in QoS-aware service configuration settings. All results are evaluated analytically or through simulation studies, where appropriate.

Acknowledgments

Composing a work like this is an honorable, but cumbersome task. A task, that would not have been possible without the guidance and support of many people, contributing both ideas and valuable feedback in many different ways. I am especially indebted to my advisor Prof. Dr. Christof Weinhardt for giving me the chance to work in his group, providing me a productive and efficient environment and giving his constant support in and beyond my research. He granted me the freedom to pursue my dissertation and supported me with valuable guidance and advice throughout my entire time as research assistant. I would also like to show my gratitude to my co-advisor Prof. Dr. Rudi Studer for his guidance and fruitful discussions especially on the knowledge management part of this work and possible extensions thereof. Further thanks go to the other members of the committee, Prof. Dr. Thomas Setzer for his valuable feedback and the rich discussions on optimization techniques, and Prof. Dr. Marc Wouters, who has been so kind to guide me through a fair and insightful defense.

Furthermore, I would like to show my gratitude to the entire team of the research group on Information and Market Engineering at the Institute of Information Systems and Marketing (IISM) as well as to all my colleagues of the research division on Information Process Engineering (IPE) at the FZI Research Center for Information Technology. Special thanks go to Dr. Stephan Grimm and Dr. Jens Wissmann for their ideas and constructive criticism regarding the technical part of this work, Dr. Henner Gimpel for his constant support and guidance in and beyond the fields of negotiations and mechanism design, and Prof. Dr. Thomas Setzer and Dr. Jochen Martin for their valuable feedback on the optimization part of this work. In addition, I am grateful to Dr. Benjamin Blau, Dr. Tobias Conte and Dr. Clemens van Dinther for their ideas and support in the early phase of my research. Special thanks go to Rico Knapper for supporting me in all kinds of situations. Concerning the readability of the work at hand, I am indebted to Nico Rödder for his constant and diligent proofreading and his constructive feedback. I am especially indebted to my loyal companions Rico Knapper and Dr. Jochen Martin, not only for contributing valuable input to my research, but, most importantly, for always providing me a fun atmosphere at work.

Most notably, I want thank my parents Hannelore and Michael as well as my sister Jennifer for their constant and loving support throughout my life. Finally, I am particularly grateful to my girlfriend Andrea. Without her constant love and patience, this work would not have been possible.

Steffen Haak

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ABox	Assertional Box
API	Application Programming Interface
BIP	Binary Integer Program(ming)
BPEL	Business Process Execution Language
CSP	Constraint Satisfaction Problem
CWA	Closed World Assumption
DFS	Depth First Search
ETL	Extract, Transform, Load
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
ICT	Information and Communication Technology
IP	Integer Program(ming)
IS	Information Systems
IT	Information Technology
JSON	JavaScript Object Notation
LP	Linear Program(ming)
MAUT	Multi-Attribute Utility Theory
OCR	Optical Character Recognition
OWA	Open World Assumption
OWL	Web Ontology Language
PaaS	Platform-as-a-Service
PDF	Portable Document Format
QoS	Quality of Service
REST	Representational State Transfer
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol And RDF Query Language
SVN	Service Value Network
TBox	Terminological Box
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
XML	eXtensible Markup Language

XSLT	Extensible Stylesheet Language Transformation

Part I

Foundations and Preliminaries

Chapter 1

Introduction

Today, customization of products and services is well-set in almost any domain. Cars are individualized by consumers via configurators, pre-fabricated houses can be customized and even everyday products like food and clothes are subject to mass customization [151]. For services, individual and close interaction between consumer and provider happens almost by definition. With technological progress and rising customer expectations as mutually depending elements, off-the-shelf offerings to fit the greatest common denominator of potential consumer demands go out of date in favor of individualized solutions.

Historically, mass customization is a logical progression of the era of mass markets [89]. It is defined as the ability to offer individually designed products and services to every customer on a large scale basis [42]. High process agility, flexibility and integration as well as automation are prerequisites for mass customization in the manufacturing world. Besides innovative processes, information technology (IT) has been identified as one of the main enablers, mainly driving automation and customer involvement [42]. New businesses, selling customizable products over the Web, continue to appear and replace the classical off-the-shelf markets. Initially starting with high-priced goods like cars (e.g. *Audi*[1] or *BMW*[2]), this trend has also captured medium-priced goods like clothes. A prominent example in this segment is *TailorStore*[3], an online retailer selling custom-made shirts and other business clothes to the masses with almost an infinite set of potential customer choices, be it the fabric, the cuff or even the style of the collar. Further examples are *Herrenschmiede*[4] or *Amerano*[5]. And even low-priced everyday products like groceries nowadays are sold customizable via the Web, as in the case of *mymuesli.com*, where one can get a cereal blend of one's very own individual choice.

On the downside, consumers are faced with an increased complexity in their purchasing decision. While creating one's own cereal blend or deciding on the stereo or color of a car still seems feasible, configuration of complex technical products can become a challenge only experts can master. Even higher complexity is faced

[1]`http://konfigurator.audi.de/`
[2]`http://www.bmw.de/de/de/general/configurations_center/configurator.html`
[3]`http://www.tailorstore.de`
[4]`http://www.herrenschmiede.de/`
[5]`http://www.amerano.de/`

by providers offering customized products and services. Not only do they have to assure that consumers do not create offers which are technically infeasible, they also have to automate both assembly and pricing of their custom offers in order to achieve large scale volumes, as required in a mass customization market.

While IT is a main driver for mass customization [42], the above mentioned complexity is the major reason for the lack of mass customized Cloud computing service offers in reality. Without a doubt, Cloud computing has been recognized as the most important trend in recent years [34], promising the "long-held dream" of computing as a utility [6] with benefits like eased scalability, virtually endless resources and pay-as-you-go pricing schemes. Whether it is in an end-user or business context, today, consumers who are willing to use or integrate Cloud services[6] from external service providers are confronted with a crucial design decision, yielding two different options: Either (1) they build the required business functionality themselves, using a set of highly customizable commodity services, or (2) they use an out-of-the-box service offer designed to achieve a distinct objective. The first option is commonly implemented by means of so called Infrastructure-as-a-service (IaaS) or Platform-as-a-Service (PaaS)[7] offers, the second option is commonly referred to as Software-as-a-Service (SaaS). By their nature, SaaS offers are designed to fit the greatest common denominator of consumer requirements and offer little room for customization. Consumers building their own applications based on IaaS offers might obtain services that perfectly fit their expectations, but come at the cost of an immense complexity on the side of the consumer. Pre-configured images for IaaS offers can help to reduce this complexity. According to *thecloudmarket.com*, there are already over 46,000[8] different virtual machine images from various providers that can be run as virtual appliances on Amazon's Elastic Compute Cloud (EC2) platform. However, these images are not built on-demand to meet individual consumer needs.

While this large variety gives further proof of the importance of mass customized service offers, it becomes obvious that consumers have very individual needs. Even with these images consumers have little support for the technical and economic decision process of selecting an appropriate image, for deploying the image on the Cloud or for making further customizations. In addition, expert knowledge in configuration and deployment of Cloud resources is a prerequisite for creating such individually engineered services, as it is highly unlikely that an image exists which is already configured exactly according to the individual needs.

Unquestionably, a service provider taking on the quest of offering Cloud services tailored to individual consumer needs in a mass customization fashion would have a significant economic impact. Yet, many new challenges arise when shifting the above mentioned complexity from the consumer to the service provider. Mass customization requires the offer creation to follow an automated process. Within this process, providers and consumers first have to find a way of communicating requests for customized services in a structured, well-formed and standardized way.

[6]In the remainder of this thesis the term *Cloud service* is used synonymously for *Cloud computing service*.

[7]IaaS offers raw compute resources, whereas PaaS is a combination of IaaS with a stack of software development frameworks on top if it.

[8]Source: `http://thecloudmarket.com/stats`, last access in September 2012

Such a request comprises both functional and non-functional requirements. The functional part thereby has to be based on some sort of catalog describing the generic capabilities offered by the provider. Without such a catalog, consumers will not be able to formulate the functional requirements of their request – or at least not in a sense that request and capabilities have a chance to match each other. Once the generic functional requirements are defined, the provider will have to derive feasible service configurations based on the resources available to him.

Having defined a specific functionality, the agreement on non-functional attributes of a mass customized service is not a black and white decision. Consumers may have an optimal configuration in mind which would perfectly fulfill their needs along multiple quality dimensions, paired with a maximum willingness to pay. Offerings that yield slightly lower quality may still be good enough, yet come along with a decreased willingness to pay. A mechanism helping provider and consumer to find a satisfactory agreement on both quality and price for both parties has to be established. This mechanism closely goes in hand with the provider being able to find the optimal configuration – optimal with respect to the consumer's non-functional requirements and her willingness to pay – from an economic profit-maximizing perspective. The last step of the automated process occurs, once both parties have agreed on both the functional and non-functional part of the service offer, as the actual service deployment occurs to complete the process.

The challenges above to some extent also apply to managed services and therefore are not restricted to Cloud services. Given the flexibility, short life cycles and pay-as-you-go pricing schemes of Cloud computing, lowering transaction costs through automation is particularly important for Cloud service offers. Hence, this work primarily addresses the challenges in a Cloud computing setting. Nevertheless, given the similarity of some challenges, the results might also be transferable to the offering process of traditional IT service offers. While this thesis cannot provide a complete and final solution to all of theses challenges, the contributions provided in the following can be a step towards mass customized Cloud service offers.

1.1 Research Questions and Contributions

It is the goal of this thesis to address the obstacles indicated in the previous section. The research within this work is hence dedicated to the overall challenge of automating the offer process for individualized Cloud services. The focus hereby lies on three distinct research topics.

The engineering part is set in the context of automatic service composition, i.e. the question on how to derive feasible service alternatives based on abstract functional consumer requirements. Related work for this topic mainly can be found in the areas of Web service composition (e.g. [16]) and ontology-based configuration (e.g. [153]). The first research question hereby addresses a problem from knowledge management by trying to externalize expert knowledge on available service resources[9]. Modeling these resources does not only mean describing single entities

[9]Within this work, a *service resource* is defined as abstraction of any technical resource (or any combination of technical resources) which does not fulfill an entire business functionality by itself.

and possible abstractions in a taxonomy, but also interdependencies and compatibilities among these resources. The offer creation process for a mass customized Cloud service needs to reflect the non-functional consumer preferences, thus also requiring meta information on Quality-of-Service (QoS) and price to be modeled. As the externalized knowledge base is to be used for creating service offers for individual service requests, it has to be designed so that feasible service configurations can be derived automatically. The information on QoS and price needs to be included so that the externalized knowledge can be used to derive economically sound, i.e. profit maximizing, decisions with respect to non-functional consumer preferences, as these influence the potential revenue achievable with a given consumer request.

Research Question 1 $\prec$Knowledge Management for Service Resources$\succ$. *Is it possible to design a semantic service description framework where one can model knowledge on service resources so that technically feasible service configurations matching the functional requirements of a consumer request can be derived automatically?*

Closely related to the design of such a framework is the issue of information up-to-dateness. Information in any decision making context needs to reflect the most current state of nature. An ontology (or any other kind of knowledge base) is of little use if the contained information is outdated. While keeping the structural knowledge, i.e. information on existence and relationship of service resources, is hardly feasible in an automated manner, simple data values like the QoS and price of resources fluctuate substantially and can be kept up-to-date by an automated information aggregation approach. For the designated approach to be applicable in the context of mass customizing Cloud services, it has to be scalable with respect to the number of service resources requiring updates. In addition, the information contained within the framework has to be accessible from shared environments, as in any business setting typically more than one system and one user is required to have access to the knowledge base. Based on these considerations the second research question can be derived.

Research Question 2 $\prec$Ontology Update Mechanism$\succ$. *Is it possible to design an automated ontology update mechanism which integrates information on QoS or price from manifold data sources in an automated and structured manner, so that the ontology's data values are kept up-to-date and can be accessed in a shared environment, and can such a mechanism be scalable?*

Having laid the groundwork by externalizing knowledge on service resources in a formal manner and by keeping meta-information on QoS and price up-to-date, the question remains open on how to derive feasible service configuration alternatives for individual consumer requests.

To achieve the previously mentioned complexity reduction on the side of the consumer, the requests are assumed to be of abstract manner. The consumer selects an abstract functionality or any combination of functionalities, e.g. a content management system and an online survey tool, which she is willing to purchase. The selected capabilities are abstract in a sense, that the functional requirements formulated in the request describe the desired functionality from a business perspective, but do not contain the entire service configuration from a detailed technical perspective. The exact configuration, or different potential configurations to be more precise, is supposed to be derived automatically using the externalized knowledge from the service description framework.

Similar approaches can be found in the field of software dependency management systems (e.g. APT [141]), where the consumer selects an application of her desire and the dependency management system takes care of installing complementary software required by the chosen application. Such an approach, which is fundamentally different to the literature in the area of Web service composition, would be a novelty to the field of Cloud service engineering. We can rephrase these considerations as the following research question.

Research Question 3 $\prec$SERVICE ENGINEERING ALGORITHM$\succ$. *Is it possible to design an algorithm that is capable of deriving all feasible service configuration alternatives based on a set of abstract functional requirements originating from a semantic service description framework, and can that algorithm be scalable?*

Once being able to derive feasible service configurations which meet the functional requirements of a consumer request for an individualized Cloud service, it remains open which configuration to select and how to price the service offer in awareness of non-functional consumer preferences regarding the resulting QoS. The negotiation part of this thesis is dedicated to a game-theoretic consideration on suitable mechanisms in this context. Both provider and consumer do not only have to agree on the functionality provided by the service, but also on non-functional properties like QoS and price. While finding the optimal quality strongly influences the overall added value of the agreement, the price is merely a vehicle of distributing this value among the two negotiating parties. Especially in an automated online scenario as within this thesis, different mechanisms guiding provider and consumer through the negotiation can lead to different results – impacting both the efficiency of the negotiation, i.e. how close is the negotiated agreement to the Pareto-optimal agreement, and the individual value obtained.

Strong theoretical results in the area of bilateral trading, like the seminal Theorem 1 from Myerson and Satterthwaite [121], along with other work from the area of multi-attributive procurement auctions (e.g. [21]) are the surrounding for this research focus. Given quasi-linear preferences, it is impossible to design a mechanism that achieves individual rationality, efficiency, and budget balance at once, regardless whether incentive compatibility is fulfilled or not [121]. Incentive compatibility, however, is the prerequisite for an efficient outcome. Budget balance and individual rationality are required to enable sustainability and implementability over time [126].[10] Hence, the named desiderata need to be balanced. The considerations in this area are coined from a practical perspective. Therefore, the focus lies on the sustainability and implementability of the negotiation mechanism, leading to the following research question.

Research Question 4 $\prec$MULTI-ATTRIBUTIVE NEGOTIATION MECHANISMS$\succ$. *What are possible budget balanced and individual rational mechanisms for bilaterally negotiating quality and price of a service and how do they perform with respect to the economic desiderata incentive compatibility and ex-post efficiency?*

Finally – given one has found a suitable automated negotiation mechanism – the optimization part covers the challenge of finding the profit maximizing configuration from the derived set of feasible alternatives. The derived set is hereby assumed to be available as output of the custom service algorithm in a form similar to a Service Value Network (SVN) [23], where paths or subgraphs of the network resemble different configuration alternatives. The profit to be maximized is depending on both quality and price, with the quality of a certain configuration simultaneously influencing the provider's costs and the consumer's willingness to pay, which, hence, also influences the price achievable in the negotiation. Clearly, both negotiation mechanism and optimization algorithm are closely intertwined.

The topic of finding optimal service configurations in a QoS and price-aware manner is related to the winner determination problem in multi-attributive or combinatorial procurement auctions [18, 126] and to the winner determination problem in complex service auctions [28]. In the latter, the problem is identical to finding an optimal path in an SVN. Within these problems, optimization techniques like shortest-path algorithms (e.g. [44]) and linear mathematical programming approaches are applied for optimization. Similar approaches are applicable for the problem instance in the context of optimizing mass customized Cloud services described in this thesis.

Depending on the problem formulation, many different challenges arise as the computational complexity is depended on the formal structure of the SVN, the profit maximizing objective function and the aggregation of QoS attribute values, like the service's response time or availability.

Research Question 5 $\prec$OPTIMIZATION TECHNIQUES FOR SERVICE CONFIGURATION$\succ$. *What are possible optimization techniques for finding the profit maximizing service configuration with respect to QoS and price from a set of alternatives (given as a graph structure) and how do they perform with respect to runtime and allocative optimality?*

[10]For the reader not familiar with these concepts, Section 2.3.2 gives a brief explanation on the classical mechanism design desiderata and other important foundations from mechanism design.

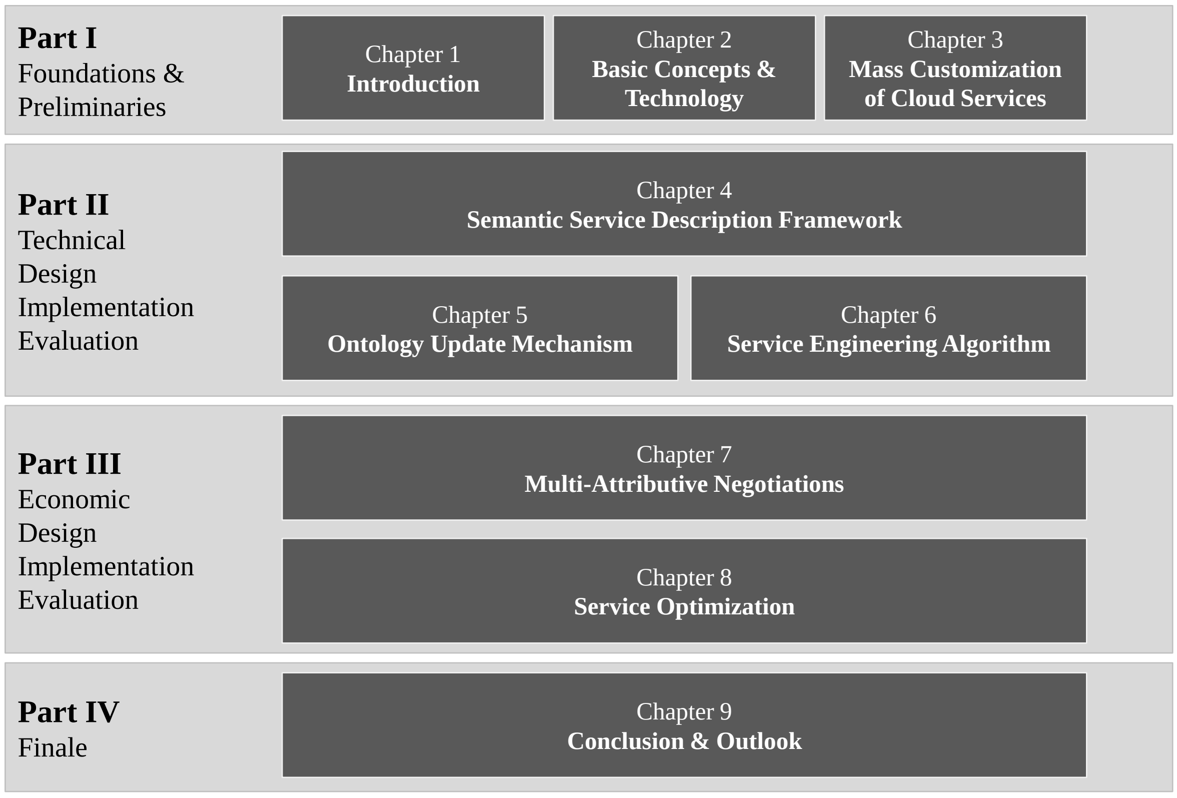


Figure 1.1: Structure of the thesis

The five research questions above are of synoptic nature and give an overview on the research contributions presented in the subsequent chapters. The structure of the thesis, which is presented in the following section, is closely oriented towards these research questions, with each research question being investigated on in an individual chapter.

1.2 Outline

The work at hand is subdivided into four parts. Part I includes essential foundations, i.e. an introduction on basic concepts and technologies, and a more detailed description of the overall scenario of mass customizing Cloud services. Part II concentrates on the technical contributions answering Research Questions 1 to 3, whereas Part III contains the economic contributions of the thesis addressing Research Questions 4 and 5. The individual contributions are divided into separate chapters, each chapter containing the corresponding requirements, related work, implementation (where applicable) and evaluation. Part IV concludes the thesis, by summarizing the contributions, shedding light on limitations and highlighting future research directions.

A high-level illustration of the structure of the thesis is depicted in Figure 1.1. Part I is opened by the current chapter. It introduces the reader to the topic of mass customization in the context of Cloud services and gives an overview on the thesis's structure, the research development and publications. The following Chapter 2 provides the reader with a broad understanding on underlying concepts and technologies, which are prerequisites for the actual research contributions following in Parts II and III. Chapter 3 is a detailed presentation of the overall scenario, which sets the context for the work at hand.

Part II introduces the technical design, implementation and evaluation of this thesis. Chapter 4 addresses Research Question 1 by presenting a semantic service description framework which enables providers to model their service resources so that the resulting knowledge base can be used to derive mass customized services in an automated manner by means of an algorithm. The latter is presented in Chapter 6, trying to answer Research Question 3. Such a knowledge base requires constant information updates. Both, concept and implementation of a mechanism, to achieve such updates automatically is given in Chapter 5, covering Research Question 2.

The economic contributions follow in Part III. Addressing Research Question 4, Chapter 7 investigates on three bilateral negotiation mechanisms for finding agreements on QoS and price between provider and consumer. In such a negotiation scenario, the provider is faced with the decision on what service to offer to fulfill the functional requirements of the consumer while maximizing his own profit. In a scenario, where many different configurations lead to the same functionality, but different quality levels, optimization approaches are required to find the profit maximizing configuration. A detailed investigation on different optimization techniques in this context are presented in Chapter 8, tackling Research Question 5.

Lastly, Part IV concludes the thesis. In the final chapter, the key contributions are revisited and summarized. Further, limitations of the work at hand are critically analyzed and future work as well as complementary topics are addressed.

1.3 Publications and Research Development

The research topic of this thesis originated in a prior research project called *SemPIT* [11] founded by IBM as Center for Advanced Studies. The goal of SemPIT was to help IBM by supporting their service offer creation process through using semantic technologies in the planning phase of a service offer. While the project resulted in a patent application [14], the achieved ontology-based solution was limited to validating and economically evaluating service offers. It, however, served as initial starting point for the semantic description framework described in Chapter 4.

A similar ontology-based approach for designing and ranking blueprints of RESTful service mashups was presented at the *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web*, a workshop held in conjunction with the *World Wide Web Conference 2009* (WWW) and published in the corresponding proceedings [24].

The overall concept of this thesis was presented at and influenced by the doctoral consortium prior to the Int. Conference on Wirtschaftsinformatik 2011 and published in its proceedings [62]. The semantic service description framework and

[11]Semantic and Policy-Based IT Management and Provisioning

the algorithm for deriving consumer specific Cloud services were presented at the *8th Extended Semantic Web Conference* and published in its proceedings [65].

The findings on multi-attributive negotiation mechanisms have been submitted to the *Int. Conference on Wirtschaftsinformatik 2013* [64].

A minor part of the chapter dealing with optimization techniques describing an efficient approximation for aggregating multiplicative QoS values (like a service's availability) was presented at the *45th Hawaii International Conference on System Sciences 2012* (HICSS) and published in its proceedings [63].

Further, complementary work in the area of autonomic benchmarking for Cloud infrastructures has been presented at the *1st IEEE/ACM Workshop on Autonomic Computing for Economics*, co-located with the *8th IEEE/ACM International Conference on Autonomic Computing* (ICAC), and published in its proceedings [66]. However, it is not included in the work at hand.

Consequently, parts of the above listed publications were adopted verbatim throughout this thesis.

Chapter 2

Basic Concepts and Technologies

In this chapter, fundamental definitions and technologies required throughout this thesis are presented. Section 2.1 covers the relevant aspects of the field of knowledge management, i.e. conceptualization, persistence of and reasoning on structured knowledge using semantic technologies. The subsequent Section 2.2 describes and defines services, Service-Oriented Architectures, Cloud services, mass customization of services and Service Value Networks. Economic foundations follow in Section 2.3. An introduction to optimization techniques is provided in Section 2.4. The chapter is concluded by a summary given in Section 2.5.

2.1 Knowledge Management

The notion of knowledge has been an important issue in philosophy ever since the ancient Greek era. Knowledge management – as an attempt to treat knowledge as an organizational resource – has gained importance in recent years, especially with the ongoing transformation from an industrial society to a knowledge society in the most developed countries. In this trend, IS research has fostered knowledge management systems with the aim to support creation, transfer, and application of knowledge in organizations [2]. Semantic technologies have evolved as central part to such knowledge management systems. Within this context, this section covers the technological foundations relevant for managing the knowledge necessary for mass customization of Cloud services.

The section is structured as follows: Ontologies, as a way to formally describe shared concepts, i.e. explicitly defined knowledge, are described in Section 2.1.1. Concrete formalisms to model, serialize and work with ontologies are presented in Section 2.1.2. Formalized and machine-readable knowledge can be used to draw conclusions based on logical reasoning. A brief overview on reasoning and available reasoners is also given in Section 2.1.2. Lastly, Section 2.1.3 gives an overview on modern concepts for storing and administrating ontologies in a centralized way.

2.1.1 Ontologies

The term ontology was first coined by Aristotle [5] describing a branch of meta physics that investigates on the philosophical problem of existence. In computer science an ontology is commonly understood as a knowledge base containing formalized knowledge over a certain domain of interest. There exist many different definitions. A very descriptive one is the following: "The role of ontologies is to capture domain knowledge in a generic way and to provide a commonly agreed upon understanding of a domain. The common vocabulary of an ontology, defining the meaning of terms and their relations, is usually organized in a taxonomy and contains modeling primitives such as classes, relations, functions, and axioms." [69] Gruber defines an ontology as a formal explicit specification of a shared conceptualization [59]. This notion has been widely adopted and is commonly used within the Semantic Web community, leading to the following definition by Studer et al. [145]:

Definition 2.1 [ONTOLOGY]. *An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest.*

Conceptualization refers to the focus being on describing things in an abstract manner, i.e. trying to find generalizations, rather than describing individuals. The formal and explicit specification ensures readability and interpretability by machines, a major advantage over unstructured information. The formal nature further allows to derive additional knowledge implicitly contained in the ontology by means of logical reasoning, thus reducing the modeling overhead, as implicit information does not have to be stated explicitly. Reasoning can also be used to detect inconsistencies, revealing statements with conflicting declarations.

The following elements (adapted from [41]) are most important for understanding and modeling an ontology:

Concept Concepts, often referred to as classes, are used to describe a broad range of things, be it abstract or concrete, atomic or composite, fictious or real. Other than instances, concepts do not describe individual objects, as they typically refer to several instances with common properties. Concepts are typically organized in taxonomies, defining sub- and super-relationships.

Example: The concept *car* can be used to define anything motorized that has four wheels and can transport one or more people from location A to location B.

Taxonomy A taxonomy expresses a hierarchy over different concepts, defining inheritance relations between super- and sub-concepts.

Example: $car \sqsubseteq vehicle$, defining that a car is a specification of a vehicle, inheriting all properties associated with vehicles in general.

Instance Instances, or individuals as they are also referred to, are similar to instantiations of a class in object-oriented programming. They embody the described objects contained in an ontology. Every element in the ontology that is not a concept, is an instance of a certain concept.

Example: A blue Porsche 911 existing in real life could be an instance of the concept *car*.

Relation Relations define the type of interaction between two or more instances. Generally they can be arbitrary n-ary sets. In the most widely used ontology languages they are binary. The domain and range of a relation is specified on the conceptual level.

Example: The relationship defining the ownership of the above mentioned Porsche could be defined as "Porsche911 *has-owner* SomePerson".

Axiom Axioms are sentences that are considered to be always true, without them being proved or demonstrated. The truth for an axiom is taken granted. Thus axioms build the foundation for any further logical deduction from the ontology. Axioms can be used to model constraints on information, deduce new information or verify the correctness of information.

Example: $car \sqsubseteq vehicle \sqcap_4 hasWheel$, i.e. cars are vehicles with four wheels.

The knowledge base of an ontology can be divided into two types of knowledge, the terminological box (TBox) and the assertional box (ABox). The TBox contains the conceptual or terminological knowledge, i.e. concepts, axioms and relation definitions. The ABox contains the assertional knowledge, i.e. instances and their relations. The way knowledge is described in the ABox depends on the formal conceptualization of the TBox.

2.1.2 Ontology Formalisms

This section covers the formalisms OWL, SWRL and SPARQL for representing and querying knowledge. A complete picture of the Semantic Web stack[1] is depicted in Figure 2.1. The formalisms presented in the following are the basic technology for the technical approach in the work at hand. The Web Ontology Language (OWL) is the World Wide Web Consortium (W3C)[2] standardized family of knowledge representation languages for authoring ontologies. Within this thesis, it was chosen as formal language for knowledge representation as it is the most widely adopted language yielding the expressiveness needed while simultaneously offering a wide support by authoring tools and reasoning engines. The main ideas behind OWL and its logical foundations are discussed in Section 2.1.2.1. The Semantic Web Rule Language (SWRL) can be used as extension to OWL, offering increased expressiveness through Horn-like rules. It is described in Section 2.1.2.3. SPARQL, covered in Section 2.1.2.4, is a graph-based query language similar to SQL[3], allowing to retrieve subsets of interest from the ontology. Lastly, a further advantage of formalized knowledge is the possibility of using this knowledge for reasoning, i.e. to infer logical consequences which are implicitly contained, but not explicitly stated. OWL reasoning is described in Section 2.1.2.5.

[1]`http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/0130-sb-W3CTechSemWeb.pdf`

[2]World Wide Web Consortium `www.w3.org`

[3]Structured Query Language

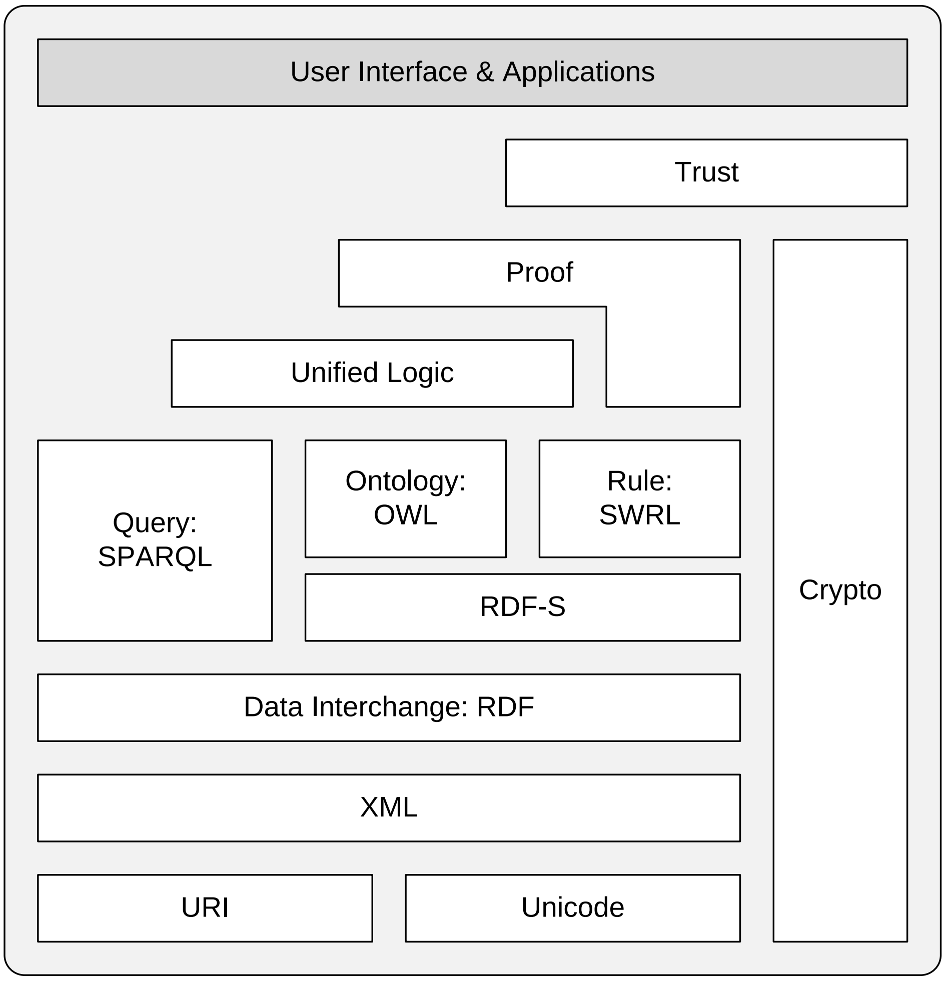


Figure 2.1: Semantic Web Stack

2.1.2.1 OWL

OWL [106, 12] was developed in the effort to promote the mutual understanding of knowledge concepts in distributed and shared environments. In 2004, the expressive ontology language became a recommendation of the W3C. Historically evolving from SHOE and DAML+OIL, its aim is to offer a widely-accepted and backward compatible standard for knowledge sharing on the Web. OWL has its logical foundation in the first-order logic called description logics. OWL ontologies can be serialized to either XML[4] or XML/RDF[5] documents. OWL offers three variants differing in their expressiveness (cf. Section 2.1.2.2).

In 2009, OWL 2 was introduced as an advancement to OWL. It adds new functionality with respect to OWL [111]. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressiveness, including:

- keys
- property chains

[4]Extensible Markup Language (`www.w3.org/XML/`)
[5]Resource Description Framework (`www.w3.org/RDF/`)

- richer data types, data ranges
- qualified cardinality restrictions
- asymmetric, reflexive, and disjoint properties
- enhanced annotation capabilities

OWL 2 is fully backward compatible, thus ontologies authored in OWL can be used in OWL 2 without additional effort.

2.1.2.2 Description Logics

The development of description logics stems from frame languages like *KL-ONE*, a system for representing knowledge in Artificial Intelligence programs [32]. It provided the logical foundation for interpreting individuals, unary concepts and binary roles in between them. Concepts can be constructed by both concept and role constructors. In addition, terminological axioms define the relation between concepts and roles, while assertional facts are statements about individuals of these concepts and their properties.

The computational tractability for computing inferences over the beliefs encoded within a knowledge-based systems, i.e. reasoning, has been a major focus of research in the past years [31]. The tractability is characterized by the trade-off between expressiveness of a knowledge representation language and the difficulty of reasoning over it. From a practical perspective, soundness and completeness of the the derived logical consequences are important factors. However, depending on the expressiveness, algorithms ensuring both soundness and completeness do not necessarily always terminate. If termination by inference algorithms cannot be guaranteed, the logic behind a knowledge representation language is called *undecidable*.

Whether a description logic is decidable depends on the allowed constructs for concepts and roles, all of which define the expressiveness of the language. An overview on different constructs for description logics is given in Table 2.1. As mentioned in the previous section, OWL comes in three different variants offering different subsets of these constructs:

OWL Lite is smallest and least expressive fragment of OWL. It is equivalent to the description logic $\mathcal{SHIF}(\mathbf{D})$ It is also decidable with a worst-case complexity of $ExpTime$, yet it is little expressive. It lacks some language constructs from OWL DL like *oneOf*, *unionOf*, *disjointWith* and some others. As it yields very few advantages over OWL DL, it only has sparse support by current software technology.

OWL DL is the most prominent variant and a fragment of OWL Full. It contains all features of OWL Lite, extending it with nominals. In contrast to OWL Full it is decidable with a worst-case complexity of $NExpTime$. *DL* stands for the description logic $\mathcal{SHOIN}(\mathbf{D})$, which is equivalent to the decidable fragment of first-order logic. It has some major restrictions over RDF Schema, e.g. a concept cannot be an instance of another concept.

Symbol	Construct
$\mathcal{AL}$	Attributive Language: conjunction, universal value restriction and limited existential quantification
$\mathcal{C}$	disjunct and full existential quantification
$\mathcal{R}+$	transitive role
$\mathcal{S}$	abbreviation for $\mathcal{ALCR}+$
$\mathcal{H}$	role hierarchy (*rdfs:subPropertyOf*)
$\mathcal{I}$	inverse role
$\mathcal{F}$	functional role
$\mathcal{O}$	nominals (*owl:oneOf*, *owl:hasValue*)
$\mathcal{Q}$	qualified number restrictions
$\mathcal{N}$	unqualified number restrictions (*owl:cardinality*)
D	concrete domains

Table 2.1: Description logic constructs [8]

Symbol	Semantics	Example	Description
$\equiv$	Class equivalence	$A \equiv B$	concept A equivalent to B
$\sqsubseteq$	Class inclusion	$A \sqsubseteq B$	A is a subset of B
$\sqcap$	Class intersection	$C \equiv A \sqcap B$	C equiv. to intersection of A and B
$\sqcup$	Class union	$C \equiv A \sqcup B$	C equiv. to union of A and B
:	Role assertion	$(x,y) : r$ $r(x,y)$	x is asserted with role r to y
$\exists$	Existential restriction	$C \sqsubseteq \exists r.A$	All of C is asserted with role r to A
$\forall$	Universal restriction	$\forall r.A$	all r-successor exists in A
$\neg$	Negation	$\neg A$	not A
$\models$	Models	$O \models Ax$	Ontology O models axiom Ax

Table 2.2: Description logic notation

OWL Full contains OWL Lite, OWL DL and all features of RDF Schema. It has been shown to be undecidable [110] offering little tool support. OWL Full is the only variant allowing reification, i.e. statements about statements.

The concepts described in the later chapters rely on the conventional description logic notation [9] consisting of various symbols having a distinct semantic. An overview on these symbols is given in Table 2.2.

2.1.2.3 SWRL

Rule languages have their roots in the field of artificial intelligence. They are used to express logical consequences from a set of conditions. Originating from *Datalog*, which is syntactical subset of *Prolog*, rules are commonly understood in the sense of Horn logic [130]. A "Horn Clause" is a clause, i.e. disjunction of literals, with at most one positive literal:

$$\neg p \vee \neg q \vee \ldots \vee \neg y \vee z \tag{2.1}$$

A logically equivalent notation with improved readability is the following:

$$z \leftarrow (p \wedge q \wedge \ldots \wedge y) \tag{2.2}$$

While z denotes the head literal, i.e. consequence, $p \ldots y$ are the body literals, i.e. antecedent, of the rule. Each literal is of the form $p(t_1,\ldots,t_n)$ where p is a predicate of arity $n \geq 0$. The head of the rule holds, when all conditions of the body hold. Other than description logics, rules enable the modeling of triangle relations, thus are a suitable way to define policies [129, 82]. Rules, however, also have limitations being limited to the universal quantifier.

The *Semantic Web Rule Language* (SWRL), was submitted to the W3C in 2004[6] as a proposal to combine the Datalog rule language *RuleML* with *OWL DL*. Thus extending the variable free $\mathcal{SHOIN}(\mathbf{D})$ description logic of OWL DL with variable based rules [82]. This extends the expressiveness of OWL, making it possible to define triangle relations between concepts [93].

A SWRL rule consists of one head literal and one or more body literals. The literal can be a $\mathcal{SHOIN}(\mathbf{D})$ class, object property, data type property, data type respectively or a built-in function [130]. Let a and $b_1,\ldots,b_n$ be SWRL literals. A SWRL rule r is an expression of the form:

$$a \leftarrow b_1,\ldots,b_n \tag{2.3}$$

The SWRL built-in predicates enable simple mathematical calculations. For example, the predicate $multiply(z,x,y)$ will be true, when the equation $z = x \cdot y$ holds. Thus, the reasoner will assign the variable z with the value of $x \cdot y$. Further built-ins for addition, subtraction, etc are available and can be extended by writing own built-in predicates. The full expressiveness of SWRL comes at the price of decidability. However, a restriction to the set of DL-safe rules avoids this problem. When working with known instances only, reasoning engines support this by adding the necessary restrictions automatically [93]. SWRL has an XML based syntax to serialize SWRL rules contained in OWL ontologies.

2.1.2.4 SPARQL

The *Protocol and RDF Query Language* SPARQL is a query language for information retrieval from RDF and OWL based knowledge bases. A *query* in general is a read request to a set of formalized knowledge, i.e. typically a database or an ontology, explicitly restricting the answer set by defining constraints on the entities and their properties returned. SPARQL has an SQL-like query syntax, offering data access to both locally stored ontologies and via remote protocols like HTTP. The language syntax is shown in Listing 2.1.

2.1.2.5 OWL Reasoning

A reasoning engine, or reasoner, refers to a software which is able to infer logical consequences from a set of asserted facts or axioms comprised in an ontology. With

[6] `http://www.w3.org/Submission/SWRL/`

Listing 2.1: SPARQL syntax

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

the pervasion of OWL as standardized ontology language, different algorithms and implementations for reasoning on OWL ontologies evolved. A complete listing of all available OWL reasoners is omitted, as it has no benefit for the reader. Following is a brief description of three prominent engines having different characteristics.

Pellet[7] is an open source OWL DL reasoning engine. It offers a Java API as well as a DIG interface, enabling it to be used by modeling tools such as Protégé[8]. According to a benchmarking study, Pellet is found to be most efficient in classifying large ontologies among the four tested reasoners (Pellet, KAON2, Fact++, RacerPro). The benchmark is based on a framework for benchmarking OWL DL inference machines on large real-life ontologies [146]. The underlying algorithm is based on the tableau method. Pellet provides some support for DL-safe SWRL rules. In a performance evaluation provided by [88], Pellet was favorable to KAON2 for small cases with few individuals, but was indicating an inferior scalability. Pellet support the description logic $\mathcal{SROIQ}(\mathbf{D})$.

Fact++[9] is a C++ based reasoner, thus yielding slight performance advantages over Java implementations. Fact++ offers no rule support. Like Pellet, Fact++ is based on the tableau algorithm.

The Java based framework *KAON2*[10] uses a novel approach. Unlike most reasoning engines that are based on the tableau algorithm, the algorithm in KAON2 reduces a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program, which allows to use well-known deductive database techniques to optimize performance. Other than Pellet and Fact++ which are optimized for T-Box classification, the architecture of KAON2 was optimized for fast A-Box querying [112]. "KAON2 is the best system w.r.t the overall performance to load and respond, and shows favorable scalability" in a conjunctive query benchmark study [29]. Additionally, KAON2 supports authoring and reasoning over DL-safe SWRL rules.

Another novel approach with focus on scalable T-Box reasoning is contributed by the OWL reasoner *HermiT*.[11] Based on a hypertableau calculus, known from resolution reasoning, T-Box reasoning performance is improved by addressing two common problems: Tableau complexity is reduced through improved guessing, model size is reduced through a strategy called "anywhere blocking" [115]. Empirical evaluations show a robust classification performance and significant performance advantages over other reasoners across different real-world ontologies [139, 74].

[7]http://clarkparsia.com/pellet/
[8]http://protege.stanford.edu/
[9]http://owl.man.ac.uk/factplusplus/
[10]http://kaon2.semanticweb.org/
[11]http://www.hermit-reasoner.com/

2.1.3 Ontology Persistence

Most commonly, OWL ontologies are persisted by serialization into local files. Serialization can be based on either OWL/XML or RDF/XML notation. For authoring and querying ontologies, ontology files are deserialized and used as in-memory object, having the benefit of fast read, write and query times. However, this common approach has two major drawbacks: (1) A decentralized and file based persistence requires an immense synchronization effort in shared environments with many people authoring and querying ontologies simultaneously, and (2) with increasing size of the ontology, in-memory solutions become very costly, if not impossible, since available random-access memory is a major restriction on most hardware systems.

A database persistence layers is capable of addressing both issues, offering two key benefits to the users of ontologies: an efficient and centralized data store and virtually no limitations to data size. These advantages are traded against slower read, write and query performance [68]. Yet, scalable in-memory based database management systems currently are emerging, strongly reducing this disadvantage [140].

2.1.3.1 Jena SDB

Jena is a open source semantic framework using a triplet-based approach to author, query and store ontologies [35]. For this, Jena uses a graph structure supporting both RDF(S) and OWL ontologies. *Jena SBD*[12] is a database back end for Jena graph structures, developed for storing RDF data into relational databases. Queries to Jena knowledge bases are done by means of SPARQL (see Section 2.1.2.4). SDB also supports OWL, however, as every statement is stored as triplet into one large database table, querying OWL knowledge from SDB requires many self-joins. SDB usage for OWL ontologies thus is suboptimal [68].

Jena SDB supports the following database management systems:

- Apache Derby
- H2
- HSQLDB
- IBM DB2 9
- Microsoft SQL Server 2005
- MySQL
- Oracle 10g
- PostgreSQL

[12] `http://jena.apache.org/documentation/sdb/index.html`

2.1.3.2 Oracle Database 11g

Oracle Database 11g[13] is a commercial database management system by Oracle that offers a native support for both RDF(S) and OWL. As in Jena SDB, the underlying approach is based on a triplet store, i.e. every information is persisted as subject-predicate-object triplet into one large database table. For querying the knowledge base, both SQL and SPARQL can be used. The solution also comprises a version control system for ontologies. Apart from using SQL or SPARQL, it is possible to integrate the Jena API allowing to access the database from within a Jena based application.

2.1.3.3 OWLDB

OWLDB (aka Mnemosyne) is an open-source software specifically developed as native database persistence layer for OWL ontologies and was developed as part of the publicly funded Theseus[14] project. With OWL being an object abstraction, a triplet based persistence using RDF(S) syntax is not optimal [68]. OWLDB therefore uses an object relational projection of the ontology onto a database scheme, relating to the object model of the OWL API [13]. The projection is implemented by means of Hibernate, an open source Java persistence framework. Hibernate's database abstraction layer can connect to different database management systems like MySQL, Oracle or DB2, thus yielding a high compatibility for OWLDB. The OWL API reference implementation relies on an in-memory processing of file based ontologies. OWL DB adds a database persistence layer to the OWL API, having its focus on direct manipulations on the database layer, rather than loading the complete ontology into memory. In contrast to the in-memory approach, this yields the benefit of the ability to work with very large ontologies. In-memory approaches typically fail when the ontology is larger than the available memory space. Performance evaluations have proven this benefit with very large ontologies [68]. The architecture of OWLDB is depicted in Figure 2.2.[15]

2.2 Services

The goal of this section is to provide a thorough introduction to the service concept itself, conceptual classification models as well as related paradigms and technologies.

[13]`http://www.oracle.com/technetwork/database/options/semantic-tech/index.html`

[14]`http://www.theseus-programm.de/`

[15]Source `http://www.theseus.joint-research.org/assets/Weitere-Informationen/PosterMnemosynePythia.pdf`

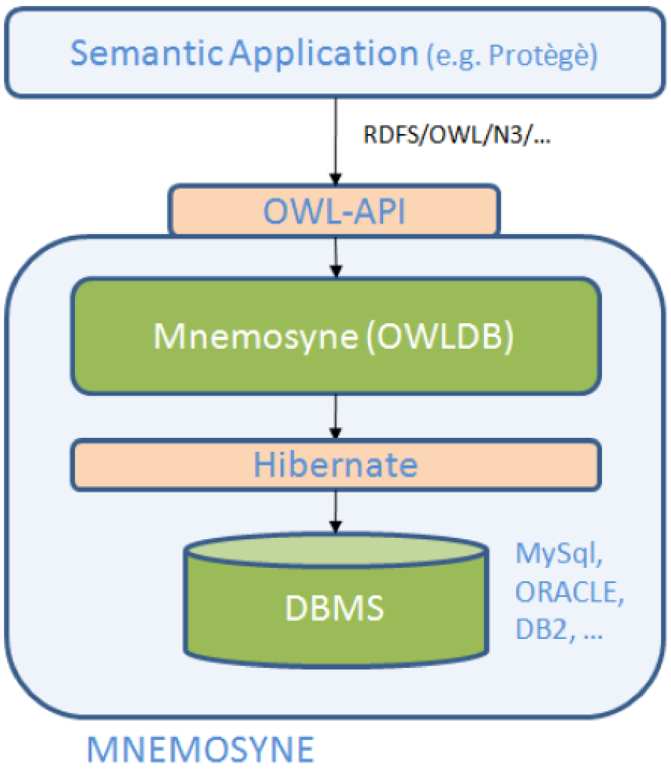


Figure 2.2: Architecture of OWLDB

2.2.1 Tangibles, Intangibles and Service Definitions

A common paradigm in economics is to distinguish between goods and services. The terms good, tangible good, intangible good and service, however, are not clearly defined in the literature. Often, these terms are used with different meanings, depending on the context or field of research. Therefore it is important to provide a common understanding which is used throughout this thesis.

2.2.1.1 Tangibles and Intangible Goods

In economics, a *good* is defined as an entity, over which ownership rights can be agreed on [71]. In contrast to a *bad*, a good is associated with a positive utility to its owner. Ownership juridically grants exclusive rights for using, altering or disposing a good and to prevent others from doing so. As ownership rights can be traded, a good is also tradable. Both goods and their owners resemble separate entities, thus production and trading are independent activities, which can be carried out separately and by different actors. The events of a good's production, distribution and consumption can be spread widely apart.

In general most goods are material, i.e. tangible. Intangible entities having the economic characteristics of a good also exist [71]. Examples are music, software, construction plans, news or other information and many more. Intangible goods have no physical dimension, but typically are stored on physical media, many being electronically transferable, thus easier to duplicate than tangible goods.

2.2.1.2 Service Definitions

Other than goods, the production and consumption of services cannot occur at separate times. According to Hill [71], two essential characteristics define a service: (1) "services cannot be produced without the agreement, co-operation and possibly active participation of the consuming unit(s)" and (2) "the outputs produced are not separate entities that exist independently of the producers or consumer".

Different definitions of services exist in various contexts. A very simple and anonymous definition states that "services are anything of economic value that cannot be dropped on your foot". In [49] two other aspects are mentioned: (1) Services are generally perishable in time, i.e. you cannot produce them to stock. At the same time, (2) service delivery usually involves a close relationship between provider and consumer. In any case, the consumer finds himself in the role of a co-producer. E.g. the service of a phone call cannot be delivered (and certainly not charged), without the telephone customer picking up the telephone, dialing the number of somebody else and eventually holding a conversation. By looking at this example, a fourth characteristic of a service becomes evident: It is produced and consumed at the same time. In general, a service is an activity or performance where the result is a change of condition of some person or good. Underlying this change of state is an agreement of the good's owner and the economic unit providing the service.

Definition 2.2 [SERVICE]. *A service is an activity which an economic unit A (service provider) performs for another economic unit B (service consumer) that results in a change of state or condition of an economic unit C whereas the output of that activity cannot circulate in the economy independently of economic unit C.*[16]

In computer science, a service (often called *software service*) can be characterized by the following qualities: It has a self describing interface, it is platform independent, based on standards and thus composable with other services. The economic unit undergoing the change of condition in a software service is intangible and electronic by nature, i.e. some kind of data. Any software service thus offers the capability to manipulate or provide data of a certain expected type. Despite the intangibility of the affected economic unit, the consequences following the change in condition can be of tangible nature. An example could be an electronic order service, which triggers the delivery of a tangible good. Within this thesis, we rely on the following definition.

Definition 2.3 [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 well-defined programmatic service interface using standardized protocols and is consistent with the provider's constraints and conditions.*[17]

[16]Definition based on [70, 51, 26].

[17]Adapted from [93].

The term software service is used in both intra-enterprise and inter-organizational contexts. Focusing specifically on the inter-organizational aspect, the Web aspect comes into play. A Web service is a specific software service exposing its functionality to the Internet using Web technologies like HTTP[18] as transport protocol and XML[19] or JSON[20] as data formats. Initially, the emergence and rapid spread of Web services brought into existence complex protocols like SOAP[21] and the WS-* stack[22] built on top of it. Recently, the trend has led to an increasing adoption of lightweight technology such as RESTful Web services [95]. Representational State Transfer (REST) [48] is a paradigm for Web applications characterized by to main properties: RESTful services are (1) stateless, i.e. any call of the Web service is independent from previous calls, and (2) resource oriented, i.e. all entities of interest are considered as a resource which is accessible by a unique identifier. This unique identifier is a URI[23] accessed using the HTTP methods GET, POST, PUT, PATCH, DELETE and OPTIONS. Despite its origin being in the Internet, the same Web technologies have also succeeded into many enterprise intranets.

Definition 2.4 [Web Service]. *A Web service is a software system identified by a URI [RFC 2396], 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.*[24]

2.2.2 Cloud Services

With compute power, storage and bandwidth more and more becoming a commodity like electricity or tap water, all being publicly available in large data centers to end consumers and businesses alike, Cloud computing as new paradigm has gained a lot of attention in the past few years. In the context of services, the idea behind Cloud computing can be stated as follows: Whoever is willing to provide any kind of software service should be able to focus on the unique value proposition of the service to be developed, rather than on its deployment or on the necessary management of the underlying computer resources. Cloud computing is a vague topic with many different opinion and definitions existing in literature. However there are some key characteristics[25], which are widely accepted:

On-demand self-service Automatic on-demand provisioning of computing capabilities (server time, storage, network bandwidth, etc.) as unilateral process without human interaction with the provider of the Cloud service.

[18]Hypertext Transfer Protocol
[19]Extensible Markup Language
[20]JavaScript Object Notation
[21]Simple Object Access Protocol
[22]Web service standards defined by the W3C (WSDL, WS-Security, WS-Agreement, WS-Policy, ...).
[23]Uniform Resource Identifier
[24]Adopted from [152].
[25]Defined by the National Institute of Standards and Technology, an institution of the U.S. Department of Commerce, in [107].

Broad network access Standardized protocols allow access to the provisioned resources for many thick or thin client platforms (workstations, mobile phones, tablets, etc.).

Resource pooling Multi-tenant model that allows the provider to offer pooled resources in a standardized manner which is independent from the physical location of the hardware resources. The user has information and control on the physical location on a high abstraction level (e.g. country, region, etc.). Virtual resources are assigned dynamically based on the current user demand, i.e. the assignment of physical resources to consumers are short-lived and only valid for the time of use. Typical resources include (but are not limited to) processing, memory, storage and network bandwidth.

Rapid elasticity Enablement of highly scalable architectures by allowing an elastic provisioning and release of resources. Scalability is possible both inward and outward and in a rapid and automatic manner according to the current load. For applications with small resource requirements compared to the size of the data center, scalability virtually seems unrestricted in quantity – possible through over-provisioning of resources and reciprocal compensation of load curves.

Measured service Metering capabilities allowing a detailed monitoring of resource usage on an individual consumer level. The obtained data is used for resource optimization, QoS management, consumer transparency and accounting allowing a detailed pay-per-use billing.

There exist different service models which offer the Cloud computing resources to potential Cloud users. The least restrictive model is called *Infrastructure-as-a-Service* (IaaS). The capability is provided as either processing, storage, network or other fundamental computing resource. The consumer typically has complete control over the resource on a virtual level. For a compute service this implies control over the operating system, file system and deployed applications, but no access to the underlying Cloud hardware. Root access to the operating system makes IaaS the most versatile service offer.

Platform-as-a-Service (PaaS) offers constrain the consumer to a certain development layer (programming languages, libraries, services, tools, etc.). Full control is given over the deployed application and some configuration options of the underlying framework. However, the possibility to manage the underlying operating system, network access or Cloud infrastructure is not given.

The most restrictive and specialized offer is *Software-as-a-Service* (SaaS). SaaS offers are closely related to the concept of an Application Service Provider (ASP), where a predefined end-user service is offered – tailored to be integrated directly into business processes of many different customers. SaaS directly offers businesses functionality without requiring developments on the consumer side, yet offers few possibilities for configuration and customization.

Within this thesis, the term *Cloud service* is used as specialization of services which originate from a Cloud infrastructure. In our sense, a Cloud service can be understood as any software service which is based on a Cloud infrastructure independent of the above mentioned service models. Typically it exposes some or all of the main Cloud characteristics. For the scope of this work, a Cloud service is therefore defined as follows.

Definition 2.5 [CLOUD SERVICE]. *A Cloud service is any kind of software or Web service which is provided based on a Cloud computing infrastructure, typically incorporating (but not limited to) the characteristics broad network access, rapid elasticity and measured service.*

Within this thesis we further rely on the notion of a *service resource*, which is an abstraction of any technical resource which can be combined with other service resources in order to provide a certain business functionality.

Definition 2.6 [SERVICE RESOURCE]. *A service resource is defined as abstraction of any technical resource (or any combination of technical resources) which can be combined with other service resources in order to provide a certain business functionality, but does not fulfill an entire business functionality by itself.*

2.2.3 Service-Oriented Architectures

As opposed to traditional software development, service-oriented architectures promote modularization and goal-oriented composition in order to enable distributed applications based on smaller reusable service entities [125].

Definition 2.7 [SERVICE-ORIENTED ARCHITECTURE]. *A service-oriented architecture (SOA) is a software design principle promoting reusable, interoperable and discoverable software services in order to provide distributed applications in a loosely-coupled manner.*

Service-oriented architectures thus foster the creation of reusable services which are optimized for a very specific purpose and can be included in different complex application scenarios or business processes. Following the key characteristics are described in more detail.

Reusability Requires that the service is self-contained in a way that it can be used for more than one business case. I.e. it offers its capabilities such that they can be used for different purposes or contexts. Applications or business processes thus can be composed by combining several reusable service components.

Loose-coupling In a service-oriented architecture, atomic service components have few or no dependencies to other components. Loose coupling can be achieved by introducing a message based bus system and clearly defined interfaces. Interfaces serve as business facade, encapsulating the logic from the actual implementation.

Interoperability Interoperability is capability of integrating two or more services with little or no adaption necessary. Communication between interoperable services is achieved by standardized, platform independent communication protocols (SOAP, CORBA, REST, etc.).

Discoverability A prerequisite for reusing a service is its discoverability. I.e. the service and its capabilities can be discovered either by the software architect or an automatic mechanism. Discoverability requires a repository system containing all available services and their machine-readable service description.

The introduction of service-oriented architectures can be seen as the fundamental driver for service mashups, Service Value Networks and Cloud computing. For completeness one should mention that the term SOA comprises more than the above mentioned requirements. A SOA additionally contains the elements for service discovery, contracting and communication.

2.2.4 Mass Customizing Services

Mass customization is defined as the ability to offer individually designed products and services to every customer on a large scale basis [42]. High process agility, flexibility and integration are prerequisites for mass customization in the manufacturing world.

The concept of mass customization is based on the notion of "economies of scope", enabling providers to offer product variety and customization through flexibility and quick responsiveness through advances in manufacturing and information technology, as well as new management methods, like lean production [89]. Mass customization allows companies to address the long tail [3], i.e. selling less of more by producing enough variety in products and/or services so that almost everyone finds what he or she wants at a reasonable price [131].

Thus, the two main characteristics are individualization and scalability. In the context of Cloud services, this implies an automated and flexible offer creation process, i.e. consumer and provider have to agree on both functional aspects and non-functional facets like QoS and price following a standardized, simple and quick process. After the offer creation process, service delivery or deployment consequently has to follow the same characteristics. Based on these characteristics, a mass customized Cloud service will be defined in this thesis as follows.

Definition 2.8 [MASS CUSTOMIZED CLOUD SERVICE]. *A Mass Customized Cloud Service is a Cloud computing service, composed by a set of software and/or Cloud resources, according to the abstract functional requirements and non-functional preferences of an individual consumer request. The offer creation and service delivery has to occur in a mass customization fashion, i.e. the negotiation on functionality, quality and price and the deployment of the Cloud service have to be simple, automated and possible on a large scale basis.*

2.2.5 Service Value Networks

Service-oriented architectures – especially service encapsulation – have fostered the emergence of highly specialized Web services. Due to standardized protocols, many of them can be combined to form complex services that support the individual

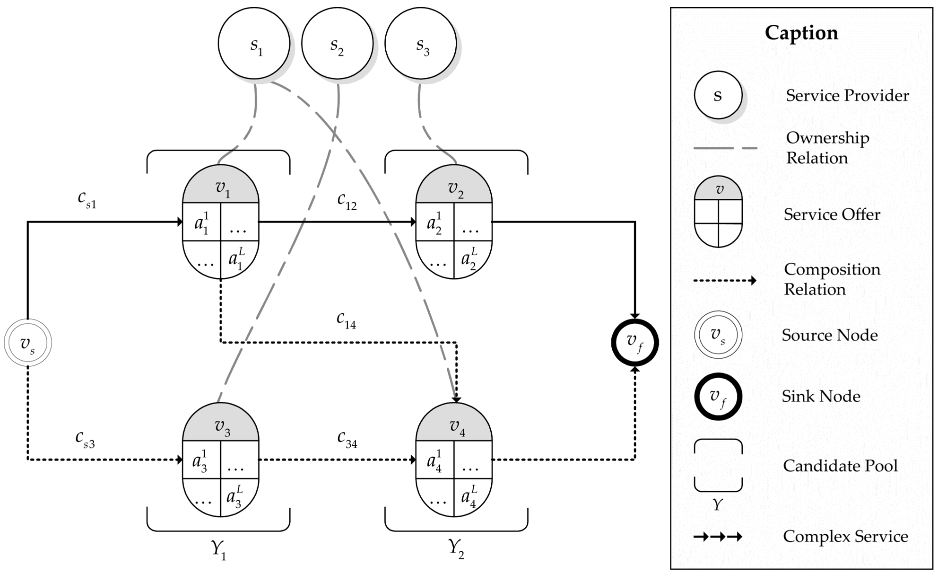


Figure 2.3: Graphical representation of a formalized SVN [23]

requirements of complex business processes, addressing the long tail [3] by enabling service mashups that are tailored to very individual consumer needs. In this trend, current literature envisions the formation of so called Service Value Networks (SVN) [11, 23].

According to Blau et al. [23], "Service Value Networks are Smart Business Networks, which provide business value through the agile and market-based composition of complex services from a steady, but open pool of complementary as well as substitutive standardized service modules by the use of ubiquitously accessible information technology".

A business network hereby refers to the most general form of an economically motivated cooperation among legal entities [73]. A smart business network constitutes a new type of business network with emphasis on the use of ICT to foster network interaction. The smartness of the business network thereby relates to the increased effectiveness and the comparative advantage achieved by the use of information technology.

Within this work, we concentrate on single providers (or intermediaries) offering Cloud services in a mass customization fashion. Therefore, we abstract from the definition above, which emphasizes the importance of composing services from different providers and a market-based coordination. However, the concept of individually composing complex services from atomic service resources according to some abstract consumer request is quite similar to the concept of forming a consumer-specific SVN. Therefore an adapted version of the formal representation of a SVN as provided in [23] is presented later in this thesis and called *service configuration graph*. A graphical representation of a formally defined SVN taken from [23] is depicted in Figure 2.3. In this graph, vertices $v_1, \ldots, v_4$ represent atomic service instances with QoS attributes a_i^j. Any path from source v_s to sink v_f represents one feasible complex service.

2.3 Economic Foundations

This section introduces the reader to the economic foundations which are prerequisites for the economic contributions following in Part III. The section is divided into two subsections: first a brief overview on negotiations is given, followed by relevant concepts from the field of mechanism design, including the main economic desiderata, the revelation principle and two impossibility theorems having an impact on the work at hand.

2.3.1 Negotiations

Negotiations are subject of many different research areas. In economics, negotiations are considered by both contracting theory and mechanism design. The focus of these research areas lies mainly on the efficiency of outcomes and the value distribution among the negotiators in the context of trade. In social sciences, negotiation is studied as interaction between humans, thus, focusing on human behavior and interpersonal communication. In computer science and information system research, negotiation is seen from a technical perspective, mainly concentrating on standardized protocols for communication among computer systems.

As a consequence, many different definitions and meanings are around. A versatile definition attempting to be of generic nature is given by Bichler et al. [19]. A slightly adapted version shall be the definition used within the work at hand:

Definition 2.9 [NEGOTIATION]. *A negotiation is a (potentially iterative) communication and decision making process between two or more agents who (1) cannot achieve their objectives through unilateral actions, (2) exchange information comprising offers, counter-offers and arguments, (3) deal with interdependent tasks, and (4) search for a consensus which is a compromise decision.*

A negotiation does not necessarily have an agreement between all parties as an outcome, i.e. a negotiation can potentially end with a disagreement. Typically, a negotiation is an iterative process, i.e. negotiators can adapt their offers over time. The process of exchanging offers and counter-offers is also known as "negotiation dance" [133].

The process of communication and decision making of a negotiation can be structured, semi-structured or unstructured. In a structured implementation, the message space and potential choice or transfer functions are predefined and obligatory for all negotiation participants. In a semi-structured negotiation implementation, the rules themselves are subject to negotiation and can change over time. In an unstructured implementation no rules are given at all. Negotiations can be bilateral, with two negotiating agents, or multilateral, with more than two parties negotiating.

In the context of this thesis, structured bilateral negotiations are studied, the main interest being on the efficiency of outcomes and the value distribution among the negotiating parties. Negotiations are investigated from a mechanism design perspective.

2.3.2 Mechanism Design

The theory of mechanism design is concerned with the design of institutions, with its focus on how institutional rules affect the outcome of individuals interacting within these rules. Individuals are assumed to be self-interested, acting strategically and to be holding private information relevant to the decision at hand [78]. A *social choice* function maps the truthful information of all individuals on a desired optimal outcome. The classic utilitarian objective is to find the outcome that maximizes the sum over the individuals' utilities for a certain outcome [126]. The mechanism design problem is to find an implementation of a certain social choice function.

An example is the bargaining between a buyer and a seller. Within this negotiation scenario, the seller has an interest in achieving a high price and thus will act as if the item subject to the bargaining is very costly. The buyer's interest is diametrically opposite, making the buyer pretend to have a very low valuation for the item to keep the price down as low as possible. The efficient outcome of the bargaining is that the trade occurs, whenever the buyer's true valuation for the item exceeds than true valuation of the seller. Mechanism design investigates how to design a negotiation protocol inducing this efficient outcome.

Taking an auction as an example institution, the auction design has a significant impact on how participants behave and on the auction's final outcome. A sealed bid auction for example will induce different bidding behavior than an oral ascending bid auction.

2.3.2.1 Mechanism Design Objectives

In mechanism design, the design of institutions is subject to research with respect to different design objectives, often called *economic desiderata*. Such a desideratum (or property of a mechanism) could be, that individuals reveal their true type, i.e. they do not lie about their private information, or that no outside payment is required for the mechanism to function in practice. In the following, the classic and most frequently named desiderata in the context of mechanism design are presented.[26]

The first desideratum reflects on the individuals incentive to actually participate in a mechanism. Naturally, rational individuals will only participate, if they expect to gain positive utility from participating, or at least, do not expect to incur utility losses.

Desideratum 2.1 [INDIVIDUAL RATIONALITY]. *A requirement that each individual weakly prefers participation in a mechanism to not participating [78], i.e. individuals should not expect to incur utility losses from just participating in a mechanism.*

Assuming risk neutral rational individuals, this is equal to the claim that the expected utility from participating should be greater or equal to the expected utility

[26]Note that a mechanism does not reveal a certain property itself. It is known to have a certain property, if its social choice function reveals the property. In addition, the properties or desiderata hold for a given state only, i.e. for a given solution concept (dominant strategies, Bayesian Nash,...) and a domain of agent preferences, e.g. quasi-linear, monotonic, etc. [126].

from not participating in the interaction. From a practical perspective, i.e. if one wants to implement a mechanism in a real world setting, and if individuals are not forced to participate, this property can be very important.

Another meaningful property, especially seen from a practical perspective, is the desideratum called *budget balance*.

Desideratum 2.2 [BUDGET BALANCE]. *A mechanism is budget balanced if the sum of transfers to the individuals equals the sum of payments from the individuals, i.e. there are no net transfers out of or into the system and all payments made to the mechanism are redistributed among the individuals without subsidization from outside [126].*[27]

In other words, if an outside payments to the mechanism is required, a mechanism is not budget balanced. A mechanism, e.g. an implementation of a central exchange, most commonly is required to be balanced, as the authority running the exchange has no interest in subsidizing the mechanism, unless, however, the exchange is implemented for non-profit reasons.

A reasonable goal when designing a mechanism with individuals having private information relevant to the decision making is to incentivize participants in sharing their private information truthfully. No mechanism can find a decision from a global and objective perspective in an optimal manner, if the decision is made upon false information. *Incentive compatibility* ensures that individuals have no incentive in not revealing their true private information.

Desideratum 2.3 [INCENTIVE COMPATIBILITY]. *A mechanism is said to be incentive compatible, if when each individual expects that the other individuals will be honest and obedient to the rules of a mechanism, then no individual ever will do better (given the information available to him) by acting dishonestly or disobeying the rules of the mechanism [122]. A mechanism that is incentive compatible in dominant strategies is called* ***strategy proof****.*

That is, if revealing one's true type is an utility maximizing equilibrium, a mechanism is incentive compatible. Incentive compatibility is also important from a complexity perspective, as truth telling is a very simple strategy, as opposed to the complex dominant strategies which are possible in non-incentive compatible mechanisms. As stated above, (ex ante) incentive compatibility is a prerequisite for achieving an efficient outcome, however it does not guarantee efficiency. *Allocation efficiency* is the classical utilitarian objective requiring a mechanism to maximize the total utility over agents.

Desideratum 2.4 [ALLOCATION EFFICIENCY]. *A mechanism is allocative efficient if it maximizes the sum over all individual utilities [126].*

However, efficiency as a design goal is not always desirable, e.g. if the objective is to maximize a certain individuals utility, e.g. an auctioneer's revenue. In this case,

[27]A mechanism is weakly budget balanced, if there are no net payments from the mechanism to the the individuals, but there can be net payments from the participants to the mechanism.

the mechanism design problem is reformulated as an optimization problem which maximizes the utility of a particular individual.

In general, these goals can been interpreted *ex ante*, i.e. they hold in expectation given probability distributions on the private information of individuals, or *ex post*, i.e. they hold for all potential realizations, which is the stronger statement.

2.3.2.2 Revelation Principle

In general, mechanisms are differentiated into *direct*-revelation and *indirect*-revelation mechanisms. In a direct mechanism, individuals can only make direct claims about their preferences once. An indirect mechanism allows an iterative protocol, i.e. individuals can make adaptions to their claims, based on the feedback received from the mechanism.

The revelation principle was originally developed by Gibbard [52] and later extended by Green and Laffont [55], and Myerson [119, 120]. It states, that under quite weak conditions, any mechanism can be transformed into an equivalent incentive compatible direct mechanism such that it implements the identical social choice function [126]. The intuition behind is the following: Given a mechanism leads to a certain outcome in equilibrium given a set of strategy profiles for each participant, an incentive compatible direct mechanism can be designed, that simulates these strategies. The simulating mechanism thereby computes each individual's optimal strategy. As the optimal strategies can merely be computed based on truthfully reported preferences, truthfulness is a dominant strategy, leading to the same outcome. Consequently, the direct mechanism implements the same social choice function as the original mechanism.

A seminal implication follows the revelation principle: Given a social choice function can be implemented by any mechanism, it is sufficient to concentrate on incentive compatible direct-revelation mechanisms [78]. In other words, once having found an indirect mechanism, one can be certain that an equivalent direct mechanism exists. And if one can prove that no direct mechanism can exists, no indirect mechanism can implement the social choice function.

2.3.2.3 Impossibility Theorems

The revelation principle is a powerful theoretic achievement, that has led to a set of central impossibility results in classical mechanism design. The impossibility theorems essentially give proof which combination of economic desiderata no mechanism can ever achieve. The general approach behind these theorems is to first assume direct incentive compatible mechanisms, express other desired properties as mathematical conditions and then to show contradictions across the mathematical formulations of the desired properties [126].

Historically, Hurwicz [75] was one of the first to show the conflict between efficiency and strategy-proofness in a simple two agent model. The general impossibility theorem, which follows from Green and Laffont [55] as well as from later work by Hurwicz [76, 77], states that it is impossible to achieve an efficient and budget balanced social choice function in dominant strategies in a simple exchange economy. In a simple exchange economy there are buyers and sellers, selling the same good in single units.

Theorem 2.1 [Hurwicz Impossibility Theorem]. *In a simple exchange economy with quasi-linear preferences, implementing an allocative efficient, budget balanced, and strategy proof mechanism is impossible [126].*

An extension of the theorem to include Bayesian-Nash implementations is provided by Myerson and Satterthwaite [121], further strengthening the results of Hurwicz and Green & Laffont, if interim individual rationality is additionally required. The impossibility is provided by Myerson and Satterthwaite in the first theorem of their seminal work and demonstrated in a scenario where two agents are trading one good.

Theorem 2.2 [Myerson-Satterthwaite Impossibility Theorem]. *No Bayesian-Nash incentive compatible mechanism can ever be implemented such that it is allocative efficient, budget balanced and (interim) individual rational at the same time, even with quasi-linear utility functions.*

As a consequence of the two above mentioned impossibility theorems, in the search for an adequate mechanism, one can at most try to achieve two of the four aforementioned economic desiderata [126].

2.4 Optimization Techniques

This section provides an overview on optimization techniques which are referred to in Part III of the thesis. The focus hereby lies on linear programming and dynamic programming. In addition, a brief overview on the industry-leading LP solving engine CPLEX is given.

2.4.1 Linear Optimization

Linear optimization, also called linear programming (LP), is one of the most commonly used techniques of the field of operations research and a subdomain of mathematical programming. A good overall introduction to this field is given for example by Chvátal [38]. Simple modeling and efficient mathematical solution concepts are key characteristics. Its purpose of minimizing or maximizing a linear function subject to constraints, which have the form of linear (in)equations, make linear optimization a universal approach for solving a broad spectrum of different optimization problems. The practical relevance of linear optimization is mainly due to its capability of solving even large problems with a multitude of variables and constraints in a sufficient amount of time.

In general, a linear optimization problem consists of an objective function of the form

$$f(x) = f(x_1, \ldots, x_n) = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n \tag{2.4}$$

and a set of linear equations. The standard notation of a linear program has the following form:

$$\begin{array}{llllllllll}
\text{max} & c_1 \cdot x_1 & + & c_2 \cdot x_2 & + & \ldots & + & c_n \cdot x_n & & \\
\text{subject to} & a_{11} \cdot x_1 & + & a_12 \cdot x_2 & + & \ldots & + & a_1n \cdot x_n & \leq & b_1 \\
 & a_{21} \cdot x_1 & + & a_{22} \cdot x_2 & + & \ldots & + & a_{2n} \cdot x_n & \leq & b_2 \\
 & \vdots & & & & & & & & \vdots \\
 & a_{m1} \cdot x_1 & + & a_{m2} \cdot x_2 & + & \ldots & + & a_{mn} \cdot x_n & \leq & b_m \\
 & x_1, \ldots, x_n & \geq & 0 & & & & & &
\end{array}$$

It is equivalent to the shorter matrix notation

$$\begin{array}{rrcl}
\text{max} & c^T x & & \\
\text{subject to} & Ax & \leq & b \\
 & x & \geq & 0
\end{array}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. Hereby n denotes the number of decision variables, m the number of linear constraints, c the vector of objective function coefficients, x the vector of decision variables, b the vector of values of the constraint equations' right hand side and A the coefficient matrix. Each linear programming problem of generic form can be transformed into the standard form by a set of transformations.

2.4.2 Simplex Algorithm

Historically, the Simplex algorithm was first introduced in 1947 by Dantzig as a simple and efficient way for solving LP problems [123]. It is able to solve them exactly and in a finite number of iterations, with unsolvable or unrestricted LPs being recognized. Since then, many improvements have made the Simplex algorithms become the most important and most widely used optimization techniques.

The intuition behind the algorithm comes from a geometric perspective on the linear optimization problem. The solution space of any solvable LP in standard notation can be interpreted as a convex polyhedron. The algorithm hereby can start from an arbitrary corner of the polyhedron and "runs" along the edges until it has reached the optimal corner. In fact, the smartness of Simplex is the way of doing so, as this approach is not unstructured or random. In each step, only corners improving the objective value are selected, making each iteration an improvement over the status quo. The most advanced implementations of the Simplex algorithm can be found in solving engines like Gurobi[28] or the industry leading software CPLEX[29] by IBM (cf. Section 2.4.5).

2.4.3 Integer Programming

In the standard LP formulations, decision variables have the value domain of real numbers. In many practical problems, integer or binary values are a requirement, especially when dealing with indivisible goods or true/false decisions. Problems where one, many or all decision variables are integer variables are called *integer program* (IP). If integer and real value decision variables are mixed, we speak of a *mixed integer program* (MIP); in case of binary variables only, we have a *binary integer program (BIP)*.

While dealing with integers at first appears to make things simpler, the opposite is the case. Indeed, analogously to LP problems, the solution space of IP problems is also surrounded by a convex polyhedron. Nevertheless, the exact solution space cannot be obtained by the cut of a finite amount of half spaces, as in the LP problem, since it would contain also non-integer solution which by definition are invalid.

Different approaches exist to handle the increased complexity. Commonly used are the *branch & bound* approach [94, 43] and the *cutting plane* methods [54], which both are relying on LP-relaxation. I.e. the problem is first solved with real value decision problems. Afterward, further constraints ensuring integer values are added consecutively.

2.4.4 Dynamic Programming

Dynamic programming was originally introduced in the 1940s by Bellman [15]. A dynamic programming problem is an optimization problem which is characterized by the fact, that it can be broken down into slightly smaller overlapping subproblems and that is has the property of optimal substructure, following Bellman's principle of optimality. From the perspective of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. Dynamic programming can also be used to solve certain linear programming problems. However, it is not restricted to linear optimization problems, but has a vast field of applications, from string algorithms to the calculation of a Fibonacci sequence.

2.4.4.1 Bellman's Principle of Optimality

For using a dynamic programming approach for solving an optimization problem, the optimization problem has to be breakable into smaller subproblems which follow Bellman's principle of optimality [15]. The principle states, that an optimal policy has the property that whatever the initial state and initial decision are, the

[28]Gurobi Optimizer `http://www.gurobi.com/`

[29]IBM ILOG CPLEX `http://www.ibm.com/software/integration/optimization/linear-programming/`

remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

In other words, the optimal solution to a problem is constituted of optimal solutions to its subproblems only. Considering for example the problem of finding the shortest path from location A to location D. If the shortest path is A-B-C-D, then the Bellman property states that A-C, B-D, A-B, B-C and C-D as subproblems of A-D must also be shortest paths. Such problems can be solved by Dijkstra's algorithm, which is presented in the following.

2.4.4.2 Dijkstra's Algorithm

Dijkstra's algorithm, which was published by Dijkstra in 1959 [44], is a dynamic programming approach to solve the single-source shortest-path problem for graphs with non-negative edge path costs. The algorithm by Dijkstra has a runtime of $O(|V^2|)$, which can be further improved by optimized data structures, making Dijkstra's algorithm the most efficient single-source shortest-path algorithm available. As all dynamic programming approaches, it is based on Bellman's principle of optimality.

2.4.5 CPLEX

CPLEX[30] originally was developed by a company called *CPLEX Optimization Inc.*, further developed by the company *ILOG*, which later was bought by *IBM*. It was named after the Simplex algorithm and *C*, the programming language it is written in. Today, CPLEX comes with many additional methods for mathematical programming and constraint programming, also offering interfaces to other programming languages than C. CPLEX currently is the industry-leading standard for mathematical programming software and obtains frequent updates with many improvements from current research in the field of mathematical programming and constraint programming.

2.5 Summary

In this section, fundamental concepts and basic technologies which are prerequisites for the main parts of this thesis were presented. First, an overview on modern knowledge management including concepts like ontologies and description logics as well as technologies like OWL, SWRL, SPARQL and OWLDB was given. Following, we presented important service concepts, from very generic definitions to modern concepts like Service Value Networks. In addition, our understanding on a mass customized Cloud service was provided and given as a definition. For the negotiation part of this thesis, the required economic foundations were given, stemming mainly from the field of mechanism design. Lastly, optimization techniques

[30]The full product name is *IBM ILOG CPLEX Optimizer*.

as used later for finding optimal service configurations were introduced, including linear programming and dynamic programming approaches.

Chapter 3

Mass Customization of Cloud Services

This chapter is dedicated to the envisioned process of mass customizing Cloud services. Hereby, the reader shall be given a thorough understanding on the process of creating an offer for a Cloud service which is tailored to a specific consumer's functional requirements and non-functional preferences. In order to do so, Section 3.1 refines the scenario which was presented in the introduction of this thesis. General assumptions in the overall context of this work are given in Section 3.2. In Section 3.3, the offer creation process is described. The chapter concludes with a summary given in Section 3.4.

3.1 Scenario

Mass customization has become a major business factor in many industries. In the area of Cloud services, providers wishing to offer individualized services in a mass customization fashion are confronted with many different challenges. For the scope of this work, we consult a scenario from the perspective of a Cloud provider wishing to offer mass customized Cloud services (cf. Section 2.2.4) individualized to specific consumer requests. Such a consumer request is assumed to contain a certain business functionality as requirement, which is abstract in a sense that it claims a specific functionality needed, but not exactly by which means it should be provided, i.e. by using what kind of resources. It is assumed that the functionality cannot be attained by simple services like storage, compute power or other similar commodity services not requiring any consumer individualization. In this context, one can think of two different variants for providing such customized services:

- A *single service provider* – having both resources and knowledge to offer the required capabilities – provides the services by combining its own resources.
- An *intermediary* – having knowledge on the resources of other providers which can be combined to meet the required capabilities – provides the services by integrating all external resources and offering them as mass customized service to the consumer.

Both variants yield slightly different peculiarities, especially with respect to prices and QoS of resources. The individual characteristics of each variant are described in the following two sections.

3.1.1 Single Provider Variant

This variant is characterized by a single provider offering the mass customized service based on own resources, i.e. hard- and software are owned, operated and maintained by the single service provider. The prices for using a resource are *endogenous*, i.e. they depend on the current load factor, especially since hardware resources are associated with high upfront but low variable costs. In such a scenario, resource prices optimally are determined by revenue management approaches [132]. Nevertheless, if the required amount of compute resources necessary for fulfilling the request is small compared to the overall resource capacity, momentarily static prices can be assumed as the data center's load factor will not significantly be affected by the incoming request. As there are no third parties involved in the service delivery, the catalog which has to be offered to the consumer is assumed to be maintained by the single provider.

3.1.2 Intermediary Variant

In the intermediary variant, both resource prices and QoS service levels[1] of all resources offered are known to the intermediary, i.e. price and quality of each atomic service entity are *exogenous* and not influenced by the intermediary. For an intermediary with considerable small number of service requests compared to the overall resource capacity of all suppliers, this assumption seems reasonable.

Regarding the maintenance of the service catalog, the intermediary variant allows the following considerations: Selling atomic service resources through an intermediary offering mass customized Cloud services as an integrator can be an additional profitable distribution channel for providers selling single service resources. Thus, a collaborative construction and maintenance of the intermediary's service catalog as joint work with the suppliers appears reasonable, especially from an incentive point of view. It is realizable by means of a collaborative ontology evolution approach, which is, however, not within the scope of this thesis.

3.2 Assumptions

We concentrate, however, on the commonalities of both variants[2] and postulate the following assumptions for our scenario. By nature, both assumptions and solutions presented later in this work are closely intertwined.

[1]Described in Service Level Agreements (SLA) of the supplying providers.

[2]In the following, the term *provider* is interchangeable with the term *intermediary*, as all assumptions, solutions and evaluations are independent of the variant.

First, we reflect on the nature of the consumer request for a mass customized service, which is assumed to be two-fold comprising both functional and non-functional requirements.

Assumption 3.1 [SERVICE REQUEST]. *The request for the mass customized service consists of a functional part (what the service is capable of doing) and a non-functional part (how the service is capable of doing it).*

The functional requirements must contain a formal and unambiguous description of the desired functionalities. That is, provider and consumer must have a common language for describing a functionality, which can be of abstract nature.

Assumption 3.2 [FUNCTIONAL REQUIREMENTS]. *The functional part is an unambiguous machine-readable subset of the functionalities that are offered by the provider and can be of abstract nature.*

The consumer may have preferences regarding non-functional aspects of the desired service, i.e. the quality and the price which is delivered. Quality and price naturally are interdependent, as a higher quality commonly goes along with a higher willingness to pay. A utility function is assumed to be the vehicle for the consumer's preferences.

Assumption 3.3 [NON-FUNCTIONAL REQUIREMENTS]. *The non-functional part contains the requester's preferences on non-functional service properties like Quality-of-Service (QoS) or price in form of a utility function.*

As stated before, provider and consumer need to have a language for agreeing on the desired functionality. In the scenario at hand, we assume that the provider is offering a set of abstract functionalities in form of a catalog. Examples for such abstract business functionalities could be a content management system, a billing service or a data warehouse, just to give some arbitrary examples.

Assumption 3.4 [SERVICE CATALOG]. *All generic abstract functionalities offered by the provider are described in a machine-readable catalog for the consumer to choose from.*

For simplicity reasons, we assume the set of functionalities requested in the functional part must be satisfied completely by the configuration, otherwise there is no agreement between both parties. Hence, we neglect the case of offering configurations which are suboptimal with respect to the functional requirements.

Assumption 3.5 [FUNCTIONAL AGREEMENT]. *The functional part has to be met completely for an agreement to be reached, i.e. the requested functionality has to be provided on the given abstraction level. Otherwise, there will be no agreement and thus no service delivery at all.*

In most cases, given the functional requirements are of abstract nature, many different feasible configurations exist that match the functional requirements. That is many different combinations of Cloud resources – hard- and software, so to speak – can deliver the same abstract functionality. However, at different quality and cost levels. In the sketched scenario, we assume the provider to be in charge – as the party having the expert knowledge – of choosing the best configuration. As best we define the configuration yielding the highest profit, i.e. the biggest difference between the price that can be achieved and the costs. As the price is dependent on the utility function of the non-functional part of the request, this implies finding the configuration with the highest margin between monetarized consumer utility and costs.

Assumption 3.6 [CONFIGURATION SELECTION]. *Due to the abstract nature of the functional requirements, there can be more than one service configuration comprising different resources fulfilling the functional requirement. It is up to the provider to choose which configuration is offered. The resulting QoS will be reflected in the price for the service, based on the non-functional preferences stated by the consumer in her request.*

In order to reflect the quality in the consumer preferences, QoS properties of the service resources must be available, i.e. they need to be measurable and known to the provider. In addition, we assume that the overall Quality-of-Service can be computed by aggregating the QoS values of the underlying resources.

Assumption 3.7 [QUALITY-OF-SERVICE]. *The QoS properties (e.g. response time, availability, etc.) of the available service resources are measurable, known to the service provider and can be aggregated to obtain overall values.*

Lastly, rational utility maximizing individuals are assumed. That is altruistic or other irrational behavior is not reflected, neither on provider side, nor on consumer side.

Assumption 3.8 [UTILITY MAXIMIZATION]. *In both scenarios, consumer and provider act rational and pursue the objective of utility/profit maximization.*

As this thesis comprises contributions to different fields of research, these assumptions define the overall boundaries of the scenario. They are refined, where necessary, in the corresponding chapters.

3.3 Offer Creation Process

Independent of the variant of the sketched scenario, the offer creation phase for a mass customized Cloud service consists of three typical steps, request formulation, functional engineering and economic optimization. The three steps are depicted in Figure 3.1 and described in the following.

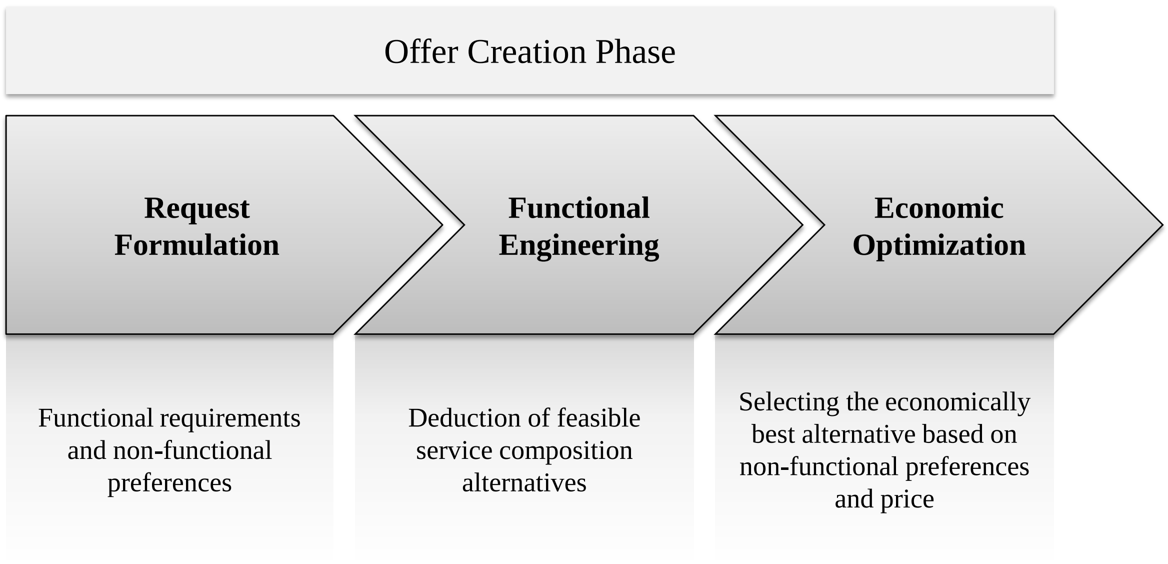

Offer Creation Phase
Request Formulation
Functional Engineering
Economic Optimization
Functional requirements and non-functional preferences
Deduction of feasible service composition alternatives
Selecting the economically best alternative based on non-functional preferences and price

Figure 3.1: Offer creation process for mass customized services

Request Formulation. Formulation of the request for a mass customized Cloud service containing both *functional* requirements, i.e. what is the required functionality of the desired service, and *non-functional* QoS preferences, i.e. how is the desired service supposed to deliver its functionality. Non-functional preferences typically are defined by some utility or scoring function over different values of technical QoS attributes like *response time*, *throughput* or *availability* and contains information on the willingness to pay for the service.

For a better understanding and to strengthen the relevance of this research, examples will be provided for each of the three steps. In these examples, a startup company (called consumer C) serves as an illustration of a potential consumer, requesting an individualized service. By means of this service the startup company wants to offer data mining services via the Web.

Example 3.1. *Consumer C requests a service consisting of a data mining engine, a Web server, an application server, and a customer relationship management component, defining the functional requirements. Concerning the non-functional preferences, the service should be at least 99,9% available, yet C is not willing to pay more for an availability above 99,99%. A service perfectly fulfilling these requirements is worth 500 monetary units per month.*

Functional Engineering. Engineering the service from the functional perspective. This requires some mechanism that is capable of deriving a set of feasible service configuration alternatives which fulfill the functional requirements of the service request. The result of this process is a graph structure, representing dependencies, iterative calls and compatibility.

Example 3.2. *For each of the different abstract functional requirements from Example 3.1, different software or SaaS solutions are available. Each of them has different dependencies that have to be fulfilled by an interoperable service resource. A sample graph potentially*

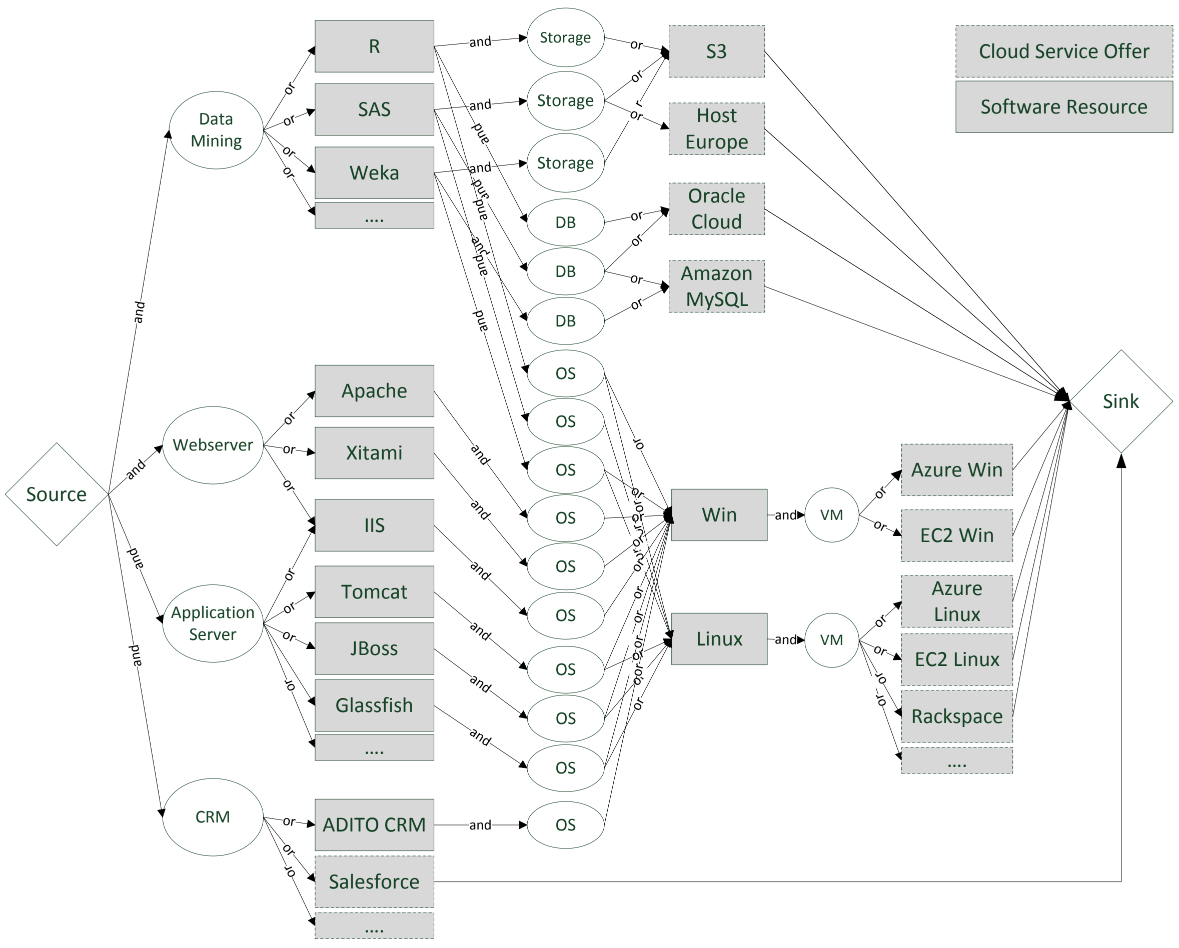


Figure 3.2: Illustrative example graph

resulting from the functional engineering step is depicted in in Figure 3.2.[3] *In this graph, the functional requirements are represented by the white colored vertices connected to the source. All white colored round vertices represent abstract requirements. The gray colored rectangular vertices depict potential service resources. Different resources are available for each functional requirement and bring along further dependencies. In the depicted graph, the data mining software SAS is assumed to require an additional storage service and a database management system in order to function. Within the graph, many different combination of service resources resemble feasible solutions to the service request of C. One of the many potential configurations in the example graph could be SAS on a Windows virtual machine instance of Amazon's EC2, combined with S3 for storage, Oracle as database service, Apache Web server and Tomcat application server running on a further virtual machine, and Salesforce as CRM service.*

Economic Optimization. Selecting the best alternative from the feasible set according to economic goals. Typically, this implies finding the best alternative with respect to consumer satisfaction and costs, represented by a goal function representing the profit obtained by a certain service agreement. The combinatorial complexity in such a graph can get very high, even for small problem instances.

[3]Note that the graph is illustrative, hence, it does not claim to be exhaustive.

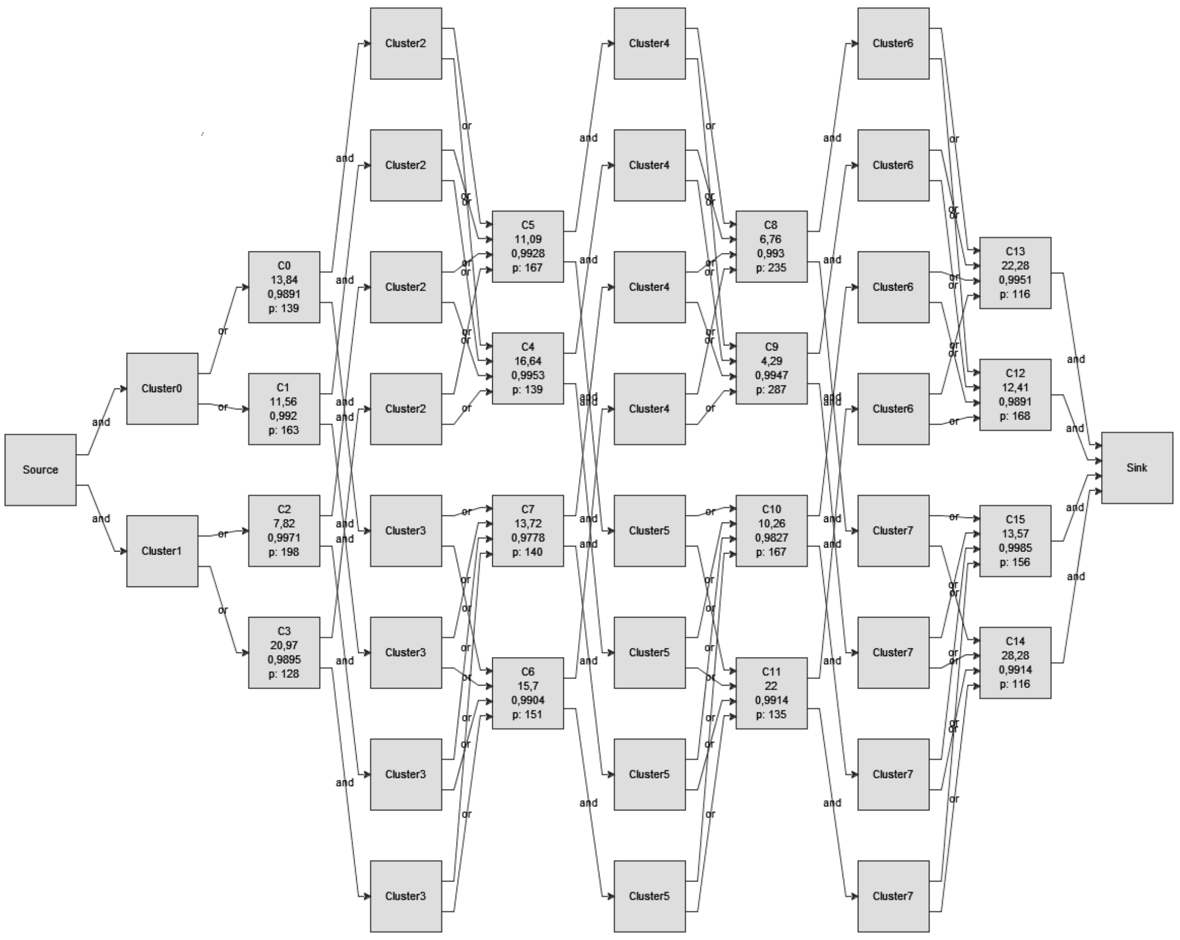


Figure 3.3: Formalized example graph

Example 3.3. *To illustrate the combinatorial complexity, Figure 3.3 shows a more formal representation of a graph as it can result from the functional engineering step, similar to Graph 3.2, without naming the concrete service instantiations. Analogously, different service resources (gray vertices named* $C1, C2, \ldots$*) are connected according to resolved dependencies (gray vertices named* $Cluster1, Cluster2, \ldots$*) and interoperability among resources. Two QoS attributes (response time in row two and availability in row three) are depicted in each service resource. Row four describes the resource price. Despite being a rather small graph instance, this illustrative graph structure has 114,688 possible combinations as solution space to the optimization problem. The challenge is to find the profit maximizing combination with respect to the willingness to pay and the quality requirements of consumer* $\mathcal{C}$*, as well as the costs accruing from the resource usage.*

Automating all three steps is a major challenge. It is the goal of this thesis to address this challenge. The first two steps, request formulation and functional engineering, are captured in Part II, while the economic optimization and the closely intertwined aspect of negotiation mechanisms follow in Part III of this thesis.

The interaction between a potential service consumer and the provider is depicted as sequence diagram in Figure 3.4 to illustrate the intertwining of all three steps. The sequence diagram serves as an additional outline of the following chapters, as we first describe the *Semantic Service Description Framework*, referred to as

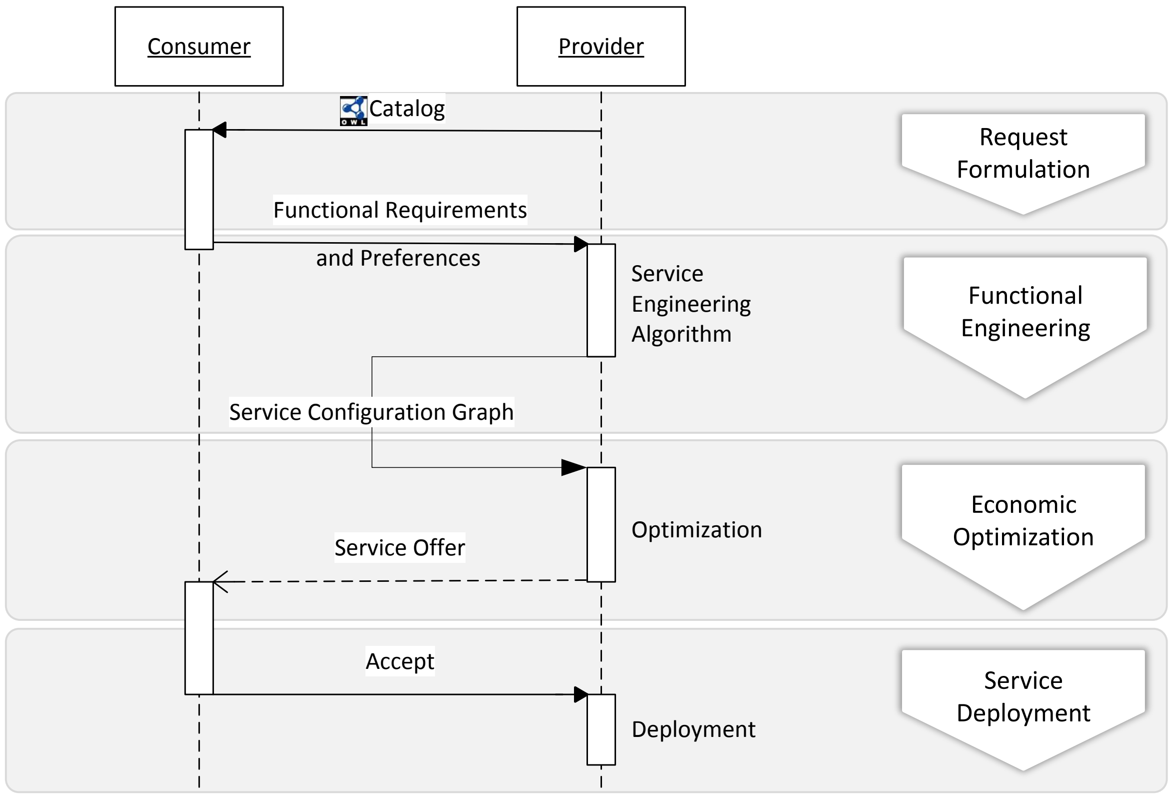


Figure 3.4: Sequence diagram for mass customizing a Cloud service

"Catalog", in Chapter 4. Closely related is the *Ontology Update Mechanism* following in Chapter 5. The *Service Engineering Algorithm* computing a graph over feasible service configurations is described thereafter in Chapter 6. The economic optimization, which is dependent on the established negotiation mechanism (cf. Chapter 7), is covered in Chapter 8.

3.4 Summary

In this chapter, the reader was introduced to mass customization of Cloud services and the envisioned scenario which sets the context for the work at hand. Within the scenario, two variants – one where a single service provider offers the customized services and one with an intermediary doing so – were described. Common to both of them, is the challenge of finding potential service configurations which meet the abstract functional consumer requirements and selecting the profit maximizing configuration, in the many cases were there is more than one feasible configuration.

Thereafter, we presented the broad assumptions surrounding the thesis, including aspects on the consumer request, the way offered functionality is described, the existence and knowledge on QoS properties and economic assumptions on the behavior of provider and consumer acting as rational and utility maximizing agents. Lastly, the envisioned offer creation process was presented and a sequence diagram on the interaction between provider and consumer in the sketched scenario was provided.

Part II

Technical Design, Implementation and Evaluation

Chapter 4

Semantic Service Description Framework

Large scale and automated engineering of Cloud services requires explicit knowledge on the available service resources. In this chapter, a framework for semantically describing service resources is presented. The framework is designed such that mass customized services comprising subsets of the available resources can be derived automatically according to abstract functional requirements given in a consumer request. The framework hence has two purposes: (1) It should be able to describe resources, their interdependencies and compatibilities and (2) at the same time report abstract functionalities, that can be built upon these resources for potential consumers to chose from.

The remainder of this chapter is structured as follows. Requirements on the designated framework are elaborated on in Section 4.1, followed by the corresponding related work in Section 4.2. The main contribution, an ontology framework for describing service resources, is introduced in Section 4.3. Evaluation and discussion come thereafter in Section 4.4. The chapter is concluded by a summary given in Section 4.5. The results of this chapter are largely based on [65].

4.1 Requirements

The requirements posed upon a service description framework partially can be derived from the overall scenario of this thesis and to some part are of generic nature for achieving a sustainable, reasonable and practical solution.

First of all, with many "bloated" ontology solutions for service description available, we seek a lightweight description approach with focus on the main objectives required for the engineering of mass customized Cloud services.

Requirement 4.1 [LIGHTWEIGHT DESCRIPTION APPROACH]. *The framework should be a lightweight description approach with strong focus on its core functionality – the derivation of customized Cloud services.*

The advantage of a lightweight solution is its focus and comprehensibility, while additional features can be easily extended or existing standard ontologies could be merged, if needed.

Its core objective is to provide a method for formalizing knowledge such that abstract functionalities and available service resources needed for providing these functionalities can be described in a machine-readable way. Thus, the framework should enable the construction of a taxonomy on both resources and abstract functionalities.

Requirement 4.2 [Taxonomy on Resources and Functionalities]. *The framework should enable the creation of a knowledge base comprising a taxonomy on service resources and functional requirements in as many abstraction levels as needed.*

Other than any approach for describing the I/O-based matching of Web services, the focus is on deriving service configurations based upon a selected set of abstract functionalities. This can be achieved by describing dependencies and compatibility among the abstract functionalities and the service resources. By resolving dependencies while simultaneously ensuring compatibility, required service resources can be resolved starting from the requested functionalities, thus deriving potential configurations.

Requirement 4.3 [Dependency and Compatibility Modeling]. *The framework should allow to model both dependency and compatibility relations among functionalities and service resources.*

Naturally, creating and maintaining such a knowledge base requires an immense modeling effort. If a certain relation is applicable for a whole group of similar resources, it should be modeled only once for the group rather than stating this relation for each instance of the group. We therefore require the framework to reduce the modeling overhead as much as possible by avoiding redundancies through modeling information on the highest abstraction level possible.

Requirement 4.4 [Reduced Modeling Overhead]. *The framework should foster the reduction of modeling overhead by allowing to state knowledge on relations between resources and/or functionalities on any abstraction level.*

As stated before, the designated framework has a clear research focus on enabling the derivation of service configurations by modeling interdependencies and compatibilities. However, a practical usage might require many more details or further information to be included. Therefore, the framework should be easily extensible.

Requirement 4.5 [Extensibility]. *The framework should be easily extensible to allow more details in the model or to include further information or more complex rules on compatibility.*

As the modeled knowledge might also be of interest to other institutions or partners, or knowledge stemming from external partner should be includable, the format of knowledge representation should be exchangeable across the border of institutions by being standard compliant.

Requirement 4.6 [STANDARD COMPLIANCE]. *The framework should be standard compliant to ensure the exchangeability across the border of institutions in order to import/export information from partners or other institutions.*

All of the above requirements built the surrounding structure for the design of semantic service description framework. In addition, these requirements help in distinguishing other existing solutions from the related literature, which follows in the succeeding section.

4.2 Related Work

The related work in this context can be found in the research areas of semantic Web service description and ontology-based configuration. Regarding the first, different standard proposals have been made in the past, including *OWL-S*, *WSMO*[1], *METEOR-S*[2] and *IRS-II*[3]. Along with other work in the context of Web service mashups, they are described in Section 4.2.1. Concerning the latter, different approaches from literature are described in Section 4.2.2.

4.2.1 Semantic Description of Web Services

OWL-S originated as part of the DAML program[4] and is an ontology for describing Web services based on OWL (cf. Section 2.1.2.1). The ontology comprises three parts: (1) the service profile telling a service-seeking or -matchmaking agent what the service does; (2) the service model which describes how to use the service by giving details on the semantic content of requests and the conditions under which particular outcomes will occur; (3) the service grounding specifying the access protocol for addressing the service [102]. The vocabulary defined by OWL-S may be used to provide semantic annotations of services, and automatic agents may process this information.

WSMO is a conceptual framework for describing Web services using formal semantics. It is targeted to facilitate automated discovery, combination and invocation of services on the Web. The following semantic Web design principles are (among others) incorporated in WSMO [134]. WSMO is *Web compliant*, relying on standards like URI, XML and other W3C recommendations. It is *ontology-based*, defining its own ontology model, claiming to be general enough to capture the other existing ontology languages. The recommended modeling language is WSML, although other

[1]Web Service Modeling Ontology
[2]Managing End-To-End OpeRations for Semantic Web Services
[3]Internet Reasoning Service
[4]DARPA Agent Markup Language – `http://www.daml.org/`

ontology languages are also supported. WSMO fosters *strict decoupling*, as resources are described isolated, i.e. the description of a resource does not include references to other resources that might be used or interacted with. In addition, WSMO emphasizes the importance of *mediation* between heterogeneous resources. Another design principle incorporated is the *ontological role separation*. WSMO differentiates between the high level preferences of a Web user and the actual technical desires upon an offered Web service. Lastly, it distinguishes *description* and *implementation*, offering a concise and sound description framework, independent from the actual implementation of a Web service. Yet, to ensure execution capabilities, it aims to be compliant with current technologies used for Web service execution.

WSMO consists of four top-level elements, representing the main concepts for Web service modeling: *Ontologies* are the foundation of WSMO, as they provide the terminology for all other WSMO elements. Yet, they are no pure terminology in a sense that they only focus on syntactical issues. In fact, they can be seen as an abstraction covering current ontology languages the creators of WSMO found to be most reasonable for describing Web services. *Web services* embody the computational entities, i.e. the interfaces and capabilities of a service using the terminology of ontologies. Hereby the choreography describes the way to communicate with a service, while the orchestration characterizes the interaction with other services. The service's capabilities can be defined based on pre- or post-conditions, assumptions, and effects. QoS attributes can also be defined as non-functional properties. *Goals* describe the service requester's requirements with respect to the requested functionality. These requirements can be defined, stating the desired capabilities and interfaces using the ontologies terminology. Goals can thus be seen as the users view on the Web service usage process. *Mediators* are responsible for ensuring interoperability between heterogeneous WSMO services. They are supposed to resolve incompatibilities on process, data or protocol level.

The focus of *METEOR-S* lies on workflow management techniques for transactional workflows in the context of Web services [100]. METEOR-S is a framework supporting the whole lifecycle of a semantic Web process using semantic technologies. This lifecycle consists of five main stages: development, annotation, discovery, composition and execution [1]. METEOR-S provides functionality to semantically annotate Web services, to describe data semantics, i.e. input, output and exception data formats, as well as functional semantics, i.e. the offered functionality and how the incorporated processes can be addressed. Additionally, one can specify QoS specifications in form of quadruples consisting of the name, a comparison operator, the value and the unit of the specification. For describing the offered processes, it relies on the BPEL4WS[5] standard. Service selection is a three step process, split up into service discovery, constraint analysis and optimization. The UDDI based discovery engine supports annotated WSDL, WSDL-S and OWL-S as description languages, annotated WSDL being the recommended one [1]. METEOR-S uses a constraint-based selection algorithm, where the searching agent can define business as well as QoS constraints on the desired service. Constraint fulfillment is first checked, then optimized using a linear programming algorithm. The optimization algorithm at the same time brings along a mechanism for ranking various service alternatives.

[5]Business Process Execution Language for Web Services, meanwhile called WS-BPEL as part of the WS-* stack.

The goal of IRS II is to support the publication, location, composition and execution of heterogeneous web services, augmented with semantic descriptions of their functionalities, providing a one-click publishing support for different software platforms [116].

In the context of Web service mashups, Blau et al. [25] propose an ontology-based tool for planning and pricing of service mash-ups. The tool can be used to compose complex Web services from a set of known atomic services, which are stored in a domain specific ontology. Afterward, the complex service can be validated based on axioms and rules in the ontology. A progression of this approach led to a tool called *remash!*, an ontology-based solution to model and rank blueprints for RESTful web service compositions [24].

4.2.2 Ontology-Based Configuration

Product configuration is concerned with assembling a set of customizable components to produce a good which satisfies both customers' needs and technical constraints. Ardito et al. [4], for example, present a case study for an Italian furniture company, introducing ontologies to model knowledge on available furniture components in order to offer customizable furniture via the Web.

Yang et al. [153] propose the use of ontologies for reusing configuration knowledge to support and enable a development of product configuration systems. A meta ontology as means for a product configuration meta model is introduced to define general terms and relations which are common to the configuration domain. Using this meta model, one can describe all product components and constraints among them belonging to a certain product. The ontology relies on OWL (cf. Section 2.1.2.1), while constraints are modeled by means of SWRL (cf. Section 2.1.2.3). By adding user requirements as additional constraints, an inference engine is capable of deriving all potential configurations which satisfy the constraints. As inference engine, the *Java Expert System Shell* (JESS) is chosen. Therefore, the OWL/SWRL model has to be transformed into JESS rules.

A further development of the approach is proposed by Dong in a joint work with Yang and Su [45]. This work introduces a slightly adapted version of the same approach, which allows the same concept to be used in the context of mobile communication services. As example, the complex task of configuring service packages for China Mobile is given.

4.2.3 Applicability

From the description above, it becomes obvious that it is common to all semantic Web service description approaches that their focus is on describing Web services for discovery and matchmaking purposes, rather than finding configuration alternatives for a given consumer request. None of them addresses the challenges and requirements which occur in the scenario of the work at hand, at least not in way, that the approach could be adapted to our needs. This is especially the case regarding Requirements 4.2 and 4.3.

The focus of Ardito's work is on the description of the case study and the potential arising from the usage of IT for supporting the process of mass customization within this case study. Yet, the paper is superficial regarding the actual model and the algorithms used, which are not presented.

The ontology-based configuration approach by Yang and Dong is much closer related. Using their proposed concept, potential product configurations in the traditional manufacturing world (drilling machine) and potential service packages in the field of a mobile communication service can be derived. Which configurations are feasible, depends on generic constraints stemming from the domain and additional constraints which embody the wishes formulated by the consumer. However, the distinct purpose of such an ontology is the configuration of a single traditional good (drilling machine) or the packages of a single service. This connotes fundamental differences to the Cloud service scenario of this thesis, where many different functionalities or services can be combined, rather than focusing on the configuration of one single functionality (drilling machine) or service (mobile communication). The approach therefore is not applicable to the scenario of mass customizing a large set of different Cloud service offers (cf. Requirement 4.3).

4.3 Ontology Framework

In this work, OWL (cf. Section 2.1.2.1) is chosen as the formal knowledge representation language for four reasons: (1) It has been established as the leading Semantic Web standard, (2) is widely spread thus offering a large set of modeling tools and inference engines, (3) is well documented and offers description logics (DL) expressiveness that fits well the anticipated description of services functionalities and resources at class-level. In addition, (4) OWL comes with a standardized, Web-compliant serialization which ensures interoperability over the borders of single institutions.

For stating more complex compatibility constraints that go beyond the DL expressiveness, the proposed approach makes use of the rule language SWRL (cf. Section 2.1.2.3) in combination with OWL, while the usage is restricted to DL-safe rules [114] as a decidable fragment supported by OWL reasoners like Pellet [142] or KAON2 [113].

The framework comprises three different types of ontology. A generic *service ontology* defines the fundamental concepts of the framework. The actual knowledge on the known software and Cloud resources is modeled in the *domain ontology* and will differ for the various Cloud service providers. While the service ontology is a static model, the domain ontology has to be dynamic, i.e. it will need constant updates on the current resource situation. The third ontology, the *result ontology*, is used to store potential configuration results of any algorithm capable of deriving all alternatives by resolving resource dependencies (cf. Chapter 6). Both domain and result ontology import the fundamental concepts defined in the service ontology.

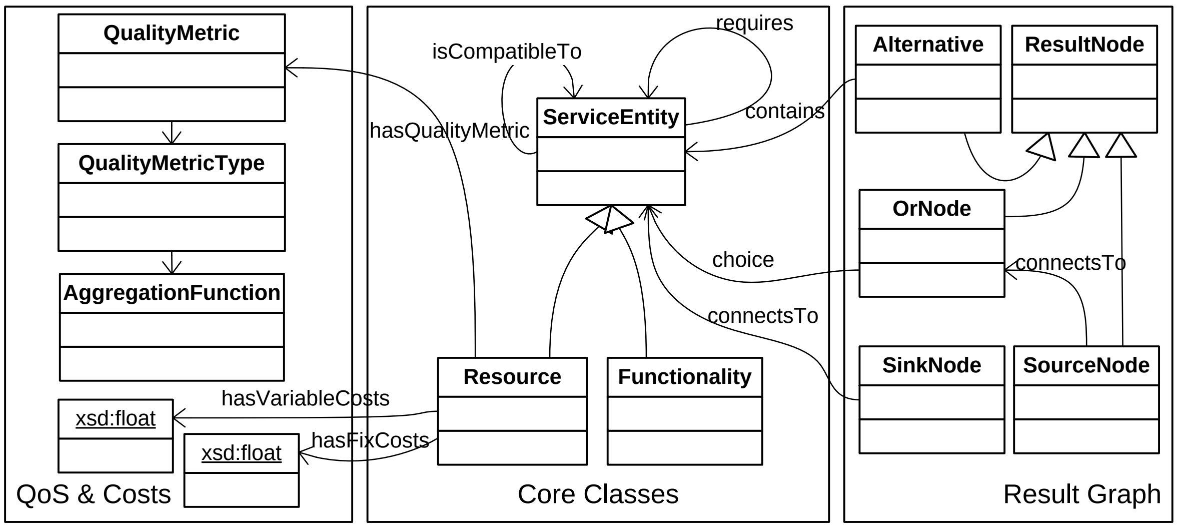


Figure 4.1: Service ontology

4.3.1 Service Ontology

The service ontology is partially depicted in Figure 4.1.[6] Basically, the service ontology provides concepts for three different aspects:

- Service resources along with dependencies and compatibility information, needed to derive all valid service configuration alternatives.
- QoS and cost meta information for evaluating these alternatives.
- Structure elements needed for an ontology representation of the resulting graph stucture.

The most fundamental concept for deriving all feasible deployment alternatives is the class *ServiceEntity* with its subclasses *Resource* and *Functionality*, along with the corresponding object properties *requires* and *isCompatibleTo*. The *requires* property is used to describe the functional dependency between two resource instances. In most cases, dependencies can and should be described in an abstract way at class-level. We can do this by including the dependency relation into the class axiom in conjunction with an object restriction in the form of an existential quantifier on the required class:

$$\begin{aligned} ResourceA &\sqsubseteq Resource \\ &\sqcap \exists requires.ResourceB \end{aligned}$$

Hereby we constitute that each resource of type *ResourceA* requires some resource of type *ResourceB*. The resources required by a functionality can be modeled analogously:

$$FunctionalityX \sqsubseteq Functionality$$

[6]For the reader's convenience we illustrate the ontology in UML notation where UML classes correspond to OWL concepts, UML associations to object or data properties, UML inheritance to subconcept relations and UML objects to OWL instances.

$$\begin{aligned} & \sqcap \ \exists requires.ResourceA \\ & \sqcap \ \exists requires.ResourceY \end{aligned}$$

As a more concrete example, we could state that every instance of the class Application requires at least some operating system:

$$\begin{aligned} Application \ \sqsubseteq \ & Resource \\ \sqcap \ & \exists requires.OS \end{aligned}$$

The compatibility can be asserted on instance level using the *isCompatibleTo* object property. That implies that there has to be one object relation between all possible combinations of interdependent resources. To reduce this modeling overhead, we propose the usage of SWRL rules, which allow us to state compatibility on class level:

(R1) $\quad isCompatibleTo(x,y) \leftarrow ResourceA(x), ResourceB(y)$

We can exploit the full expressiveness of DL-safe SWRL rules. E.g. to state that all versions of *MySQL* are compatible to all *Windows* versions except the mobile version *WindowsPhone*, we include the following rule:

(R2) $isCompatibleTo(x,y) \leftarrow MySQL(x), Windows(y), differentFrom(y,'WindowsPhone')$

For inclusion of QoS and cost information additional concepts are included. A resource can be associated with a quality metric, which itself is an association between a quality metric type and a certain value. Examples for quality metric types are response time, availability and other measurable QoS attributes. Each quality metric type must be associated with an aggregation function, which are the commonly used mathematical aggregation operators: *addition*, *subtraction*, *multiplication*, *min* and *max*. The usage of these concepts is illustrated in the following example:

$$\begin{aligned} (ResourceX, qm\_ResponseTime\_10) \ &: \ hasQualityMetric \\ (qm\_ResponseTime\_10, ResponseTime) \ &: \ hasQualityMetricType \\ (qm\_ResponseTime\_10, 10.0) \ &: \ hasQualityMetricValue \\ (ResponseTime, Addition) \ &: \ hasAggregationFunction \end{aligned}$$

The example states no other than *ResourceX* is associated with a *ResponseTime* of 10.0 and that *ResponseTime* is aggregated by means of an *Addition*. Regarding the costs, one distinguishes between non-recurring *fix costs* and recurring *variable costs*. The latter being more complex, as the total amount depends on another variable, which has to be defined in the context to serve as a multiplier. E.g. the overall usage fee for a cloud provider depends on the planned usage period for the service. Both values are modeled in form of datatype properties (*hasVariableCosts* and *hasFixCosts*) with *Resource* as domain and *xsd:float* as range.

The entire service ontology serialized as RDF/XML file can be found in Appendix A.1.

4.3.2 Domain and Result Ontology

As stated before, both domain and result ontology are not static, but undergo constant changes. The domain ontology must reflect the current state of nature regarding available service resources and their properties. A result ontology can store a subgraph of the domain ontology derived for a certain consumer request.

Domain Ontology. The domain ontology uses the concepts described in the service ontology to capture knowledge about service resources and functionalities of a certain service provider and domain of interest. This distinction between generic concepts and a domain model allows one to use the same technology – e.g. the service engineering algorithm of Chapter 6 – within different contexts or companies, just by loading a different domain ontology. In addition, knowledge can be combined by loading several domain ontologies, as long as the combination of domain models does not lead to inconsistencies, which could occur for example due to naming inconsistencies.

Through the concepts presented in this chapter, a service provider can create a domain ontology according to the resources available to him. In order to do so, dependencies and interoperability have to be modeled as presented in the preceding section by making use of the generic concepts defined in the service ontology. By doing so, the service engineering algorithm presented later in Chapter 6 can be used to derive feasible configurations. Providing a domain ontology describing the resource environment for an arbitrary service provider is not in the scope of this work. The additional scientific contribution of such an example domain ontology would be questionable due to its high specificity to the provider's circumstances.

Result Ontology. In the context of mass customization of Cloud services, it is one objective of the work at hand to allow for an automated derivation of service configuration alternatives. For this purpose, Chapter 6 presents an algorithm which is capable of doing so in combination with the framework presented in this chapter. In order to be capable of storing the resulting configuration also by means of an ontology – which is beneficial as it for example allows reasoning on the resulting information – additional concepts were added to the service ontology.

The result ontology, storing the configuration alternatives obtained from the service engineering algorithm, makes use of the concepts *SourceNode*, *SinkNode*, *OrNode* and *Alternative* which are defined in the service ontology (cf. Figure 4.1). The result information is a graph structure, therefore *SourceNode* and *SinkNode* are included as helper nodes to have a distinct starting and ending points in the graph. They correspond to the source and sink nodes in a network. The *OrNode* is introduced to capture the branching whenever there is more than one compatible resource instance that fulfills the dependency requirement.

4.4 Evaluation

The contribution of this chapter is evaluated qualitatively by comparing the presented approach to the requirements from Section 4.1. Further, a proof-of-concept implementation of the service engineering algorithm makes use of the presented framework, thus providing further evidence of its applicability. A description of the implementation follows in Section 6.5.

So far, we have presented requirements which were mainly derived from the overall scenario of this thesis, related work from the fields of Semantic Web service description and ontology-based configuration and the main contribution of this chapter, an ontology framework for describing Cloud service resources, their dependencies and compatibilities and meta information on QoS and costs. We now evaluate the contribution by confronting it with the requirements which were posed in the beginning.

Lightweight Description Approach. Having many different heavy ontologies and frameworks for service description available, an approach was sought that has a clear focus on the ability to support the derivation of Cloud service configuration alternatives. For this purpose, the description of dependencies, compatibilities and meta information of service resources is required.

The presented approach comprises a generic service ontology, which addresses exactly these issues. It does so in a simple and lightweight manner, as it only consists of a couple of generic concepts, object and datatype properties having descriptive names.

Taxonomy on Resources and Functionalities. The semantic service description framework mainly serves two purposes: offering selectable functionalities to the consumer and modeling information on service resources that are capable of providing these functionalities. In order to be able to formulate abstract functional requirements, a taxonomy from concrete to less concrete functionalities is a designated objective.

The solution enables the creation of such a taxonomy by using OWL as ontology language and providing subclasses of the concept *Functionality* in as many abstractions as desirable. An example could be the following taxonomy:

$$ApacheHTTPServer \sqsubseteq WebServer \sqsubseteq Functionality$$

The same principle applies for service resources. Considering for example Amazon's EC2 compute service as a resource providing compute power, we could state the following:

$$AmazonEC2Windows \sqsubseteq WindowsComputeService \sqsubseteq ComputeService \sqsubseteq Resource$$

Thus, the presented framework can be used to create large taxonomies on resources and functionalities by making use of the generic concepts and relations defined in the service ontology.

Dependency & Compatibility Modeling. For deriving feasible service configurations, information on dependencies between functionalities and resources and among resources themselves have to be stored. When dealing with dependencies on an abstract level, e.g. that each application requires an operating system, additional information on the compatiblity among concrete resources must be given.

Both types of information can (and must be) included in the knowledge base by making use of the object property relations *requires* and *isCompatibleTo*, as presented

Requirement	**Semantic Web Services**	**Ontology-Based Configuration**	**This Work**
4.1 Lightweight	◐	●	●
4.2 Taxonomy Resources & Funct.	○	◐	●
4.3 Dependency & Compatibility	○	○	●
4.4 Reduced Modeling Overhead	○	◐	●
4.5 Extensibility	●	●	●
4.6 Standard Compliance	●	◐	◐

Table 4.1: Review on requirements and comparison to related literature

prior in this section. Both dependency and compatibility can be stated on instance or class level.

Reduced Modeling Overhead. Creating and maintaining such an ontology requires a decent amount of work for an employee with skills in semantic technologies. Hence, any reduction in the amount of modeling which is necessary to capture all required information is of great benefit.

A lot of information, especially regarding the dependency and compatibility of resources, is of generic nature, i.e. a certain assertion is not only true for a single instance, but for a whole group of resources. Using the concepts provided in the presented framework, one can model these generic assertions on class level or by using complex rules that make use of concepts of any abstraction level. Recapturing the example from the prior section, the statement that MySQL is compatible to all non-mobile versions of Windows is a great reduction of modeling effort as compared to stating compatibility for each and every version of Windows on instance level.

Extensibility. The extensibility of a model is crucial for its applicability in practice. No generic model can be so comprehensive that it captures all potential specifics arising in different scenarios. By relying on a lightweight approach using OWL as formal language representation, this extensibility is given. Further concepts can be easily introduced and even very complex compatibilities can formulated by means of DL-safe SWRL rules. In addition, external OWL ontologies which are widely available for various domains can be imported and thus used without much additional effort.

Standard Compliance. Creating a proprietary framework brings along the major drawback that information stored within the framework cannot be reused in any other context. Neither is it possible, to import existing information if it is not available in the proprietary format. Hence, standard compliance can be of great benefit. By building the framework on the defacto standard for semantic modeling OWL in conjunction with the standardized rule language SWRL, standard compliance is partially given and allows for an easy import and export of information from an to other data sources. However, a stronger integration with the standardization efforts of the Semantic Web community and the WS-* stack could even improve this compliance and is subject to future work.

Summarizing Review. A summary of this review on requirements and the applicability of related work can be found in Table 4.1.[7] The small overlap between the requirements and the semantic frameworks in the context of Web services can be explained by the different focus of these frameworks. The work by Yang and Dong is more closer related, yet it does not offer the capability for modeling resource dependencies. The work at hand was designed according to these requirements, hence, it has the biggest overlap. However, the standard compliance could be further improved by integrating the ontology with other standards from the WS-* stack.

4.5 Summary

Engineering Cloud services in a mass customization fashion requires explicit knowledge on available resources and functionalities offered to potential consumers. In order for this knowledge to be used in an automated offer creation process, it has to be machine-readable, well-structured and must contain information which allow the derivation of configurations from selected functionalities.

In this chapter, first essential characteristics and requirements for a semantic service description framework leveraging such an engineering phase were gathered. Most important, the framework should allow the creation of a taxonomy on service resources and functionalities, which is capable of storing information on dependency, compatibility and meta information on QoS and costs.

Subsequently, we investigated on related work from two research areas: the semantic description of Web services and ontology-based configuration. The closed proximity was found in the latter area. A three-fold ontology framework was the main contribution of this chapter, allowing to explicitly model the required knowledge in an efficient, extensible and standard compliant manner. Using this framework, reduced modeling overhead can be achieved by defining relations on higher abstraction levels.

Lastly, a qualitative evaluation was given by discussing the proposed solution in context of the posed requirements. While the approach could be further improved regarding its standard compliance, it was found to meet the other requirements as expected.

[7]The fulfillment of requirements for *Semantic Web Services* and *Ontology-Based Configuration* reflects on the presented related literature in the corresponding field of research only. Thus, it does not apply in general.

Chapter 5

Ontology Update Mechanism

In order to rely on the semantic service description framework (presented in the prior chapter) for an automated offer creation of mass customized services, the underlying knowledge base is required to contain most up-to-date information. A knowledge base that does contain outdated information will inevitably result in sub-optimal engineering and optimization decisions. While structural and major content changes to the ontology require manual maintenance by a skilled knowledge engineer, detailed information on instance level on quantitative properties like QoS and price can and should be be updated by an automated mechanism. We therefore present an ontology update mechanism that is capable of holding the knowledge base in a state containing the latest information available.

This chapter thereby is structured as follows. First, a use case for a detailed illustration of the concrete scenario is presented in Section 5.1. It is a refinement of the scenario presented in Chapter 3. Thereafter, requirements for an automated ontology update mechanism are derived in Section 5.2. Related work is introduced in Section 5.3. The core concepts and a proof-of-concept implementation, which are the main contributions of this Chapter, are described in Sections 5.4 and 5.5. We evaluate the ontology update manager in Section 5.6. The chapter is concluded by a brief summary given in Section 5.7.

5.1 Use Case

The ontology update mechanism requires a more detailed use case than the scenario presented in Chapter 3. For using an automated service engineering solution – making use of the semantic service description framework in Chapter 4 and the service engineering algorithm of Chapter 6 – it is inevitable that the ontology contains the latest information available, especially in the context of economically optimal decision making (cf. Chapter 8). In the scenario variant of an intermediary that is offering mass customized services as compositions of third party offers, both price and service level changes will occur on a frequent basis. The frequency hereby can vary from months down to hours. In the single provider variant, quality and cost conditions of service resources typically change in real time, due to the ever changing load on the service provider's infrastructure. QoS and cost values thus can become quite volatile. Any expert system optimizing over different service configurations has to reflect on these values on an instantaneous basis.

A good illustration of the just mentioned challenges can be given through the following example of a service offer by an intermediary on the basis of IaaS.

Example 5.1. *Consider an intermediary offering mass customized services tailored for consumers with computational expensive short-term jobs. As deployment infrastructure, the intermediary relies on various public IaaS offers, e.g. Amazon's EC2 or Windows Azure[1]. Depending on the job characteristics and the current Cloud market situation, different offers or even spot instances[2] potentially are the cost optimal choice. Thus, optimality is subject to the current market situation, which can be quite volatile, thus requiring an automated update mechanism for the knowledge base offering shared access to the intermediary's systems. There is currently no standardized interface for accessing market information (QoS, price, ...) on the available IaaS offers. Information has to be integrated by accessing the data sources offered by each individual provider.*

Example 5.1 serves as illustration and is used for a better understanding of the requirements on the ontology update mechanism which are derived in the following section.

5.2 Requirements

In this section, generic and use case specific requirements are derived. Some requirements can be directly deduced from Research Question 2 (cf. Section 1.1) and Example 5.1, others are generic requirements. The requirements will serve as means for a qualitative evaluation of the presented solution following in Section 5.6.

As stated before, any knowledge base in the context of engineering or optimizing mass customized services has to contain up-to-date information. While updating an ontology's structural date, i.e. concepts and relations among concepts, can be hardly automated, data values like quality and price originating from external data sources can be updated automatically. However, due to many different standards and data formats available, external data sources expose their data using different interfaces and in various data formats. An information integration approach therefore has to be capable of accessing manifold data sources.

Requirement 5.1 [Manifold Data Sources]. *The ontology update mechanism has to support various data sources from different providers. Data access is characterized by the lack of a common access standard, as providers tend to offer their information services through different protocols. A common approach is to offer a REST or SOAP-based Web service. However, statically provided Web content in form of HTML, XML or CSV files are also used in practice. The support for manifold data sources therefore is a prerequisite for integrating service offers by many providers. In addition, the software architecture has*

[1] `http://www.windowsazure.com/`

[2] A dynamically priced offer of compute resources, similar to a spot market for oil, electricity or other commodity resources. Cf. `http://aws.amazon.com/de/ec2/spot-instances/`

to be designed in a modular way, such that additional data formats can be easily integrated without modifications to the core of the update mechanism.

The update mechanism should not be limited to updates on QoS and price information, but engineered as generic as possible, in order to promote its usage also beyond the scope of the presented use case.

Requirement 5.2 [ADAPTABILITY/REUSABILITY]. *The concept should not be limited to a certain ontology type or use case. Adaptability to other use cases requiring automated updates of an ontology (e.g. weather data or stock prices) has to possible without bigger integration effort. An adaptable solution simultaneously fosters its reusability.*

Depending on the entity requiring data updates, the frequency of information changes and the data source, the update process should occur in a parametrized and structured manner to use resources efficiently while simultaneously ensuring up-to-dateness.

Requirement 5.3 [CONTROLLED & STRUCTURED UPDATE PROCESS]. *The update process has to occur in a structured and controlled manner. Information changes can occur at varying frequencies, monthly, daily, hourly and sometimes even down to milliseconds. A configuration parameter defining the update interval is inevitable for reducing redundant data queries. Both data source and data transformation need to be configurable in a well structured way. The update process has to be plannable, i.e. scheduling updates into times with low system load has to be possible.*

In scenarios with large ontologies requiring frequent information updates from many different sources, scalability of the update mechanism is an important issue. A scalable software architecture ensures that decisions based on the semantic description framework are made based on correct information, as scalability reduces time lag potentially stemming from system overload. The system's scalability additionally influences the system's availability as overload situation can be application critical.

Requirement 5.4 [SCALABILITY]. *Update intervals can be short and the number of entities requiring updates and the number of data sources can be large. Therefore scalability of the designated mechanism is an important requirement for its practicability in any real-world scenario.*

Any information update implies changes to the ontology. Naturally, these changes should not endanger its structural consistency, as this would lead to wrong results or unavailabilities of the system.

Requirement 5.5 [CONSISTENCY]. *The update process executes changes on the targeted ontology. These changes should not endanger the ontology's consistency. A guaranteed consistency is inevitable for queries and reasoning operations to function after the update has occurred.*

Information aggregation from heterogeneous external resources entails the risk of data extraction or preparation errors. A visible and comprehensible update process can help reducing such errors.

Requirement 5.6 [Transparency]. *The updated data values have to be visible to the administrator in order to ensure traceability and to reduce incorrect data values. By providing an interface which enables the administrator to follow the update process and the resulting values, wrong values resulting from errors in the data extraction or preparation can be detected. These errors occur when data source change their interface or data structure.*

Reading queries to the ontology have to be possible at all times and from different systems for the semantic service description framework to function in any online scenario with decision making in near or real time.

Requirement 5.7 [Persistency & Shared Access]. *The ontology has to be accessible from a decentralized system landscape and during any update process. Thus, it is important that the ontology is persisted in a centralized manner and update processes does not block the ontology system from reading queries.*

The semantic service description framework is based on OWL. Therefore, the designated approach has to support the Web Ontology Language as formal ontology description language.

Requirement 5.8 [OWL Support]. *The ontology update manager has to be capable of updating knowledge data using OWL as formal knowledge representation language.*

All of the above requirements build the foundation for a consistent, versatile and transparent mechanism for updating information from external resources within ontology systems.

5.3 Related Work

In this section, related work is described and its applicability evaluated against the requirements listed in the preceding section. Four distinct areas or approaches are covered: ontology versioning and evolution, the ontology update language, SMILA, an extensible framework for building big data by integrating data from heterogeneous data sources, and wrappers.

5.3.1 Ontology Versioning and Evolution

By the use of ontologies in professional contexts, their administration, maintenance and progression become increasingly important. An ontology is a static abstract model of a domain of interest. However, information within this domain of interest can change or evolve over time. Thus, the development of an ontology is a dynamic process. In fact, there exists no common and standardized methodology comprising all aspects of a change management for ontologies. Yet, it is an active topic of current research. The main work in this area has been published related to the terms *ontology versioning* and *ontology evolution* [86, 124, 144].

Ontology versioning has its focus on the creation and management of different versions of a given ontology. A new version can emerge from changes to an existing ontology. A different version can also exist to describe the domain of interest from a different perspective. Ontology versioning enables access to and management of different versions of a single ontology [86].

Ontology evolution focuses on ensuring consistency in the change processes to an ontology. Ensuring consistency is important to guarantee that reasoning queries to the knowledge base can be executed and return valid results. The complexity of the evolution process increases with the size of the ontology. Thus, a structured ontology evolution process becomes inevitable for large ontologies [67].

In [85] various requirements on a *change management* for ontologies are presented. Given an ontology in two version $\{V_1, V_2\}$, these are:

Data Transformation. If V_2 evolves through a structural change of V_1, data described within V_1's structure has to be transformed to fit into V_2's structure. If the structural change, for example, is the union of concepts A and B to form C, instances of A and B have to be merged into C.

Data Access. Data described in V_1 has to be equally and correctly accessible from V_2. Compatibility between the two versions has to be ensured, i.e. queries to V_2 should give access to the same data as queries to V_1.

Ontology Update. In a shared environment, local copies can exist for many users. If the central copy is changed, all local copies have to receive the corresponding update.

Consistent Reasoning. The consistency of a changed ontology has to be validated after the change occurs, guaranteeing that reasoning operations continue to be possible.

Verification and Approval. In some scenarios changes to an ontology have to be verified and approved, before the changes can be reflected in the productively used ontology. This is especially the case when several developers work on a central ontology where changes are adopted selectively. Therefore, a user interface allowing both verification and approval or denial of submitted changes to an ontology is required.

5.3.2 Ontology Update Language

The *ontology update language* [99] is a formal language to initiate and handle reoccurring changes to an ontology. The approach is based on *SPARQL Update*[3] [138], an extension to the RDF query language SPARQL (cf. Section 2.1.2.4). While SPARQL is used to find data in RDF graphs, SPARQL Update offers the capability to update and maintain RDF graph structures. Existing RDF graphs can be updated using the

[3]http://www.w3.org/Submission/SPARQL-Update/

operations *insert* (adds data to the graph), *delete* (removes data from the graph) and *modify* (alters data from the graph).

Change Patterns. A change pattern describes how data can change within a certain domain. For example we can take the concept of a child. A child at some point will be an adult, the inversion, however, is not possible. Based on such axioms, formal specifications of possible changes to an ontology can be derived. This formal specification is often called *update rule*. Update rules are created ex-ante by the ontology developer. They are applied once a change request to the ontology arrives. A rule contains both syntactical and semantical checks of the ontology and the resulting changes, ensuring that the change request will lead to a consistent knowledge base. If, for example, an employee is leaving a company, not only the instance of the employee has to be removed, but also all relationships (e.g. "is-supervisor-of") defined in the ontology describing the company's structure.

Every update rule contains a unique identifier. Under *preconditions*, prerequisites can be defined, determining which update rule is applicable for what change request. If more than one rule matches, the first rule found is applied. The actual changes are defined int a *change request pattern*, which describes what changes are made in consequence of a change request. These are declared in the the WHERE clause of a SPARQL SELECT query.

5.3.3 SMILA – Unified Information Access Architecture

SMILA is an extensible framework for building big data and search solutions to access unstructured information in the enterprise. Besides providing essential infrastructure components and services, SMILA also delivers ready-to-use add-on components, like connectors to most relevant data sources [46]. The main purposes of SMILA are information pre-processing and information retrieval. The Eclipse[4]-based software contains deployable data connectors and services, designed for integrating large amounts of unstructured information into enterprise contexts. A scalable architecture enables SMILA to be used in cluster environments.

Architecture. Figure 5.1 shows the architecture of SMILA. The architecture consists of two distinguished components:

- First, data has to be imported into the SMILA system and processed to build either a search index, to extract an ontology or to process the data in some other way.
- Second, the gained information is used to answer information retrieval requests from users, for example search or ontology exploration requests.

The first part gathers its data by either crawling external data sources or by means of an external client. The external client pushes the data from the source into the system using a REST API. After extraction, the data is processed, i.e. an index is built or relevant information is extracted, like name, size, access rights, authors, keywords, etc. The currently offered crawlers support file system, Web and database

[4]`http://www.eclipse.org/`

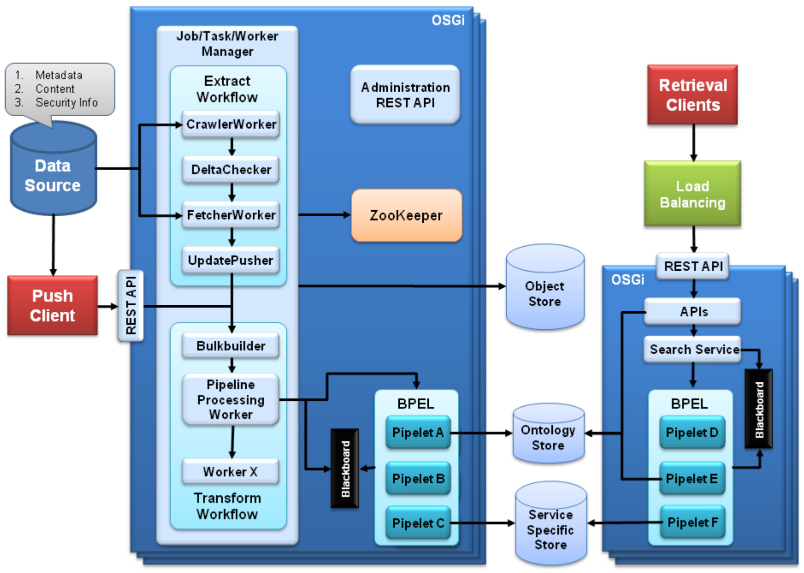


Figure 5.1: The architecture of SMILA

crawling. The focus of SMILA is to enable a search on many different kinds of unstructured documents from different sources.

5.3.4 Wrappers

In the field of information extraction the notion of a wrapper describes a group of special procedures for extracting structured data from semi-structured data sources. A common scenario for the usage of so called Web wrappers is the conversion of data from different Web pages into a relational database, e.g. for extracting names and corresponding e-mail addresses. In general, a wrapper comprises a set of extractions rules and an implementation that is capable of applying theses rules [117]. Extraction rules and implementation commonly are created manually by humans, yet, different research approaches exist, investigating on an automatic induction and maintenance of wrappers [117, 97, 108].

5.3.5 Applicability

In this section, the applicability of the presented related work is evaluated against the requirements of Section 5.3.

5.3.5.1 Ontology Versioning & Evolution

Ontology Versioning emphasizes the creation and management of different versions of an ontology. The ontology update mechanism induces changes on the service description ontology and thus also creates different ontology versions. Within the scope of this work, only the most current version is relevant for decision making. If a history of data updates becomes relevant, e.g. if historic prices or QoS values influence decision making, ontology versioning obtains significance and all aspects of ontology versioning have to be considered.

Ontology Evolution cares about the consistency of ontologies undergoing structural changes. This is an important facet for an ontology developer. For an automated update mechanism with focus on updating data values, i.e. instances and their data properties rather than structural changes on class level, only partial aspects of the ontology evolution process are relevant.

5.3.5.2 Ontology Update Language

The *Ontology Update Language* enables structured and controlled updates to an RDF graph. OWL, however, is not supported. Nevertheless, the concept of the change pattern can be applied to the scenario of the ontology update mechanism. The scenario of updating prices or QoS values thereby resembles a special case of a change pattern, as only data values of datatype properties are affected. Compared to the capabilities of the ontology update language, an automated update process is very constrained regarding the required functionality. This is especially the case since changes are not triggered by users, but by update rules which can be validated ex ante.

5.3.5.3 SMILA

SMILA's extensible framework for accessing and integrating unstructured data potentially can be the groundwork for an ontology update mechanism. Documents containing price and QoS information can be read by SMILA's crawling engine. Information extraction can be achieved by means of the integrated BPEL engine and a suitable workflow. The obtained information can be integrated into an ontology using SMILA's ontology store. However, the concept is limited to the integrated crawlers and the implementation of additional crawlers for currently not supported data sources is a cumbersome task. Regarding the ontology store, SMILA implementation is free of any predefined semantics. OWL integration is possible, but not implemented. While a SMILA-based approach to achieve an ontology update mechanism seems feasible, it appears to be less versatile and requires more effort than the development of a specific ontology update mechanism.

5.3.5.4 Wrappers

Wrappers have their focus on the extraction of data from semi-structured information sources into structured data. Depending on the implementation, existing work in the area of wrappers fulfills many of the posed requirements for an automated ontology update process and therefore can serve as a vital input and extension to the approach in the work at hand. Particularly, the concept of an automatic creation and maintenance of wrappers could be transferred to the creation and maintenance of update rules in this scenario. Nonetheless, there is currently no wrapper approach known to the author, which is capable of both extracting data from heterogeneous data sources (other than Web pages) and using the gained information for updating data values in OWL ontologies.

5.3.5.5 Summary

As summary, one can conclude that neither *Ontology Versioning & Evolution*, the *Ontology Update Language*, *SMILA* nor *wrappers* completely fulfill the requirements for an ontology update mechanism that is appropriate for the mass customization scenario described in this thesis. Nevertheless, the designated approach presented in Sections 5.4 and 5.5 can benefit from the related work which has been investigated on in this section.

5.4 Concept

This section presents the main contribution of the ontology update manager. First, major concepts and design decisions are described in Section 5.4.1. Second, the architecture of the ontology update mechanism is defined in Section 5.4.2. Lastly, we discuss different types of data sources and how to extract data from them in Section 5.4.3.1.

5.4.1 Underlying Concepts and Design Decisions

This section covers underlying concepts or principles and design decisions. First the concept of an update rule is described, followed by the notion of a self-updating ontology. Design decisions regarding the ontology persistence and the information integration are presented in two further subsections.

5.4.1.1 Update Rule

The goal of the ontology update mechanism is to update arbitrary data values of an ontology. Current values have to be extracted from external data sources and stored into the ontology. The concept is therefore similar to a data warehouse. A data warehouse is a central database designed to store large sets of data from different sources. Before the extracted data can be stored, it has to be transformed to match the data scheme defined in the data warehouse, such that queries designed for the scheme can be executed on the extracted data. Integration rules manage this transformation process. The process of integrating data into a data warehouse is called *ETL process* and consists of three parts [98]:

Extraction. All processing steps necessary to extract the desired data from the data source. Since various types of sources exist, different extraction methods are necessary. E.g. for a relational database as data source, an SQL operation has to be launched, for an XML source, the XML document has to be loaded into the parser.

Transformation. The steps required for transforming the extracted data to meet the data warehouse scheme. Typical transformations are for name, date or time values or address information, which are available in many different data formats.

Loading. The actual process of loading the extracted and transformed data into the data warehouse.

The concept of the ETL process can be applied to the problem definition of the ontology update manager. Current information from different and heterogeneous data sources have to be integrated into a service description ontology. First, we want to provide a formal description of the integration process. The integration process for updating an ontology comprises the extraction of data values from a data source, transformation commands and the storage of the newly obtained values into the ontology. For describing the integration process of an instance of the ontology, the concept of an update rule is used. An update rule thereby refers to the above mentioned integration rule in the context of data warehouses. Its purpose is to keep the data type property within an ontology up-to-date, i.e. it formally describes the update specification for an instance of a service resource.

Definition 5.1 [UPDATE RULE]. *An update rule is the 6-tuple* $\rho = (C, D, I, E, U, T)$ *with*

- *C: <OWLIndividual> instance subject to the update*
- *D: <OWLDataProperty> data type property of C containing data value requiring update*
- *I: <Int> update interval in seconds*
- *E: <ExtractionMethod> extraction method*
- *U: <URI> URI of the data source containing the latest information values*
- *T: <Object> transformations required after extraction of the data value*

The update rule is applied after the update interval has been exceeded, i.e. the elapsed time between the last update is greater than the designated update interval. This enables a structured and controlled update process. An instance of the ontology can be assigned more than one update rule. This way, different data sources for the same information can be addressed, either as backup for malfunctioning data sources or to compare, average or minimize/maximize over different values. In the intermediary variant of the scenario (cf. Section 3.1.2) for example, this allows to obtain prices for a certain 3rd party service resource from different providers.

5.4.1.2 Self-Updating Ontology

Since the ontology update mechanism is applied to update instance information within an ontology, it seems natural to maintain all update rules within the same ontology. This has three major advantages, as it (1) centralizes both knowledge and its update rules in one place, (2) allows to refer from the update rules directly to

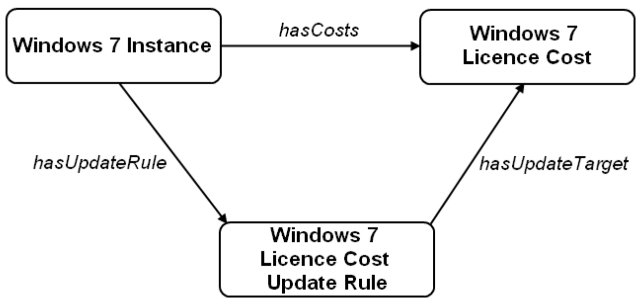


Figure 5.2: Example for object properties *hasUpdateTarget* and *hasUpdateTarget*

instances of the same ontology and (3) thus ensures consistency, as the validity of rules can be checked automatically by means of reasoning.

The service ontology is extended with a new class *UpdateRule*, having different subclasses to address different data sources. Examples are *UpdateRuleREST*, *UpdateRuleXML*, *UpdateRuleHTML*. The class *Resource* receives the additional object property *hasUpdateRule*, which indicates the relation between an instance of a service component and its corresponding update rule. The object property *hasUpdateTarget* refers to the instance receiving the update. Depending on the use case, this could be the same instance as the one associated with the update rule. However, the date value requiring the update can also be the data type property of another instance for describing a quality or cost aspect of the service component. An example for the usage of both properties is depicted in Figure 5.2.

Additional data type properties are introduced to capture the source and the time of the latest updated value. *hasValueUpdatedSourceURI:anyURI* defines the source of the current value, *hasValueUpdatedTime:dateTime* the instant of time of the last update operation that has written the current value.

A concrete update rule instance belongs to either of the subclasses of *UpdateRule*, defining its extraction method. Additional data type properties of the class *UpdateRule* are

- *hasURuleInterval:int* – contains the update interval in seconds
- *hasURuleSourceURI:anyURI* – the URI of the data source
- *hasURuleTransformation:String* – contains potential transformations

Through this extension to the service ontology which captures all aspects of an update rule, it is possible to model a domain ontology containing all commands necessary for it to be self-updating. An software architecture reading this information and controlling the update process is presented in Section 5.4.2.

5.4.1.3 Ontology Persistence and Shared Access

For persisting the ontology and providing a shared user access, the architecture is based on OWLDB (cf. Section 2.1.3.3). OWLDB is open source potentially allowing software adaptions if the use case requires such adaptions. As a knowledge base for engineering mass customized services can get quite substantial, an efficient database storage is beneficial over in-memory ontology solutions. A central database approach enables shared access by different applications running on different systems to the same knowledge base. Simultaneously, main memory consumption is lowered, as the ontology can be loaded partially and only according to the need of information. The semantic description frameworks relies on OWL as ontology language. OWLDB as object relational projection of OWL API offers a higher performance compared to triple-based storage like Jena SDB when executing OWL queries [68].

5.4.1.4 Information Integration

The presented concept is based on a *material* information integration approach, i.e. the data is collected and stored in a central ontology. This knowledge is offered to applications which process the data or do reasoning with it. This implies that all information has to be present at time of the query, as opposed to a *virtual* information integration approach, where the latest information is gathered on demand. Thus, all query and processing steps can be executed in the local memory, yielding a significant performance improvement over any virtual integration approach. While a virtual approach offers the most current information, this benefit is traded against a loss of performance. The material approach, on the other hand, requires a regularly executed update process to ensure data up-to-dateness.

Overcoming data heterogeneity. Information integration requires mastering both technical and semantical data heterogeneity. Let us for example consider price information. There exist many different data sources for price information. In addition, different technologies for representing price data and different semantics can be found and have to be dealt with. The obtained data differs in currency, format, syntax and decimal representation. The following values all potentially carry the same price information: 1.000,500, EUR 1000,50, 1000,50€, 100050, 1,251 USD, 1251$, ...

Update Result. We define a common information model for the resulting information from the update process. After data extraction and transformation, an update event is triggered bearing the current data value. The data model for an update result is defined as follows:

Definition 5.2 [Update Result]. *An update result is the 4-tuple* $\tau = (C, U, V, T)$ *with*

- *C: <OWLIndividual> instance subject to the update*
- *U: <URI> URI of the data source containing the latest information values*
- *V: <Object> new data value from data source*
- *T: <Timestamp> timestamp at which the update occurred*

Update results are written back into the ontology. They represent the most current value of a data value of interest. Even if the value has not changed since the

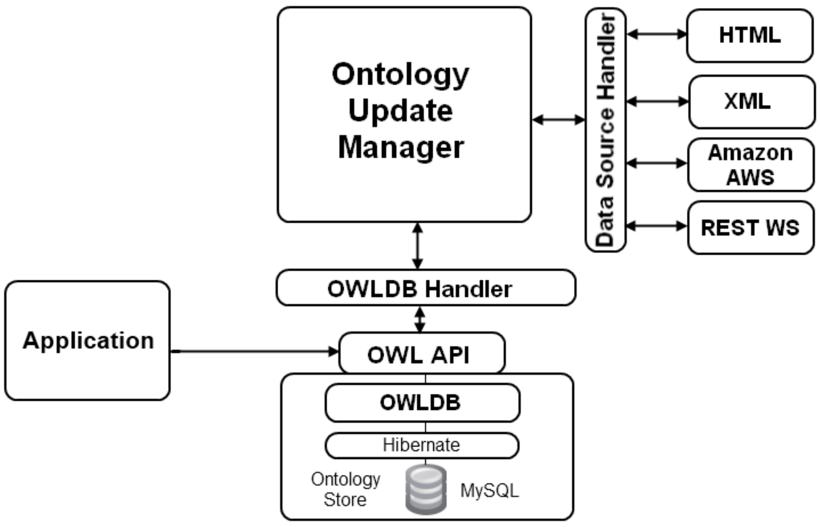


Figure 5.3: Architecture of ontology update mechanism

preceding update, knowledge on the continuous validity of the current value can be derived. Before the actual value can be written, additional checks have to be performed. If there exists only one update rule for a certain information, the value can be written directly. In case there exists more than one rule, the applicability of the various rules has to be validated. In some cases, the minimum or maximum of all obtained values yields the semantically correct information, in other cases it is the average of all values.

5.4.2 Architecture

The architecture of the ontology update mechanism comprises three main components: *OWLDB Handler* for accessing the ontology using OWLDB, the *Data Source Handler* which is responsible for the data extraction from the data source and the *Ontology Update Manager*, building the core component responsible for the coordination of the update process. It is depicted in Figure 5.3.

The *Ontology Update Manager* does not directly access the OWLDB-based ontology, but communicates with an abstraction layer, the *OWLDB Handler*. The same principle is applied to the access mechanism to external data sources. Instead of directly accessing the data source, the *Ontology Update Manager* calls the *Data Source Handler* which connects to the data source according to the present update rules and extracts the desired data value. The introduction of the layered architecture ensures the extensibility for further functionality, additional data sources or enables the exchange of certain algorithm. By relying on the concept of encapsulation, such modifications are possible without changes to the core component, easing the maintenance of the software solution.

5.4.3 Sequence of Update Process

Figure 5.4 shows a UML sequence diagram depicting the processing steps for extracting a data value from an XML document. First, the update rules $\{\rho_1,\rho_2,...\}$ are

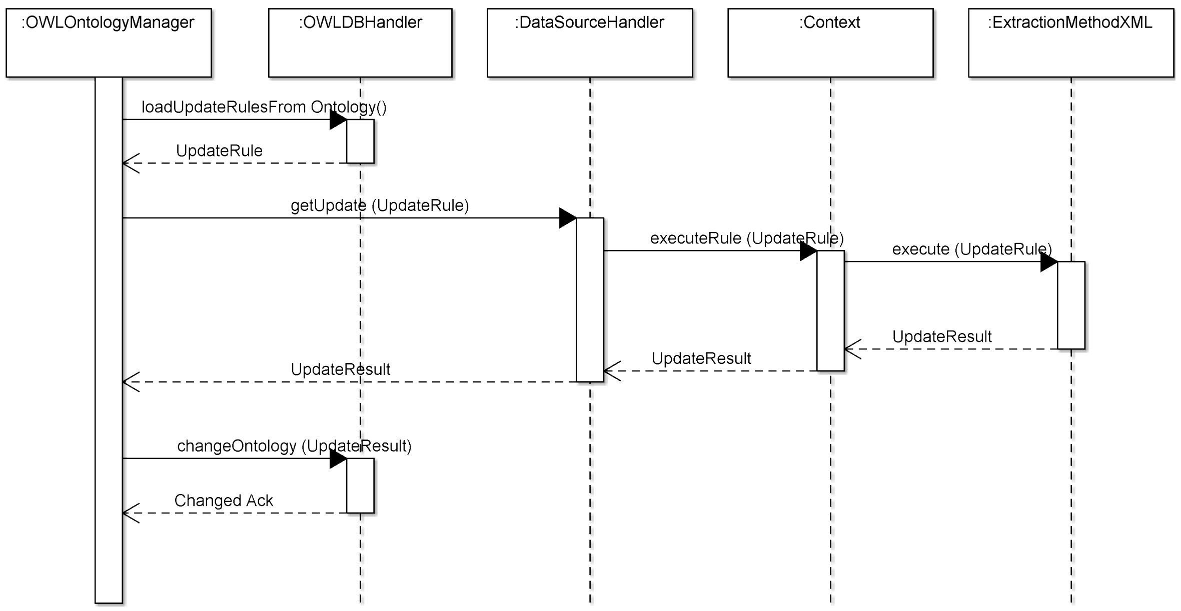


Figure 5.4: Sequence diagram of update process

loaded from the ontology into the core component by calling the method *loadUpdateRulesFromOntology()* from the *OWLDB Handler*. Thereafter, the method *getUpdate()* from the Data Source Handler is called, comprising one loaded rule ρ as method parameter. The extraction method parameter E is read by the *Data Source Handler* and the corresponding extraction method is selected. The *executeRule()* method of a context object is called with the update rule ρ as parameter. This results in the execution of the *execute()* method of the selected extraction method – the XML extraction method it this example. The data is extracted from the XML document followed by transformations T defined in the update rule ρ. The result in form of an update result object τ is returned to the *Ontology Update Manager*. At last, the method *changeOntology()* from the *OWLDB Handler* is called with τ as parameter. The value V of τ is written into the ontology and the success of the write transaction is acknowledged.

5.4.3.1 Data Extraction

When extracting data values from a given data source, both the data representation and the transport protocol providing the data strongly influence the extraction process. Representations for example can be of raw binary type, HTML or XML format or even image data. For each data representation, a corresponding data extraction method has to be designed. Commonly found data representations are XML, HTML or JSON. The different data representations can be categorized as unstructured, semi-structured or structured data. Transport protocols are typically HTTP based interfaces like RESTful services or the more complex WSDL based services. Less common are file based protocols, which are found in intranet contexts.

Unstructured Data. Unstructured data is found in documents or images and is characterized by the fact, that the data representation is not machine-readable without the use of complex algorithms. Typically, the information is easily extractable and interpretable by humans, but hard to recognize for machines. Examples could be prices in a catalog available as PDF[5] file or QoS information that is rendered as an image file. In these cases, pattern recognition (e.g. using regular expressions) and OCR[6] algorithms come into play.

Semi-Structured Data. Semi-structured data is characterized by the fact, that it contains both structured data and unstructured data. A typical example for semi-structured data is free text which is structured by machine readable tags, as it is the case in HTML Web pages. In this case, the interpreter (i.e. Web browser) knows how to handle the HTML tags like `<title>` or `<h1>`; the text contained within the tags, however, has no meaning to the interpreter. In order to extract information from semi-structured data like HTML, an index can be built, as it is done by search engines for the World Wide Web. Alternatively, the HTML file can be parsed into a document object model (DOM), which is a tree structure of the underlying document. Another approach is to transform an HTML document to an XHTML[7] document, which is an XML conform representation of the HTML document. Other than HTML, XHTML is well-formed, i.e. it is easier to parse and can be transformed by XSLT[8] processors.

Structured Data. For any data to be classified as structured data, all contained data has to follow a well defined structure. I.e. all data values are characterized by predefined and machine readable types. This allows automated parsing and automated data transformations. Examples for structured data are database tables or XML documents. For the latter, XSLT as transformation stylesheet for XSLT processor can be used to extract data values. This can be done for example by traversing the documents tree structure or by matching tag or attribute names.

Data sources offering structured data are the preferred source for an automated data extraction and ontology update. Yet, especially on the Web, many potential sources are not intended for automated data extraction, but designed for human readers and thus provided as semi-structured data.

5.5 Implementation

The presented concept is implemented in Java 1.6 with the Ontology Update Manager running as a background process. A UML class diagram showing the implementation is depicted in Figure 5.5. The background process can be monitored through the console where every step of the update process is logged. Additional transparency can be achieved by running a separate process that displays the state of the updated instance following a certain update interval. Four different extraction methods are implemented as proof-of-concept: *HTML* and *XML* as generic methods for accessing static or dynamically created web content; *Amazon SOAP Web service* and *Amazon REST Web Service* extraction methods for reading Amazon Elastic Cloud offers from Amazon's Web service. Web service, other than static or dynamic web

[5]Portable Document Format
[6]Optical Character Recognition
[7]eXtensible Hypertext Markup Language
[8]Extensible Stylesheet Language Transformation - part of the Turing complete XSL language [83]

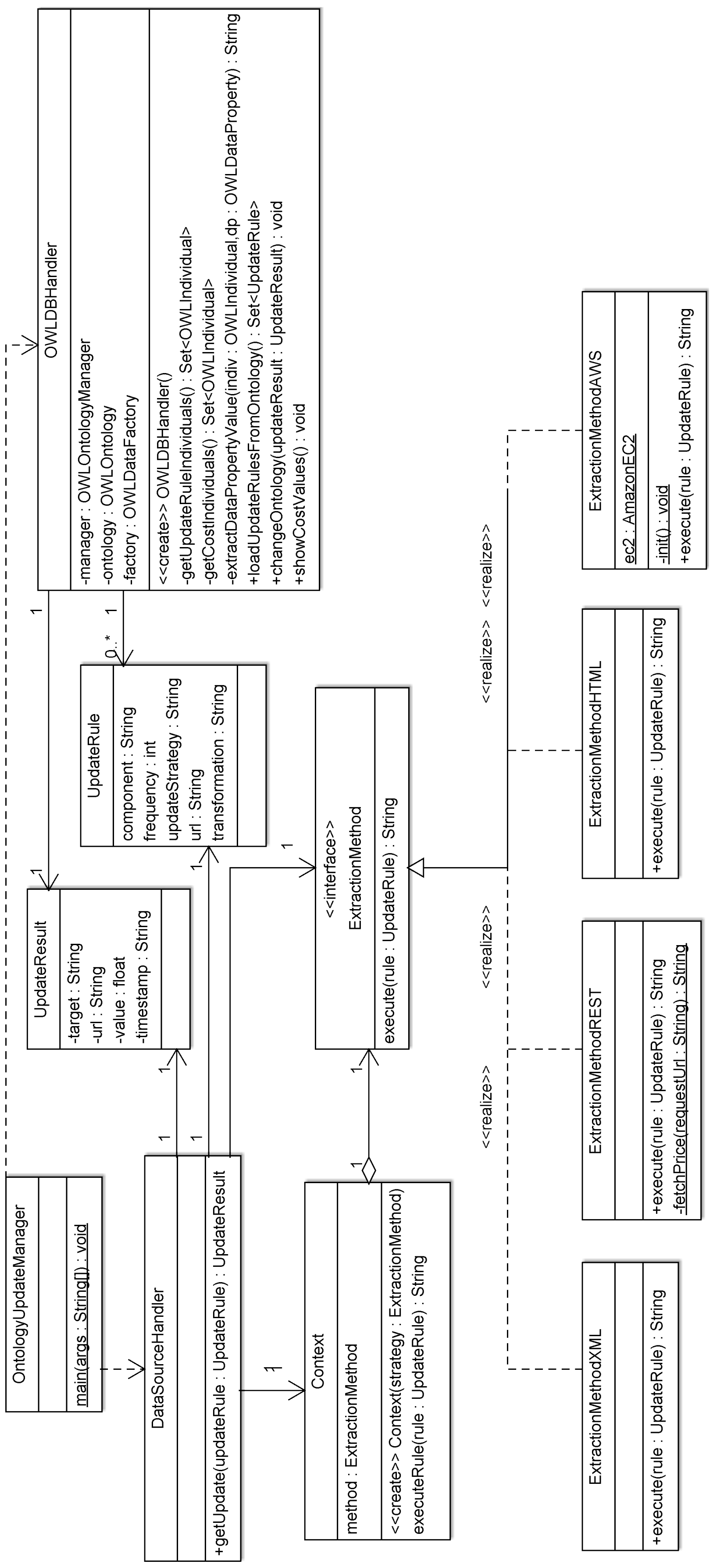

OWLDBHandler
-manager : OWLOntologyManager
-ontology : OWLOntology
-factory : OWLDataFactory
<<create>> OWLDBHandler()
-getUpdateRuleIndividuals() : Set<OWLIndividual>
-getCostIndividuals() : Set<OWLIndividual>
-extractDataPropertyValue(indiv : OWLIndividual,dp : OWLDataProperty) : String
+loadUpdateRulesFromOntology() : Set<UpdateRule>
+changeOntology(updateResult : UpdateResult) : void
+showCostValues() : void
UpdateRule
component : String
frequency : int
updateStrategy : String
url : String
transformation : String
UpdateResult
-target : String
-url : String
-value : float
-timestamp : String
<<interface>>
ExtractionMethod
execute(rule : UpdateRule) : String
OntologyUpdateManager
main(args : String[]) : void
DataSourceHandler
+getUpdate(updateRule : UpdateRule) : UpdateResult
Context
method : ExtractionMethod
<<create>> Context(strategy : ExtractionMethod)
executeRule(rule : UpdateRule) : String
ExtractionMethodAWS
ec2 : AmazonEC2
-init() : void
+execute(rule : UpdateRule) : String
ExtractionMethodHTML
+execute(rule : UpdateRule) : String
ExtractionMethodREST
+execute(rule : UpdateRule) : String
-fetchPrice(requestUrl : String) : String
ExtractionMethodXML
+execute(rule : UpdateRule) : String
<<realize>>
<<realize>>
<<realize>>
<<realize>>
1
1
0..*
1
1
1
1
1
1
1
1
1
1

Figure 5.5: UML diagram of prototypical implementation

content, expect input parameters that influence the response. Data access for all extraction methods is based on the HTTP protocol.

5.5.1 Example Update Rules

As a proof-of-concept, several update rules as prototype for different extraction methods and transformation coping with different challenges arising from the different data formats were implemented. Examples for such rules are presented in the following.

5.5.1.1 HTML

For the prototypical implementation of an HTML update rule, a regular expression based approach is chosen to address this semi-structured data format. First, the HTML Web page is accessed via HTTP from a Web server. A regular expression serves as pattern for searching the data value of interest from the complete HTML document. Lastly, the concrete data value is obtained from the regular expression engine. Consider for example an HTML document containing the following price information:

Listing 5.1: Sample HTML Document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣4.01␣Transitional//EN">
<html xmlns:og="http://opengraphprotocol.org/schema/" >
....
<span class="styple4aa"> 196.70&nbsp;&euro;</span>
....
</html>
```

The corresponding update rule $\rho = (C, I, E, U, T)$ for extracting the price information is defined in the following:

- $C :=$ *WindowsServer2008_LicenceCosts*
- $D :=$ *hasValue*
- $I :=$ 7200s
- $E :=$ HTML
- $U :=$ `https://anyshop.com/pricelist/windows_prices.html`
- $T :=$ <span class="'style4aa"'> (.*?)&nbsp;&euro;</span> *(regular expression)*

This example implementation expects a regular expression. In the example, the number "196.70" is extracted and returned as data value. The data type property *hasValue* of instance *WindowsServer2008_LicenceCosts* is updated accordingly. This rule is executed every 7200s, i.e. every two hours.

A drawback of this very simple approach is, however, that it will fail if the returned HTML document slightly differs from the expected structure.

5.5.1.2 XML

The method for extracting data values from structured information contained in an XML document is implemented by means of XSLT transformations. The XML document is read by a standard Java XML parser and than transformed using a suitable XSLT stylesheet. A sample XML document could be the following:

Listing 5.2: Sample XML Document – sample_pricelist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
  <server>
    <name>DellPowerEdgeR905</name>
    <price>8763</price>
  </server>
  <server>
    <name>DellPowerEdgeR300</name>
    <price>890</price>
  </server>
</doc>
```

For getting the desired data value, the well-defined structure of the XML document has to be analyzed and then used to create a suitable transformation. The corresponding sample XSLT stylesheet which extracts the price value for ontology instance *DellPowerEdgeR905* hence is the following:

Listing 5.3: Sample XSLT Document – transformation.xsl

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:for-each select="doc/server">
        <xsl:if test="name='DellPowerEdgeR905'">
          <xsl:value-of select="price"/>
        </xsl:if>
      </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

To refer back to the update rule notion, we can express the rule as $\rho = (C, I, E, U, T)$ with

- $C :=$ *DellPowerEdgeR905_Costs*
- $D :=$ *hasValue*
- $F :=$ 14400s
- $E :=$ XML
- $U :=$ `https://anyshop.com/pricelist/sample_pricelist.xml`
- $T :=$ transformation.xsl

5.5.1.3 SOAP Web Service

As an example for an implementation of the SOAP Web Service extraction method serves a client for getting price information on Amazon's EC2 Cloud services. Amazon offers a SOAP-based Web service together with an additional Java API[9] for accessing this Web service. It offers a broad functionality including current and historic price information on the Amazon Cloud spot market, where virtual compute instances are offered with a dynamic pricing mechanism. Prices for these spot instances are set on an hourly basis by Amazon. The corresponding update rule following the generic notation is the following: $\rho = (C, I, E, U, T)$ with

- $C :=$ AmazonEC2StandardSmall_Spot_Costs
- $D := \mathit{hasValue}$
- $F :=$ 3600s
- $E :=$ SOAP AWS
- $U :=$ m1.small *(EC2 instance type)*
- $T := \emptyset$

Potential values for the EC2 instance type are: m1.small, m1.large, m1.xlarge, t1.micro, m2.xlarge, m2.2xlarge, m2.4xlarge, c1.medium, c1.xlarge, cc1.4xlarge, cg1.4xlarge.

5.5.1.4 REST Web Service

Amazon offers information on its complete product assortment over the *Amazon Product Advertising API*[10]. Data like price information of all products offered by Amazon can be access via a REST Web Service. As an example for accessing REST Web service, an extraction method for obtaining price information on products offered by Amazon is implemented. For obtaining access to Amazon's REST service, a secret key for signing the request is required. A sample rule using this extraction method in formal notation is the following: $\rho = (C, I, E, U, T)$ with

- $C :=$ Windows7_LicenseCosts
- $D := \mathit{hasValue}$
- $F :=$ 86400s
- $E :=$ REST AWS
- $U :=$ `http://ecs.amazonaws.de/onca/xml?ItemId=B002NLKNRY`
  (ItemID refers to the product Windows 7)
- $T :=$ amazon.xsl

[9] `http://aws.amazon.com/de/sdkforjava/`
[10] `http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/`

The resulting information is encoded in an XML document, which has to be transformed by means of an XSLT transformation (*amazon.xsl*) to obtain the desired data value.

5.5.1.5 Adding Extraction Methods

If additional data sources require an alternative extraction method, a new extraction method has to be implemented and added to the set of existing methods. The Java class implementing the new method has to implement the interface *Extraction-Method*. The following five steps are necessary in order to use the additional method:

1. **Ontology Extension.** The service ontology (cf. Section 4.3.1) has to extended by adding a new class for the new extraction method as subclass of *UpdateRule*.
2. **Implementation.** A new Java class implementing the interface *Extraction-Method* has to be implemented, containing the necessary coding steps for accessing the data source and doing the required transformations.
3. **Registration to OWLDB Handler.** The new subclass has to be registered to the *OWLDB Handler*, such that update rules of the newly created extraction method are loaded.
4. **Registration to Data Source Handler.** The new extraction method has to be registered to the Data Source Handler, such that it can be executed in the corresponding procedure.
5. **Update Rule Creation.** The rules using the newly added extraction method have to be defined and added to the domain ontology.

5.6 Evaluation

In this section the above presented concept of an ontology update manager and its implementation are evaluated. We begin with a quantitative evaluation in Section 5.6.1, investigating on the system performance. In addition, a list of relevant issues for conducting a more exhaustive quantative evaluation is given. In the second part of this section, a qualitative evaluation follows in Section 5.6.2, contrasting the presented concept with the posed requirements and discussing open issues.

5.6.1 Quantitative Evaluation

A quantitative evaluation is characterized by an objective, i.e. measurable, assessment of a given solution to a certain problem. An extensive performance evaluation thereby examines the scalability of the presented approach, which is a major requirement for it to function in a real life setting. An exhaustive quantitative evaluation of a software solution additionally requires further tests. While these tests exceed the scope of this thesis, a description of the potential issues, which should be investigated on in such a test, is given.

Extraction Method	800 MHz	1200 MHz	1800 MHz
HTML	2.122	1.278	0.922
XML	0.270	0.180	0.126
SOAP AWS	2.733	2.658	2.644
REST	0.488	0.317	0.225

Table 5.1: Average execution time [s] over 100 executed update rules

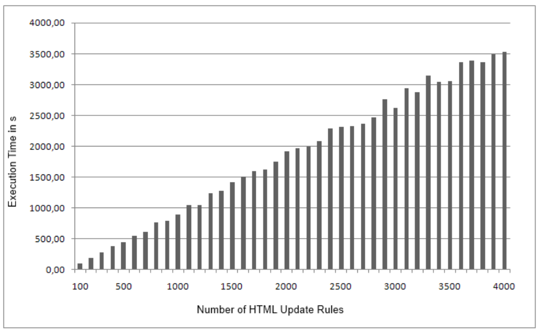
5.6.1.1 Performance Evaluation

To evaluate the proposed solution against Requirement 5.4 (scalability) various load tests have been performed to benchmark the ontology update manager concept. Considering scalability, two different concepts have to be differentiated, the horizontal and the vertical scalability [60]. A vertical scalability is given, if additional load can be handled by proportionally adding system resources (processing power, memory, storage, etc.) to a single system. Horizontal scalability is given if additional load can be handled by proportionally adding further compute nodes to a parallelized software system. In the latter, only vertical scalability is investigated, as vertical scalability can be achieved more efficiently than horizontal scalability and thus is typically tried first.

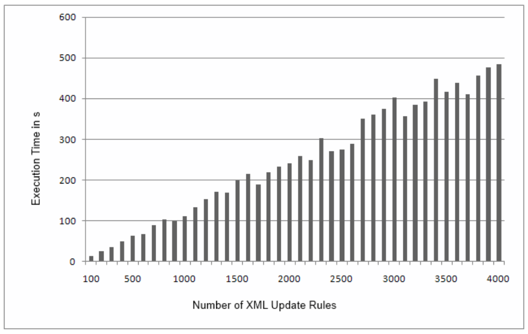
As the ontology update procedure does not involve reasoning steps, the execution time mainly depends on the number of instances and the extraction method which has to be applied. An important external factor is network latency. As the data sources implemented in the prototype are connected trough the Internet, a reliable Internet connection is inevitable for serious benchmarks. The tests therefore were conducted from within the fast access internet connection of the Karlsruhe Institute of Technology, which is offering a sufficient band width for the amount of data transferred. Yet, latency due to network congestion or slow server response time on the side of the data source providers cannot be averted and explains some of the volatility of the obtained results.

Test Environment. All tests were conducted on a single personal computer with the following characteristics: *Intel Core 2 Duo 1.8 GHz Processor, 2 GB DDR2 RAM, 100 MBit/s LAN connection, Windows XP SP3 operating system, Java Version 1.6*.

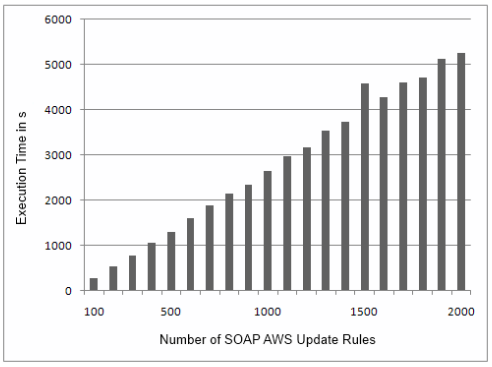
A first test was conducted to investigate on the influence of processor clock rate on the execution time of the different extraction methods. Therefore 100 update rules of a given type were executed on an ontology containing a total of 300 instances. The tests were performed in sequential order, beginning with 800 MHz, followed by 1200 MHz and 1800 MHz. The results are shown in Table 5.1. All update rules are found to be scaling with processor clock speed, yet, with one exception. The update rule connecting to the Amazon SOAP Web service only showed slightly increased performance. The processor speed has almost no influence, since the long execution time accrued from the communication with the SOAP Web service. In fact, the conducted tests with this extraction method did not lead to a full processor load. As expected, the HTML extraction method is most costly with respect to computation time due to the regular expression matching.



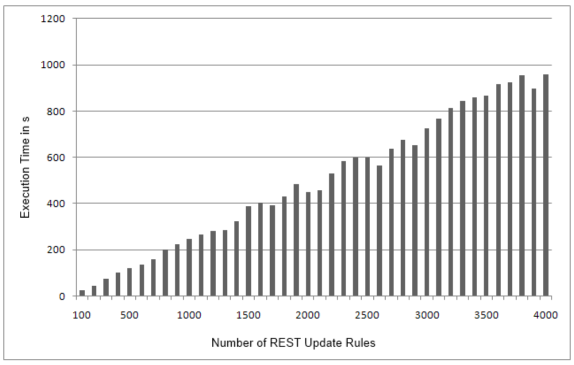
(a) HTML Extraction Method



(b) XML Extraction Method



(c) SOAP Extraction Method



(d) REST Extraction Method

Figure 5.6: Benchmark of execution time [s]

While the approach in general is found to scale with processor speed, a second performance test was conducted to find out the influence of a growing number of instances requiring an update from an external data source. The results of the conducted experiments are depicted in Figure 5.6. All tested extraction methods show a linear increase in time with a growing number of executed update rules. Hence, it can be assumed that the update mechanism has a time complexity of $\mathcal{O}(n)$ for all tested extraction methods, with n the number of update rules in the system.

5.6.1.2 Further Quantitative Evaluations

For a broad quantitative evaluation, further tests and questionnaires are inevitable to evaluate the benefit of the presented solution. We omit such tests, as they exceed the scope of this thesis. However, we want to briefly present the most relevant questions to be raised:

Horizontal Scalability. What is the potential reduction in the execution time for the update rules when the update mechanism is parallelized and distributed among several compute nodes?

Data Extraction Errors. What likelihood of data extraction errors and how do these errors affect the system?

Update Rule Lifecycle. What is the average lifespan of an update rule and how to manage the lifecycle of an update rule?

Benefit/Cost Ratio. What is the actual added value of having up-to-date information in the context of optimizing mass customized Cloud services and what are the corresponding costs accruing from the ontology update mechanism?

5.6.2 Qualitative Evaluation

In the following, the requirements of Section 5.2 are reviewed and the presented concept evaluated against them.

Manifold Data Sources. In this chapter, four different extraction methods (that have been prototypically implemented) were presented. Further data sources can be added by following the guideline in Section 5.5.1.5, enabling the presented concept to potentially handle many different protocols and formats without modification to its core.

Adaptability & Reusability. The presented concept is capable of updating any OWL-based ontology. The ontology, however, has to be adapted by adding the concept *UpdateRule* and the other additional classes and object properties mentioned in Section 5.4.1.2.

Controlled & Structured Update Process. An update rule is a formal representation of a recurring update process. The information contained like the update interval, the instance to be updated or the datatype property is used to ensure a controlled and structured update process.

Scalability. The presented approach was found to be linearly scaling with an increasing number of update rules.

Consistency. The software component *OWLDB Handler* is responsible for writing updates to the ontology. By calling the method *changeOntology()*, updated values are written into the datatype property. To ensure consistency, the data format of the update value is checked before allowing write access. As these are the only changes applied to the ontology, consistency is guaranteed.

Transparency. All information on the execution of update rules are written into a log file. In addition, a process for monitoring current values was implemented. Nevertheless, further improvements could be made by implementing a history of values and by providing user-friendly graphical interfaces.

Persistency & Shared Access. The usage of OWLDB as central ontology persistence layer allows both an efficient database persistence and shared access to the knowledge base.

OWL Support. OWL is supported through relying on OWLDB as database persistence implementation of the official OWL API.

The results of the above qualitative evaluation and the applicability of related work are summarized in Table 5.2. Ontology versioning and evolution are omitted in this overview due to the completely different focus.

Requirement	Ontology Update Language	SMILA	Wrappers	This Work
5.1 Manifold Data Sources	○	◐	◐	●
5.2 Adaptability/Reusability	●	◐	●	●
5.3 Structured Update	◐	●	●	●
5.4 Scalability	◐	●	●	●
5.5 Consistency	●	●	●	●
5.6 Transparency	●	●	◐	◐
5.7 Persistency & Shared Access	○	◐	●	●
5.8 OWL Support	●	◐	○	●

Table 5.2: Review on requirements

5.7 Summary

An ontology serving as a knowledge base for engineering and optimizing Cloud services is required to contain most recent information on both quality and costs of service resources. In this chapter, a mechanism for automating data updates to an ontology was presented. Following various design concepts like the ETL process of a data warehouse or the concept of a self-updating ontology, both concept and proof-of-concept implementation for such an update mechanism were presented. Apart from describing related work in this area, the chapter also provided a performance evaluation and a qualitative evaluation against the requirements originating mainly from the use case and Research Question 2.

By means of the presented contribution, it is possible to keep information on service resources stored in an OWL ontology up-to-date through integrating manifold data sources from the Web in a scalable, transparent and consistent manner.

Chapter 6

Service Engineering Algorithm

In the offer creation phase for a mass customized Cloud service, service provider and consumer need to agree on the functionalities provided by the desired Cloud service. In the envisioned offer creation process (cf. Chapter 3), this is achieved by the consumer selecting one or more (abstract) functionalities by means of a catalog to choose from. Thereafter, the functional engineering phase takes care of finding feasible service configurations which are capable of providing the requested functionalities. A semantic service description framework serving as knowledge base for both the catalog and the knowledge on how to find feasible configurations was presented in Chapter 4. A further challenge, however, lies in deriving all feasible configurations matching the functional requirements of a given consumer request.

The current chapter addresses this challenge by providing a service engineering algorithm which is capable of using information stored in such a framework to derive all feasible configurations by iteratively resolving dependencies starting from a set of functional requirements specified by the service consumer. It is structured as following. Requirements posed upon the algorithm are given in Section 6.1. Thereafter, related work from the areas of Web service composition, constraint programming and ontology-based configuration is presented in Section 6.2. A formal model for describing functional requirements and the resulting service configuration graph follows in Section 6.3. The chapter's main contribution – the service engineering algorithm – is described in Section 6.4. Implementation and evaluation are presented the subsequent Sections 6.5 and 6.6. The chapter is concluded by a summary given in Section 6.7. The results of this chapter are largely based on [65].

6.1 Requirements

In order for the designated algorithm to be able to derive a service configuration for a certain consumer request based on a semantic description of available service resources, we recognize four requirements as important specification for it to function in the scenario of this thesis. As stated before, the functional requirements are selected from a catalog of functionalities as provided by the semantic service description framework.[1] The algorithm therefore has to work on the basis of an ontology describing service functionalities and resources.

Requirement 6.1 [ONTOLOGY-BASED KNOWLEDGE BASE]. *The dependency and compatibility information required by the algorithm should be stored in an ontology, as it is for example provided by the semantic service description framework.*

Outgoing from the abstract functional requirements specified by the consumer potential service configurations have to be derived. The algorithm has to resolve dependencies specified in the knowledge base beginning at the functional requirements in a recursive manner until no pending dependencies are left.

Requirement 6.2 [DEPENDENCY RESOLUTION]. *The algorithm should provide an automatic resolution of all (transitive) dependencies between functionality and service resource classes, starting from the top level functionalities defined in the functional requirements, ending when no more unresolved dependencies exist and resulting in a graph structure comprising all feasible configurations.*

During the recursive resolution of dependencies, compatibility between service resources needs to be checked in order to prevent unfeasible service configurations to be contained in the resulting graph structure.

Requirement 6.3 [COMPATIBILITY]. *Compatibility/interoperability between depending resources has to be validated and thus feasibility of potential configurations needs to be ensured at all times.*

To ensure practicability of the algorithm in real-life scenarios, we require determinism, termination and scalability as preventable non-determinism, interminableness or exponential time complexity would lead to suboptimal or unsatisfied consumer requests and thus should be avoided if possible.

Requirement 6.4 [DETERMINISM, TERMINATION & SCALABILITY]. *The algorithm should be deterministic, i.e. identical input will lead to identical output, terminating in finite time and scalable, i.e. having a polynomial growing time complexity with respect to the problem size.*[2]

The requirements above serve as design goal, for investigating on the applicability of related work and for evaluating the resulting algorithm qualitatively.

[1]A detailed solution on how to offer such a catalog through a graphical user interface to end consumers is not in the scope of this work.

[2]The problem size is mainly influenced by the consumer's functional requirements and the ontology's structure.

6.2 Related Work

For solving the technical problem of deriving configuration alternatives, a broad spectrum of research areas has to be covered, from Web service composition to techniques from operations research and constraint programming.

Berardi et al. [16] address the problem of automatic service composition by describing a service in terms of an execution tree, then making use of finite state machines to check for possible service compositions that match a requested behavior. Lécué and Léger [96] present an AI planning-oriented approach using Semantics. Based on causal link matrices, the algorithm calculates a regression-based optimal service chain. Both [16] and [96] concentrate on input/output based matching of Web services, thus they are not suitable for deriving mass customized Cloud services, where the interfaces between service resources are much more complex and cannot be described in terms of input and output. Another work from the Semantic Web context by Lamparter et al. [92] describes the matching process between requests and offers of Web services. This approach is related to the matching of compatibility constraints (relevant for Requirement 6.3), however it does not include a mechanism for resolving dependencies.

Blau et al. [25] propose an ontology-based tool for planning and pricing of service mash-ups. The tool can be used to compose complex Web services from a set of known atomic services, which are stored in a domain specific ontology. Thereafter, the complex service can be validated based on axioms and rules in the ontology. The complex service, however, has to be planned manually within the tool.

Sabin et al. [137] present different constraint programming approaches for product configuration. Van Hoeve [72] describes optimization approaches for constraint satisfaction problems. In [81] an optimization framework combining constraint programming with a description logic is provided. An ontology-based configuration approach as presented by Yang et al. [153] and Dong et al. [45] was already mentioned in Chapter 4. In both papers, an OWL ontology in conjunction with SWRL rules is used to formalize the constraints of a constraint satisfaction problem (CSP) [109]. Consumer requirements are added as additional constraints to the CSP. Feasible configurations are derived by solving the CSP through transforming the concepts modeled in OWL and the constraints formulated by means of SWRL into the JESS rule engine, a forward-chaining reasoning method implementing the Rete algorithm for processing rules. Common to all of these approaches is that the configuration problem itself is static with a well-defined set of configuration options which is known ex ante. This differs from the problem which is considered in this work, where configurations can be completely distinct from each other. Configuration options, thus, are only contained implicitly in the model and have to be made explicit for every individual consumer request.

Two software tools that include a transitive dependency management should also be mentioned, which are mainly related to Requirements 6.2 and 6.3. Advanced Packaging Tool (APT) [141] is a package management system to handle the installation and removal of software packages on Linux distributions. APT allows to automatically install further packages required by the desired software to avoid missing dependencies. The dependency management also includes compatibility checks, however only in form of a rather simple version level check. Another dependency manager is Apache Ivy.[3] Dependencies in Ivy are resolved transitively, i.e. you have to declare only direct dependencies, further dependencies of required resources are resolved automatically. Both approaches, however, do not allow any semantic annotation which would enable more complex dependencies or interoperability constraints.

6.3 Formal Model

In the following, formal representations of the functional requirements and the resulting service configuration graph are given.

6.3.1 Functional Requirements

Functional requirements describe *what* is needed, i.e. the resources required for a functioning service, whereas non-functional requirements describe *how* it is needed, i.e. the desired quality of the service. For the declarative description of functional requirements, we build on concepts taken from an ontology that contains background information in form of an abstract resource and functionality description layer (service ontology) and a domain-specific layer containing domain-dependent descriptions of available resources (domain ontology). Based on the concepts from the semantic service description framework described in Chapter 4, we define functional requirements as follows.

Definition 6.1 [Functional Requirements]. *For a background ontology O the functional requirements for a mass customized Cloud service are described by the set of concepts $\mathcal{C}$, with $\mathcal{C} = \{C_1, \ldots, C_n\}$ and $C_i \sqsubseteq$ Functionality, i.e. each C_i being a subclass of the concept Functionality in O.*

6.3.2 Service Configuration Graph

By resolving the dependencies for the set of functionalities defined in the consumer's functional requirements, additional knowledge cannot be gained. One can only make knowledge explicit, that is implicitly contained in the knowledge base. The nature of this knowledge is best described by graph structure, which is described in the following.

As the additional gained information is only valuable in the context of a certain consumer request, it would not be useful to materialize the entire domain ontology for each and every potential configuration that could be obtained by the entire set of potential service requests. It is, however, reasonable to also rely on an OWL ontology for persisting the consumer request specific gained knowledge, so that it can be used for further reasoning tasks. To do so, the result ontology presented in Chapter 4 makes use of the concepts *SourceNode*, *SinkNode*, *OrNode* and *Alternative*, all

[3] http://ant.apache.org/ivy/

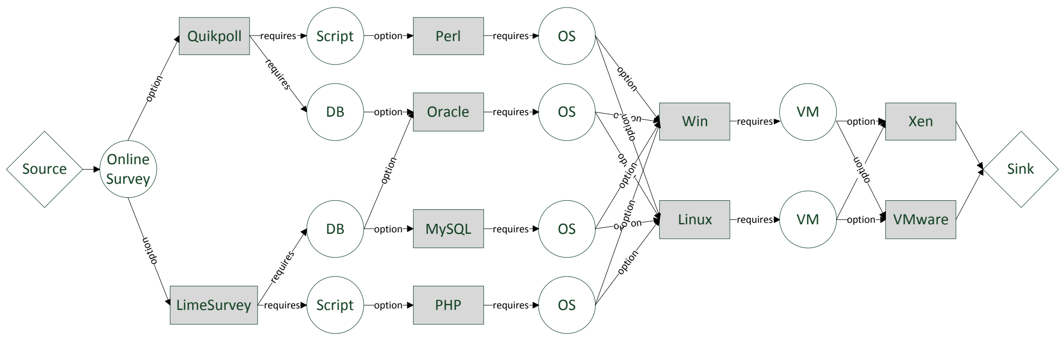


Figure 6.1: Example dependency graph

defined in the service ontology. *SourceNode* and *SinkNode* are a helper nodes to have a distinct starting and ending points in the graph. They correspond to the source and sink nodes in a network. The *OrNode* is introduced to capture the branching whenever there is more than one compatible resource instance that fulfills the dependency requirement.

In the remainder of this chapter we work with a more formal notation for the service configuration graph, as this notation is more appropriate in the context of the later presented pseudo algorithm.[4] A simple and arbitrary example graph is depicted in Figure 6.1.

Definition 6.2 [SERVICE CONFIGURATION GRAPH]. *For a background ontology O and functional requirements given as a set of concepts C, the* service configuration graph $\mathcal{G} = (V,E)$ *is a directed, acyclic, labeled graph with vertices V and edges E and a labeling function $\mathcal{L}$, recursively defined as follows:*

- *$n_0 \in V$ is the source node of $\mathcal{G}$*
- *$n_\infty \in V$ is the sink node of $\mathcal{G}$*
- *there is a node $n_C \in V$ with label $\mathcal{L}(n_C) = C$ for each (atomic or nominal) class $C \in \mathcal{C}$ and an edge $(n_0, n_C) \in E$*
- *if $n \in V$ is a node with an atomic class label $\mathcal{L}(n) = A$ then there is a node n_o with label $\mathcal{L}(n_o) = \{o\}$ for each individual $o \in O$ with $O \models A(o)$, and an edge $e = (n, n_o)$ with label $\mathcal{L}(e) =$ or*
- *if $n \in V$ is a node with a nominal class label $\mathcal{L}(n) = \{o\}$ then there is a node n_C with label $\mathcal{L}(n_C) = C$ for each atomic or nominal class C with $O \models C(x)$ for all x such that $O \models requires(o,x)$ and $O \models isCompatibleTo(o,x)$, and an edge $e = (n, n_C)$ with label $\mathcal{L}(e) =$ and*

6.4 Service Engineering Algorithm

OWL reasoners typically construct models for answering standard reasoning tasks, but do not expose them as such. Since it is needed to explicitly access these models

[4]Note that both notations are semantically the same.

as configuration alternatives at the instance-level, an algorithmic solution is chosen, making use of OWL reasoning capabilities in between the construction of dependency graphs.

Algorithm 6.1 initiateAlgorithm(C, O; $\mathcal{G}$) – Initiate the construction of a dependency graph.

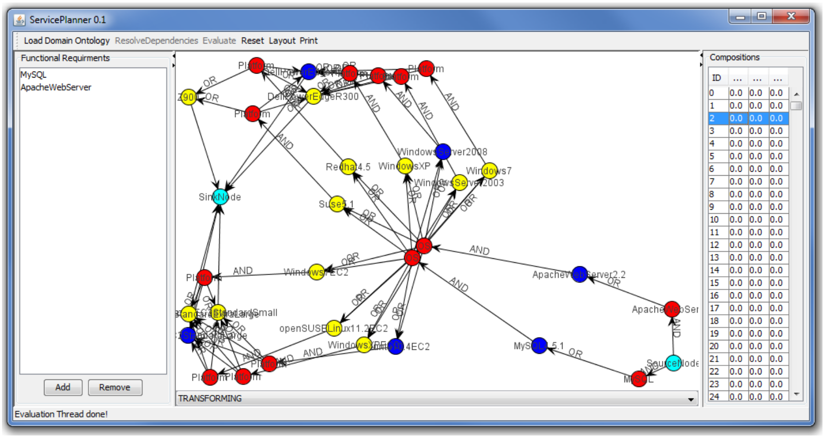
```
Require: a set of concepts $\mathcal{C}$ as functional requirements and ontology $O$
Ensure: $\mathcal{G}$ contains the service configuration graph for $\mathcal{C}$
  $V := \{n_0, n_\infty\}$, $V^* := E := \emptyset$
  for all $C \in \mathcal{C}$ do
    $V := V \cup \{n_C\}$, $\mathcal{L}(n_C) := C$
    $E := E \cup \{(n_0, n_C)\}$, $\mathcal{L}((n_0, n_C)) =$ and
    for all $o$ with $O \models C(o)$ do
      $V := V \cup \{n_o\}$, $\mathcal{L}(n_o) = \{o\}$
      $E := E \cup \{(n_C, n_o)\}$, $\mathcal{L}((n_C, n_o)) =$ or
      deriveServiceConfiguration($O$, $o$, $\mathcal{G}$, $V^*$)
    end for
  end for
```

Algorithm 6.2 deriveServiceConfiguration(O, o; $\mathcal{G}$, V^*) – Recursively construct a service configuration graph for a given ontology and resource instance.

```
Require: an ontology $O$ and a resource instance $o \in O$
Ensure: $\mathcal{G} = (V, E)$ contains a service configuration graph for $o$, $V^*$ contains all re-
    source instance nodes visited
  $V^* := V^* \cup \{n_o\}$
  $\mathcal{C} := \emptyset$, getRequiredClasses($o$; $\mathcal{C}$)
  if $\mathcal{C} = \emptyset$ then $E := E \cup \{(n_o, n_\infty)\}$
  for all $C \in \mathcal{C}$ do
    $V := V \cup \{n_C\}$, $\mathcal{L}(n_C) := C$
    $E := E \cup \{(n_o, n_C)\}$, $\mathcal{L}((n_o, n_C)) =$ and
    for all $o'$ with $O \models C(o')$ and $O \models isCompatibleTo(o, o')$ do
      $V := V \cup \{n_{o'}\}$, $\mathcal{L}(n_{o'}) = \{o'\}$
      $E := E \cup \{(n_C, n_{o'})\}$, $\mathcal{L}((n_C, n_{o'})) =$ or
      if $n_{o'} \notin V^*$ then deriveServiceConfiguration($O$, $o'$, $\mathcal{G}$, $V^*$)
    end for
  end for
```

The procedure initiateAlgorithm in Algorithm 6.1 initiates the construction of a service configuration graph, starting from functional requirements given in $\mathcal{C}$, and calls the procedure deriveServiceConfiguration in Algorithm 6.2, which recursively finds suitable service resource instances by following the object property *requires*. In the procedure getRequiredClasses the reasoning engine is invoked with the following SPARQL query to find all implicitly stated dependencies, i.e. through a class axiom rather than explicitly on instance level:

```
SELECT ?sub ?t ?obj
WHERE {
    ?sub owl:sameAs dm:component .
```



Functional Requirments

MySQL
ApacheWebServer

Compositions

ID	...	...	...
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	0.0	0.0	0.0
6	0.0	0.0	0.0
7	0.0	0.0	0.0
8	0.0	0.0	0.0
9	0.0	0.0	0.0
10	0.0	0.0	0.0
11	0.0	0.0	0.0
12	0.0	0.0	0.0
13	0.0	0.0	0.0
14	0.0	0.0	0.0
15	0.0	0.0	0.0
16	0.0	0.0	0.0
17	0.0	0.0	0.0
18	0.0	0.0	0.0
19	0.0	0.0	0.0
20	0.0	0.0	0.0
21	0.0	0.0	0.0
22	0.0	0.0	0.0
23	0.0	0.0	0.0
24	0.0	0.0	0.0

Figure 6.2: Screenshot of service engineering algorithm prototype

```
    ?sub so:requires _:b0 .
    _:b0 rdf:type ?t.
    ?obj rdf:type ?t }
ORDER BY ?t
```

so hereby refers to the name space of the service ontology, *dm* to the name space of the domain ontology. The literal *_:b0* refers to a blank node, i.e. there do not exist two individuals for which we find the *requires* property. Yet, based on the axiomatic knowledge, we know there has to be at least one.

The query will be answered with a set of all types characterizing the blank node. This results in one disadvantage: We are only interested in the most specific class assertions. If the knowledge base for example contained the information that each and every application needs an operating system, the result set of the query would contain the class *OS*, however also every superclass up to *Thing*. Therefore, in a second step, one needs to find out the most specific classes, i.e. all classes that have no subclasses also contained in the result set. This can be achieved by a simple algorithm which has a worst case runtime of $\mathcal{O}(\pi^2)$ subsumption checks, with π the number of classes in the class hierarchy.

As there might be redundant dependencies, which by themselves again might have further dependencies, visited resources (V^*) are memorized, as their dependencies do not need to be resolved more than once. Further the algorithm remembers unfulfilled requirements and recursively traces them back, deleting unfeasible paths. For are better comprehensibility these steps are not included in the above printed pseudo algorithm, as they would only confuse the reader.

6.5 Implementation

As proof-of-concept, a prototype was implemented in Java. As reasoning engine Pellet [142] was chosen, as to the best knowledge of the author, it is the only OWL DL reasoner that is capable of reasoning with both SWRL rules and SPARQL queries that involve blank nodes. A screenshot of the prototype application is depicted in Figure 6.2.

The ontology update mechanism (cf. Chapter 5) was integrated by means of OWL DB (trough relying on the OWL API). This approach allows to rely on domain knowledge which is kept up-to-date automatically. Further, the optimization problem following in Chapter 8 has been implemented as binary integer program using CPLEX and was successfully integrated into the prototype.[5]

6.6 Evaluation

In the following, the presented solution is thoroughly evaluated against the requirements posed upon the service engineering algorithm.

Ontology-Based Knowledge Base. The service engineering algorithm, which was presented prior in this chapter, derives feasible service configurations based on a set of concepts C serving as functional requirements of a consumer request. The concepts in C are taken from an ontology which is built upon the concepts of the generic service ontology presented in Chapter 4. The algorithm then reveals knowledge that is implicitly contained in the ontology and only valuable for a certain consumer request by using the information modeled on an abstract level using object properties *requires* and *isCompatibleTo*. Thus, an algorithm was presented that derives service configurations by relying on an ontology approach for knowledge representation.

Dependency Resolution. Two procedures were presented as core contribution of this chapter. The first procedure initiateAlgorithm initiates the algorithm and starts resolving dependencies beginning at the concepts in C, which represent the functional requirements given by a service consumer. The second procedure deriveServiceConfiguration then recursively resolves all dependencies, until no furhter dependencies are left. Unfulfilled dependencies are traced back and unfeasible configurations thus removed from the resulting service configuration graph. Hence, the requirement of dependency resolution is met.

Compatibility. The recursive procedure deriveServiceConfiguration only adds service resource instances from the ontology, which are considered compatible through the object property *isCompatibleTo*. The semantic service description framework hereby allows complex SWRL rules for defining compatibility, which can also make use of additional object or datatype properties associated with a resource instance. This way, compatibility among dependent service resources is ensured throughout the entire service configuration graph.

Determinism, Termination & Scalability. The algorithm contains no stochastic conditional branches, therefore given the same input C and O, the algorithm always derives the same resulting service configuration graph $\mathcal{G}$ in a deterministic manner. Loops are successfully prevented by remembering already visited nodes in V^*, thus the algorithm is guaranteed to terminate.

Lastly, for evaluating the scalability of the presented approach, the time complexity of the algorithm needs to be investigated on. With α denoting the number of classes in C, β the maximum number of instances in any resource class,

[5]A simplified version of the prototype, without OWL DB and CPLEX integration can be executed using Java Web Start at `http://research.steffenhaak.de/ServicePlanner/`.

Requirement	Berardi	Lécué	Lamparter	Blau	Sabin	Yang / Dong	APT / IVY	This Work
6.1 Ontology-Based KB	○	○	●	●	○	●	○	●
6.2 Dependency Resolution	◐	○	○	○	○	○	●	●
6.3 Compatibility	●	◐	●	●	●	●	●	●
6.4 Determinism, Termination & Scalability	●	●	●	○	●	●	●	●

Table 6.1: Review on requirements

γ the maximum number of dependencies of any resource class, π the maximum number of classes in any class hierarchy of a required resource class and λ the maximum dependency depth, the algorithm has a worst case time complexity of $\mathcal{O}\left(\alpha \cdot \beta^{\lambda+1} \cdot \pi^2 \cdot \gamma^\lambda\right)$. We find an exponential time complexity in the maximum dependency depth λ, which, however, can be assumed constant and not growing with the size of an ontology. With respect to all other metrics, the algorithm shows a polynomial worst case time complexity. In addition, the expected average time complexity presumably will be a lot better, since the above calculation is based on maximum values and ignores the fact, that not all resource instances contained in a class are always compatible.

Summarizing Review. The considerations from above are summarized in Table 6.1, along with an overview on the applicability of the presented related work.

6.7 Summary

After presenting requirements and related work, an algorithm for deriving potential configurations for a mass customized Cloud service was introduced in this chapter. The algorithm takes a domain ontology (making use of the semantic service description framework from Chapter 4) and a set of functionalities as input. It then derives a service configuration graph by resolving all dependencies stored in the ontology for the selected functionalities in a recursive and compatibility ensuring manner, making implicitly contained knowledge explicit in form of a graph structure. The algorithm is deterministic, terminating and scalable – showing polynomial time complexity with an increasing ontology size. Lastly, the applicability of related work and an overview on the the qualitative evaluation was given.

Part III

Economic Design, Implementation and Evaluation

Chapter 7

Multi-Attributive Negotiations

This thesis is set in the domain of mass customized Cloud services. Within this context, provider and consumer need to agree on both functional and non-functional properties for the envisioned service offer. The negotiation process concerning the functional part thereby follows a binary characteristic, i.e. functionality either sufficiently matches the requirements or not. In Part II, an extensive framework for deriving feasible service configurations matching the functional requirements of a consumer request was presented. Dealing with the non-functional requirements like QoS and price brings along further challenges for this process.

In this chapter, a theoretical model is introduced in order to evaluate three mechanisms for bilateral negotiation on multiple attributes with respect to their economic properties. Two of the mechanisms are stylized representations of existing mechanisms, one – DISCOUNTBIDDING – is newly introduced. In the light of the impossibility theorems by Hurwicz and Myerson and Satterthwaite, the requirements of individual rationality and budget balance are maintained due to their practical importance, incentive compatibility is abandoned, and the effect of strategic bidding on efficiency and distribution of economic surplus is studied in different settings. For an introduction on these theorems and mechanism design in general refer to Section 2.3.2.

To the end of this chapter, Section 7.1 redefines the broad scenario presented in Chapter 3 to form a more detailed negotiation scenario. Related work is reviewed in Section 7.2, followed by a description of the used methodology in Section 7.3. In Section 7.4, the economic model for describing the preferences, utility functions and welfare is presented. Section 7.5 formalizes the negotiation mechanisms, Sections 7.6 and 7.7 analyze strategic behavior and outcomes in settings under complete and incomplete information with different levels of risk and risk aversion. Based on these findings, a comparison of the mechanisms in both settings follows in Section 7.8. Finally, Section 7.9 concludes the findings, sketches their limitations and describes the implications for the subsequent chapter on service optimization and for the overall scenario of this thesis. The results of this chapter are largely based on [64].

7.1 Scenario

As stated before, in the scenario of mass customizing Cloud services, the offer creation process requires provider and consumer to negotiate over functionality, quality and price of a service. Regarding the quality, there is an almost infinite space of options depending on the complexity of the configuration, from different levels of response time or instructions per second to scalability of the service, availability levels, and the like. The consumer may have an optimal quality set in mind which would perfectly fulfill her needs along multiple quality dimensions, paired with a maximum willingness to pay. Offerings that yield slightly lower quality may still be good enough, yet come along with a decreased willingness to pay. In other words, it appears realistic to set some upper and lower quality boundaries to properly define a consumer request. Receiving a quality higher than the upper boundary does not increase a consumer's utility any further while the lower boundary marks the tipping point at which offerings are no more acceptable for the consumer and, thus, drops its utility to zero.[1] The utility from services in between these boundaries at a given price can be expressed in a scoring rule or function, as it is commonly used in multi-dimensional auctions (e.g. [37, 33, 21, 127, 7]).

Transferring the general setting of this thesis into a negotiation scenario for mass customized services, the challenge for service providers is to decide which negotiation mechanism to use for defining an agreement on quality and price, as described above. A multi-dimensional negotiation between consumer and provider typically includes an *integrative* and a *distributive* element [149, 133]. The integrative element shall identify the optimal service level that maximizes the difference from consumer utility and provider costs. In other words and when expressing consumer utility in monetary terms, it specifies the value or economic surplus created in the negotiation. The distributive element determines how the economic surplus created in the integrative part is actually distributed between the negotiating parties. Thus, in this part, the market participants claim their stake in the created value. Negotiators acting strategically typically address integrative and distributive elements in parallel, thereby limiting their own ability to mutually maximize the economic surplus.

A negotiation mechanism or, more general, a market mechanism is typically judged with respect to four desirable economic desiderata (cf. Section 2.3).

Individual rationality ensures that participants do not expect to incur losses from the mechanism and, thus, do not have to consider opting out from participation in the negotiation.

Incentive compatibility lets participants honestly report their true type, i.e. true preferences, as a (Bayesian) equilibrium strategy.

(Ex-post) allocation efficiency requires that whenever a configuration with a certain service level exists for which the consumer's valuation is higher than the provider's costs, an agreement is reached and the agreement maximizes the sum over both consumer's and provider's utility.

[1]While there might be even slight changes in utility above/below these boundaries, the assumption of hard borders simplifies the model, is common in literature and is believed to have little influence on practical implications.

Budget balance denotes that no outside payments are required to realize the outcome.[2]

Yet, the economic outcome of such a setting is restricted by a multitude of strong theoretic results: Given quasi-linear preferences, it is impossible to design a Bayesian-Nash incentive compatible mechanism that achieves individual rationality, efficiency, and budget balance at once (cf. Section 2.3.2.3)[3]. However, incentive compatibility is the prerequisite for an efficient outcome. Hence, the named characteristics need to be balanced in some way. For instance, efficient mechanisms can lead to a considerable need to plough in money, as the famous Vickrey-Clarke-Groves (VCG) mechanisms give proof of [148, 39, 58]. Likewise, individually rational and budget balanced mechanisms can result in highly inefficient outcomes [105, 10]. Additionally, budget balance and individual rationality are compulsory characteristics to enable sustainability and implementability over time [103, 126]. On the one hand, participants are unwilling to voluntarily participate in a market mechanism in which they expect to incur losses. On the other hand, continuous subsidiaries are unrealistic, if there is no third party to externally subsidize the mechanism.

In order to evaluate the performance of different negotiation mechanisms, a market scenario being an economic abstraction of the scenario presented in Section 3.1 is consulted. It has the following assumptions and characteristics:

- Multiple consumers are interested to procure a mass customized service of a particular functionality. Besides the service's functionality and price, quality attributes play a major role.
- The consumers are endowed with heterogeneous requirements and a different willingness to pay for variable QoS levels. The consumers' individual willingness to pay subject to a certain quality level is not fully known to the public.
- Multiple providers are present in the market. They offer mass customized services in accordance with the solutions presented prior within this thesis. Providers and consumers agree on the functionality of a service by having the consumer select the functionalities of her desire from a catalog offered by the provider. The potential configurations for the selected functionalities differ in their quality of service and prices, subject to negotiation.
- The providers have different technologies and internal processes that influence their internal costs for service provision. These factors and costs are not fully known to the public.
- Both consumers and providers act rationally and strategically in their own interest.

[2]This definition describes weak budget balance. Strong budget balance requires the sum of all net transfers to be zero, cf. Section 2.3.2.1.

[3]The result cited above is derived in Theorem 1 of [121]. Given this impossibility, their Theorem 2, provides the grounds for testing whether a given incentive-compatible, individually rational mechanism maximizes the expected economic surplus. This theorem does not apply to the present scenario, as its robustness depends on knowledge of both distributions of player types, it is restricted to single-attribute negotiations (price only) and a trusted 3rd party is required for its implementation.

- Since both consumers' and providers' types are not publicly known, the market exhibits uncertainty of the optimal matching. Yet, both consumers and providers may have some information or expectation on the other market side's type which is reflected in their strategies.
- Consumers and providers interact bilaterally. There is no third party running a central exchange.
- To facilitate online scenarios with low transactions costs, interaction is restricted to two-stage mechanisms, in preference to more complicated negotiation mechanisms. This simplification follows the standard assumption in bargaining models that delay of reaching an agreement is costly and, thus, rational agents should reach an agreement immediately with the first offers (cf. the seminal paper by Rubinstein [135] and the literature building on it).

In the shade of this game theoretic model, analytical and numerical results can be derived. These results would not be feasible in the broad scenario which is much more complex on a technical level. Nevertheless, the insights gained can be transferred to the practical scenario and the optimization approach presented in the subsequent chapter.

7.2 Related Work

There are several market mechanisms to be used in such a scenario. The simplest and most widely-used mechanism is a *fixed price* for a specifically defined service without any further information exchange. Such a fixed price can either be set by the provider or the consumer. In both cases, the bidding language is very simple, yet very little information is transferred hindering the discovery of the optimal service specification given the provider's and consumer's types. In game theoretic terms, such a take-it-or-leave-it fixed price offer is an ultimatum game [61], the simplest form of a negotiation mechanism. A subtle variation commonly observed in practice is to not offer a single quality-price combination, but a small set of such combinations to choose from. This increases the likelihood of discovering the optimal service configuration at the cost of complexity of decision making for both sides.

Recently, a variation of this simple mechanism has gained importance. Using *opaque selling*, the provider or typically an intermediary guarantees certain quality attributes and the price of a custom service, yet hides its very identity until the transaction is completed. Despite the fact, that existing offerings for travel deals like *Priceline* or *Hotwire* also match consumer requirements with different quality-price-offers, *opaque selling* is an additional distribution channel for already existing take-it-or-leave-it offers, with a very limited quality-spectrum. Literature in this area [80, 47] has a strong revenue management focus, i.e. capacity management and dynamic pricing, with quality of service and customization not being in the center of attention.

More complex mechanisms to tackle the multi-dimensionality at hand are multi-attribute negotiations and auctions. Multi-attributivity, as introduced to auction design by [37] and [33], allows for the negotiation on non-price attributes by referring to multiple features of a single unit [150]. In the last decade, powerful computer networks have brought up electronic markets that can handle the complex sale or procurement of multi-attribute services and products through automated negotiation and the determination of the allocation [53, 17, 36, 20]. Such mechanisms' bidding languages are rich and, therefore, a lot of information is exchanged. Multi-attribute procurement auctions based on the family of VCG mechanism are incentive compatible and allocatively efficient, yet suffer from their impossibility to achieve budget balance. Approaches to achieve budget balance by foregoing efficiency mostly result in very complex mechanisms, either with respect to their accomplishment or, due to complex transfer functions, with respect to the strategies that can be played. For instance, [127] introduces an iterative protocol while [128], [90], and [27] propose budget balanced approximations of VCG mechanisms, at the cost of sophisticated transfer functions which lead to very complex strategy considerations. [40] introduced a multi-attribute mechanism based on cooperative game theory with some interesting properties in networked environments, however, also leading to highly complex strategies of the participants.

In summary, common mechanisms in the scenario sketched above either reveal very little information, or their bidding language and/or transfer function is complex. The information that is exchanged is oftentimes biased by strategic behavior of the participants. This circumstance suggests that there is a trade-off between the simplicity of a mechanism and its richness (which is linked to the information content exchanged between the negotiating parties). Moreover, strategic misrepresentation of the participants' preferences can be observed due to the mixture of the integrative and the distributive part of the negotiations.

7.3 Methodology

To facilitate mass customization of individualized and quality-differentiated services, a mechanism is sought that fulfills individual rationality and budget balance and, at the same time, keeps efficiency as well as truthful information content high, while maintaining a simple bidding language. In order to evaluate how different mechanisms perform in the present setting, we scrutinize the following three mechanisms:

TupleBidding One party poses a take-it-or-leave-it offer of a single price-quality tuple, the other party either accepts or not.

ScoringBidding One party proposes a complete scoring function over the set of possible price-quality combinations. The other party either selects one tuple described by this function or rejects to agree at all.

DiscountBidding Like the scoring function mechanism, but in addition to the scoring function, the proposing party requests a price discount it requires on any tuple described by the scoring function.

TupleBidding and ScoringBidding represent extreme cases on the continuum of simple and rich information exchange – they are stylized representations of commonly used fixed price and multi-attribute mechanisms. Intermediate versions like proposing multiple price-quality tuples and extensions like repeated offer exchanges are possible, but offer only limited insight in qualitative differences of the mechanisms.

DISCOUNTBIDDING is newly introduced as an extension of SCORINGBIDDING. The basic idea is to separate the integrative and distributive element of the negotiation with the scoring function allowing to identify the optimal price-quality combination and the discount factor allowing to claim value. All three mechanisms are formally characterized and evaluated – both under complete and incomplete information.

For the case of *complete information*, i.e. the consumer is completely and correctly aware of the providers true type, both strategies and the resulting equilibria are deduced analytically.

For the case of *incomplete information*, a parameter capturing the consumer's risk is introduced. A numerical simulation study is conducted for computing strategies and equilibria. A quasi-Monte Carlo simulation is chosen as deterministic method for approximating the expected payoffs for a given bidding parameters. The bidding parameters thereby are uniformly sampled within their value domain. The bidding value offering the highest expected consumer payoff is recognized as equilibrium strategy along with the corresponding provider payoff and the obtained welfare for this equilibrium. A more detailed description of the numerical simulation follows in Section 7.7.2.

7.4 Model of Preferences

Consider two parties, the service provider $\mathcal{P}$ and the consumer $\mathcal{C}$, negotiating over quality $q \in (0,1)$ and corresponding price $p \in (0,1)$ of a service. Both quality and price are individually normalized to the unit interval. The provider $\mathcal{P}$ has a cost function

$$C(q) = q^b \tag{7.1}$$

with $b \in [1,\infty)$ representing the cost of providing a service of given quality. $b = 1$ yields a linear, $b > 1$ a convex cost function with positive and increasing marginal costs of quality. The provider $\mathcal{P}$ has the quasi-linear utility function

$$U_{\mathcal{P}}(q) = p - C(q) \tag{7.2}$$

Analogously, the consumer $\mathcal{C}$ has a scoring function

$$S(q) = q^a \tag{7.3}$$

with $a \in (0,b)$ over different qualities of service.

The structural form using a monomial to capture the cost and the scoring functions is chosen for three reasons: (1) It is a common approach in the related literature to capture each quality-attribute with one monomial, e.g. [21, 27]. (2) It allows for both simple and comprehensible analytical considerations along with computational tractability and (3) it ensures proximity to realistic scenarios under consideration of the preference elicitation challenges.

$a > 0$ ensures strong monotonicity in quality, i.e. $\mathcal{C}$ prefers a higher quality over a lower quality, and $a < b$ ensures the existence of a mutually beneficial agreement. In practice and for a given provider and consumer it might be the case that no mutually beneficial agreement on a q-p-tuple exists. However, in this case no mechanism could yield an individually rational agreement. This case is neglected and we assume the existence of a mutually beneficial agreement for comparing the ability of different negotiation mechanisms in identifying the optimal agreement. $\mathcal{C}$ has the quasi-linear utility function

$$U_{\mathcal{C}}(q) = S(q) - p \tag{7.4}$$

The economic surplus – also termed as welfare – generated by an agreement is defined as sum of utilities

$$W(q) = U_{\mathcal{C}}(q) + U_{\mathcal{P}}(q) = S(q) - C(q) = q^a - q^b \tag{7.5}$$

Thus, welfare is determined by the quality that the two negotiating parties agree on, while it is independent of the price, as long as a price does not hinder the mutual agreement between the two negotiating parties. The price is merely the mechanism for distributing this welfare between the two parties. The optimal quality q^* maximizing welfare for given types of $\mathcal{P}$ and $\mathcal{C}$, is obtained by equating the first derivation of the welfare function with zero, solving for q and ensuring that the extreme value is a maximum:

$$q^* = \underset{q}{\operatorname{argmax}}\, W(q) = \left(\frac{b}{a}\right)^{\frac{1}{a-b}} \in (0,1) \tag{7.6}$$

The corresponding maximal welfare is:

$$w^* = \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{b}{a-b}} \tag{7.7}$$

Figure 7.1 sketches an example of the above model with $a = 0.5$, $b = 2$, and resulting $q^* = 0.397$. The price is arbitrarily chosen as $p = 0.35 \in [c(q^*), S(q^*)]$ and both parties' utilities are positive. Graphically speaking, the assumptions on $C(q)$ and $S(q)$ assure the existence of an area of individually rational potential agreements in between the two curves. The objective of a negotiation is to determine a point $\{q, p\}$ within this area. It is in both parties' interest[4] and Pareto optimal to choose $q = q^*$ so that the distance of the two curves, i.e. welfare, is maximized. The price p anywhere on the intersection of q^* with the area distributes this welfare between both parties. For an omniscient arbitrator with a given fairness perception, this is a relatively easy task. For $\mathcal{P}$ and $\mathcal{C}$ themselves under incomplete information and both acting strategically, this task, however, is very complex.

[4]Only from a global perspective. Depending on the value distribution, i.e. the agreed price, either party can have an interest to choose a different quality at a more favorable price.

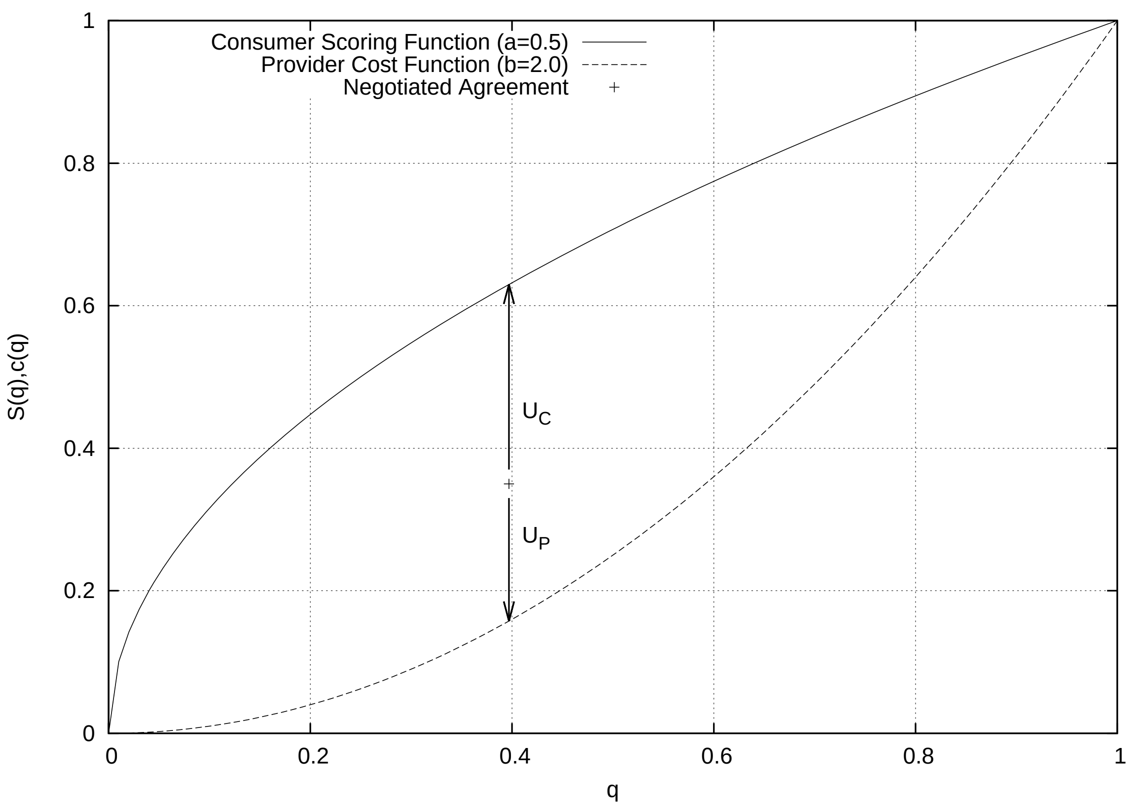


Figure 7.1: Example for scoring and cost functions, optimal quality, price and utilities

7.5 Negotiation Mechanisms

Three negotiation mechanisms can help $\mathcal{P}$ and $\mathcal{C}$ in their complex task. Without loss of generality, the consumer $\mathcal{C}$ is assumed going first and the provider $\mathcal{P}$ responding to the offer. As the scenario and model are symmetric with respect to the roles, the evaluation with an inverted order of action would follow analogously.

7.5.1 TupleBidding

This mechanism represents the classical take-it-or-leave-it business, as one party proposes a quality-price pair, the other party accepts or declines. Formally speaking:

1. $\mathcal{C}$ submits a binding bid of a tuple $\{q, p\}$.
2. $\mathcal{P}$ can accept exactly this tuple as agreement or reject it leading to no agreement. Assuming myopic utility maximization within the scope of this negotiation, $\mathcal{P}$ accepts if $p \geq C(q)$ and rejects otherwise.[5]

This mechanism is chosen for comparison for two reasons: (1) It offers a maximum simplicity in its bidding language and (2) it is commonly found in practice. Providers offer their product of a certain quality for a given price. It is then up to the consumer, whether she wants to accept the deal or not.

[5]For simplicity of the following analysis acceptance is assumed in case of indifference, i.e. for $p = C(q)$.

7.5.2 ScoringBidding

ScoringBidding represents the extreme case of rich information exchange. The consumer potentially could reveal complete information about her true type, while this certainly is not in her own interest. More formally, this mechanism consists of the following two steps:

1. $\mathcal{C}$ submits a binding bid of a scoring function $\hat{S}(q)$ or – in a less generic setting – the parameters of a scoring function of given functional form, e.g. $\hat{a}$ for $\hat{S}(q) = q^{\hat{a}}$. $\mathcal{C}$ can (and will) choose $\hat{S}(q) \neq S(q)$, i.e. she can strategically misrepresent her type.
2. $\mathcal{P}$ sets a quality $\hat{q}$ and price $\hat{p} \leq \hat{S}(\hat{q})$. $\{\hat{q}, \hat{p}\}$ is the agreement. If there exists a $\hat{q}$ with $\hat{S}(\hat{q}) \geq C(\hat{q})$, $\mathcal{P}$ maximizes his utility with $\hat{p} = \hat{S}(\hat{q})$ and $\hat{q} = \text{argmax}_q\, \hat{p} - C(q) = \text{argmax}_q\, \hat{S}(q) - C(q)$. Otherwise, $\mathcal{P}$ rejects any agreement.

As stated before, this mechanism is designed following the multi-attributive auction theory [18, 21, 27, 7] and was chosen as benchmark for rich information exchange mechanisms.

7.5.3 DiscountBidding

This mechanism is similar to ScoringBidding in its amount of information exchanged. However, DiscountBidding allows the consumer to claim an additional price discount, i.e. a price reduction over her scoring value for the chosen quality. Formally, one denotes:

1. $\mathcal{C}$ submits a binding bid of a scoring function $\hat{S}(q)$ or, as above, a single parameter, e.g. $\hat{a}$. In addition, $\mathcal{C}$ submits a discount value d.
2. $\mathcal{P}$ sets a quality $\hat{q}$ and price $\hat{p} \leq \hat{S}(\hat{q}) - d$. $\{\hat{q}, \hat{p}\}$ is the agreement. If there exists a $\hat{q}$ with $\hat{S}(\hat{q}) - d \geq C(\hat{q})$, $\mathcal{P}$ maximizes his utility with $\hat{p} = \hat{S}(\hat{q}) - d$ and $\hat{q} = \text{argmax}_q\, \hat{p} - C(q) = \text{argmax}_q\, \hat{S}(q) - d - C(q)$. Otherwise, $\mathcal{P}$ rejects any agreement.

DiscountBidding is an extension to ScoringBidding and newly introduced in this thesis. It allows the consumer to claim value independent of the chosen quality.

7.6 Evaluation under Complete Information

In this section optimal bidding behavior, utility, and welfare is derived for both $\mathcal{C}$ and $\mathcal{P}$ analytically. $\mathcal{C}$ is assumed to be perfectly informed, i.e. $\mathcal{C}$ accurately knows $\mathcal{P}$'s cost function $C(q)$ and $\mathcal{P}$'s profit maximizing price-quality decision function. Section 7.7 will relax this complete information setting and allow for risk and risk aversion.

7.6.1 TupleBidding

$\mathcal{C}$ bids a tuple $\{q,p\}$ which maximizes her utility, i.e.

$$\max_{q,p} U_{\mathcal{C}} = \max_{q,p} S(q) - p \tag{7.8}$$

making sure that $\mathcal{P}$ accepts, i.e. $p \geq C(q)$. For no combination $\{q,p\}$ there is a benefit for $\mathcal{C}$ of setting $p > C(q)$, so $\mathcal{C}$ will set $p = C(q)$ while finding the efficient and, thus, welfare maximizing quality

$$q^* = \operatorname*{argmax}_{q} S(q) - C(q) \tag{7.9}$$

and claiming the entire rent.[6] Thus $\mathcal{C}$ maximizes her utility by bidding

$$q^* = \left(\frac{b}{a}\right)^{\frac{1}{a-b}} \tag{7.10}$$

(cf. Equation 7.6) and $p = C(q^*)$. $\mathcal{P}$ will accept this bid, as the price is equal to his costs at q^*, i.e. $p \geq C(q^*)$, resulting in the following utilities and welfare.

Consumer Utility $U_{\mathcal{C}}$	*Provider Utility* $U_{\mathcal{P}}$	*Welfare* W
$U_{\mathcal{C}}(q^*) = S(q^*) - p$ $= (q^*)^a - (q^*)^b$ $= \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{b}{a-b}} = w^*$	$U_{\mathcal{P}}(q^*) = p - C(q^*) = 0$	$W(q^*) = U_{\mathcal{C}}(q^*) = w^*$

7.6.2 ScoringBidding

$\mathcal{C}$ wants to sumbit a bid $\hat{S}(q) = q^{\hat{a}}$ that maximizes her utility $U_{\mathcal{C}}$. She will not submit a bid of $\hat{a} > b$, as this will inevitably lead to no agreement. $\hat{a} = b$ leaves the provider indifferent in the quality he chooses, as any quality yields the same (zero) utility to him. This is a major risk to the consumer. Thus, and as it is a theoretic artifact of complete information and the assumptions made on the scoring and cost function, the case of $\hat{a} = b$ is excluded and $\hat{a} < b$ is assumed. The following analysis includes $\hat{a}$ approaching the limit b, i.e. the consumer bidding infinitely close to the provider's cost function. $\mathcal{P}$'s response to $\mathcal{C}$'s bid of $\hat{a}$ will be[7]

$$\hat{q} = \operatorname*{argmax}_{q} \hat{S}(q) - C(q) = \left(\frac{b}{\hat{a}}\right)^{\frac{1}{\hat{a}-b}} \tag{7.11}$$

In order to maximize her utility, $\mathcal{C}$ has to solve the optimization

$$\hat{a}^* = \operatorname*{argmax}_{\hat{a}} \; U_{\mathcal{C}} = \operatorname*{argmax}_{\hat{a}} \; S(\hat{q}) - p \tag{7.12}$$

[6]In Figure 7.1, by bidding tuple $\{q^*, C(q^*)\}$, $U_{\mathcal{C}}$ (upper arrow) is at the maximum, while $U_{\mathcal{P}}$ (lower arrow) is zero.

[7]Analogously to the calculation of q^* in Equation (7.6) in Section 7.4.

As in this mechanism $\mathcal{P}$ will set the price $p = \hat{S}(q)$, $\mathcal{C}$'s optimization problem is

$$\hat{a}^* = \underset{\hat{a}}{\operatorname{argmax}}\, S(\hat{q}) - \hat{S}(\hat{q}) \tag{7.13}$$

As $\mathcal{P}$'s response $\hat{q}$ depends on $\hat{a}$, one obtains the optimization problem[8]

$$\hat{a}^* = \underset{\hat{a}}{\operatorname{argmax}}\, S(\hat{q}(\hat{a})) - \hat{S}(\hat{q}(\hat{a})) \tag{7.14}$$

$$= \underset{\hat{a}}{\operatorname{argmax}} \left(\frac{b}{\hat{a}}\right)^{\frac{a}{\hat{a}-b}} - \left(\frac{b}{\hat{a}}\right)^{\frac{\hat{a}}{\hat{a}-b}} \tag{7.15}$$

Intuitively, $\mathcal{C}$'s utility will increase in $\hat{a} > a$. In fact, one is confronted with a corner solution. Following, an analytical proof is sketched, showing that $U_{\mathcal{C}}$ is at its maximum for $\hat{a} \to b$. One has to distinguish between three cases:

1. $\hat{a} < a$: bidding a value $\hat{a} < a$ inevitably leads to a negative utility, as
$$p = \hat{S}(q) > S(q), \forall q \in (0,1)$$
2. $\hat{a} = a$: truthful bidding leads to a utility of zero for $\mathcal{C}$, i.e.
$$U_{\mathcal{C}} = \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{a}{a-b}} = 0$$
3. $b > \hat{a} > a$: $U_{\mathcal{C}}(\hat{q}(\hat{a})) = S(\hat{q}(\hat{a})) - \hat{S}(\hat{q}(\hat{a}))$ increases strictly monotonically in $\hat{a}$.[9] Thus, $\mathcal{C}$'s optimization leads to $\hat{a} \to b$.

As shown, one has to regard the corner solution for $y \to 1$ or $\hat{a} \to b$ or $\hat{a} = b - \epsilon$ with $\epsilon \to 0$. By regarding the corresponding limits for $\epsilon \to 0$, the welfare as well as provider and consumer utilities can be derived.

Consumer Utility $U_{\mathcal{C}}$	*Provider Utility $U_{\mathcal{P}}$*	*Welfare W*
$U_{\mathcal{C}}(\hat{q})$ $= \lim_{\epsilon \to 0} \frac{b}{b-\epsilon}^{\frac{a}{-\epsilon}} - \frac{b}{b-\epsilon}^{\frac{b-\epsilon}{-\epsilon}}$ $= e^{-\frac{a}{b}} - \frac{1}{e}$	$U_{\mathcal{P}}(\hat{q})$ $= \lim_{\epsilon \to 0} \frac{b}{b-\epsilon}^{\frac{b-\epsilon}{-\epsilon}} - \frac{b}{b-\epsilon}^{\frac{b}{-\epsilon}}$ $= \frac{1}{e} - \frac{1}{e} = 0$	$W(\hat{q}) = U_{\mathcal{C}}(\hat{q}) = e^{-\frac{a}{b}} - \frac{1}{e}$

7.6.3 DISCOUNTBIDDING

$\mathcal{P}$ only accepts, if he obtains a utility of $U_{\mathcal{P}} \geq 0$, i.e. the agreed price must be greater or equal to the costs for the agreed quality:

$$p \geq C(\hat{q}) \Leftrightarrow \hat{q}^{\hat{a}} - d \geq \hat{q}^{b} \tag{7.16}$$

[8] The notation $\hat{q}(\hat{a})$ is used to indicate this relationship of $\hat{q}$ and $\hat{a}$, without explicitly defining a function.

[9] Cf. proof in Appendix B.1.

$\mathcal{C}$ can therefore at most obtain a utility as high as the welfare of a chosen quality, $U_\mathcal{C} = W(\hat{q})$. The optimal strategy for $\mathcal{C}$, thus, is to achieve $U_\mathcal{C} = w^*$, which implies $U_\mathcal{P} = 0$. There only exists one welfare maximizing quality $\hat{q}^*$ such that $W(\hat{q}^*) = w^*$. Hence, $\mathcal{C}$ can only obtain $U_\mathcal{C} = w^*$, if and only if $\hat{q}$ is equal to $\hat{q}^*$. As $\mathcal{P}$'s optimal response is

$$\hat{q} = \underset{q}{\operatorname{argmax}}\, \hat{S}(q) - c(q) = \underset{q}{\operatorname{argmax}}\, q^{\hat{a}} - q^b \tag{7.17}$$

and

$$\hat{q}^* = \underset{q}{\operatorname{argmax}}\, q^a - q^b \tag{7.18}$$

both terms must be equal, i.e.

$$q^{\hat{a}} - q^b \overset{!}{=} q^a - q^b \quad \Leftrightarrow \quad \hat{a} \overset{!}{=} a \quad \forall q \in (0,1) \tag{7.19}$$

In other words, no other $\hat{a}$ with $\hat{a} \neq a$ can lead to a welfare maximizing agreement. Bidding $\hat{a}$ truthfully is the only way to ensure a maximal welfare, which is quite intuitive.

Let us regard the second bidding parameter d. When bidding $\hat{a}$ truthfully, i.e. $S(q) = \hat{S}(q)$, the following holds:

$$U_\mathcal{C} = S(q) - p = S(q) - \hat{S}(q) + d = d \tag{7.20}$$

If $\mathcal{C}$ wants to exploit the full rent, $U_\mathcal{C} = w^*$, she has to bid $d = w^*$, thus optimal bidding for her consists of $\hat{a}^* = a$ and

$$d^* = w^* = \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{b}{a-b}} \tag{7.21}$$

if the bid is to be accepted by $\mathcal{P}$. $\mathcal{P}$ only accepts if

$$\hat{q}^{\hat{a}^*} - d^* \geq \hat{q}^b \tag{7.22}$$

which is no other than

$$(\hat{q}^*)^a - w^* \geq (\hat{q}^*)^b \Leftrightarrow (\hat{q}^*)^a - (\hat{q}^*)^b \geq w^* \Leftrightarrow w^* \geq w^* \tag{7.23}$$

which is always true, thus $\mathcal{P}$ will accept $\mathcal{C}$'s bid.

Consumer Utility $U_\mathcal{C}$	*Provider Utility $U_\mathcal{P}$*	*Welfare W*
$U_\mathcal{C}(q^*) = S(q^*) - p$ $= (q^*)^a - (q^*)^b$ $= \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{b}{a-b}} = w^*$	$U_\mathcal{P}(q^*) = p - C(q^*) = 0$	$W(q^*) = U_\mathcal{C}(q^*) = w^*$

7.7 Evaluation under Incomplete Information

This section deals with the more realistic case of incomplete information and risk aversion. Incomplete information implies risk in the decision making of $\mathcal{C}$. For the provider, there is no risk, as the negotiation mechanisms require him to react to $\mathcal{C}$'s bids, not her true type. When inverting the mechanisms towards the provider going first, he would face the risk. As mentioned before, the analysis is agnostic to the specific roles – all results equally apply the other way round, when the mechanisms are inverted.

7.7.1 Extended model under risk

$\mathcal{C}$ continues being informed about the state of the world and the rules of the game with one exception: the parameter b of $\mathcal{P}$'s cost curve. However, $\mathcal{C}$ knows that b is uniformly distributed and $\mathcal{C}$ knows the support, i.e. $b \sim U(m-s, m+s)$ with m and s being public knowledge. s is a proxy for $\mathcal{C}$'s risk; the higher s, the higher the risk. $\mathcal{P}$ still knows b, i.e. his cost function. To capture $\mathcal{C}$'s risk aversion, parameter r, the function

$$R(x) = x^{\frac{1}{1+r}} \tag{7.24}$$

and its corresponding inverse function

$$R^{-1}(x) = x^{1+r} \tag{7.25}$$

are introduced. $r = 0$ implies $\mathcal{C}$ being risk-neutral, while an increasing $r > 0$ implies $\mathcal{C}$ being increasingly risk averse. The risk aversion is reflected in $\mathcal{C}$'s utility function

$$U_{\mathcal{C}}(q) = R\left(S(q) - p(q)\right) = \left(S(q) - p(q)\right)^{\frac{1}{1+r}} \tag{7.26}$$

In order to compare the welfare in situations with different risk aversion and in order not to overvalue the utility of $\mathcal{C}$, the welfare function is adapted by back-transforming the influence of r[10]

$$W(q) = R^{-1}(U_{\mathcal{C}}(q)) + U_{\mathcal{P}}(q) = \left(S(q) - p(q)\right)^{1+r} + p(q) - c(q) \tag{7.27}$$

The case of complete information could be solved analytically (cf. Section 7.6); for incomplete information a numerical analysis is chosen. A wide range of parameters a, b, s, r is used as input to the simulation in order to test for sensitivities and assure reliability of the results. Three reasons indicate that the numerical solution is accurate: (1) For the border case of complete information, the numerical and the analytical solution coincide; (2) all strategic effects and utility comparisons vary either smoothly within the parameter range or, in case of step functions, are intuitive and straightforward to explain; (3) the qualitative effects and comparisons hold true for all parameter configurations tested.

[10]Any interpersonal utility comparison or definition of welfare is subject to normative assumptions on the relative importance of different players. While being aware of potential concerns, the $\mathcal{C}$'s expected utility is normalized to the unit interval by the use of R^{-1} and the welfare defined as sum of $\mathcal{P}$'s utility and $\mathcal{C}$'s normalized utility. This is the simplest and least questionable approach that serves our purpose of a numerical representation of welfare.

7.7.2 Numerical Simulation

The numerical simulation was performed with Java by enumerating all bidding parameters, the support of b's distribution and the risk parameter s with an increment of $\epsilon = 0.001$. The simulation was run for a wide range of different utility and cost parameters a and b and different risk aversion parameters r, varying from zero to four. For each parameter combination and each negotiation mechanism, the strategic interaction was played to identify optima in bidding strategies and the associated outcome. A sample algorithm in pseudo notation for SCORINGBIDDING is given in Algorithm 7.1. The algorithms for TUPLEBIDDING and DISCOUNTBIDDING follow analogously, as they only iterate over other bidding parameters.

Algorithm 7.1 Numerical Simulation Algorithm (SCORINGBIDDING)

```
Require: true scoring parameter $a$, cost parameter $m$, risk aversion $r$, increment $\epsilon$
Ensure: optimal bidding parameter $\hat{a}^*$, expected utilities $E(U_\mathcal{C})^*$ and $E(U_\mathcal{P})^*$ and
  welfare $W^*$ are obtained for each $s \in (0,1]$
  for $s = 0 \to 1$ do
    $s \leftarrow s + \epsilon$
    $\hat{a}^* \leftarrow 0, E(U_\mathcal{C})^* \leftarrow 0, E(U_\mathcal{P})^* \leftarrow 0, W^* \leftarrow 0$
    for $\hat{a} = a \to m + s$ do
      $\hat{a} \leftarrow \hat{a} + \epsilon$
      $E(U_\mathcal{C}) \leftarrow 0, E(U_\mathcal{P}) \leftarrow 0, W \leftarrow 0$
      for $b = m - s \to m + s$ do
        $b \leftarrow b + \epsilon$
        if $\hat{a} < b$ then
          $\hat{q} \leftarrow \frac{b}{\hat{a}}^{\frac{1}{\hat{a}-b}}$
          $\hat{p} \leftarrow \hat{q}^{\hat{a}}$
          $U_\mathcal{C} \leftarrow (\hat{q}^a - \hat{p})^{\frac{1}{1+r}}$
          $U_\mathcal{P} \leftarrow \hat{p} - \hat{q}^b$
          $E(U_\mathcal{C}) \leftarrow E(U_\mathcal{C}) + \frac{U_\mathcal{C}}{(2s/\epsilon)+1}$
          $E(U_\mathcal{P}) \leftarrow E(U_\mathcal{P}) + \frac{U_\mathcal{P}}{(2s/\epsilon)+1}$
          $W \leftarrow E(U_\mathcal{C})^{r+1} + E(U_\mathcal{P})$
        end if
      end for
      if $E(U_\mathcal{C}) > E(U_\mathcal{C})^*$ then
        $\hat{a}^* \leftarrow \hat{a}$
        $E(U_\mathcal{C})^* \leftarrow E(U_\mathcal{C})$
        $E(U_\mathcal{P})^* \leftarrow E(U_\mathcal{P})$
        $W^* \leftarrow W$
      end if
    end for
    StoreResult($s$, $\hat{a}^*$, $E(U_\mathcal{C})^*$, $E(U_\mathcal{P})^*$, $W^*$)
  end for
```

The graphs depicted in the following sections sketch selected results for one particular setting with $a = 0.5$, $b = 2$, $s \in [0,1]$, $r \in \{0,1\}$[11]. The slight zick-zack-pattern

[11]Further results for different parameters a and b can be found in Appendix B.3.

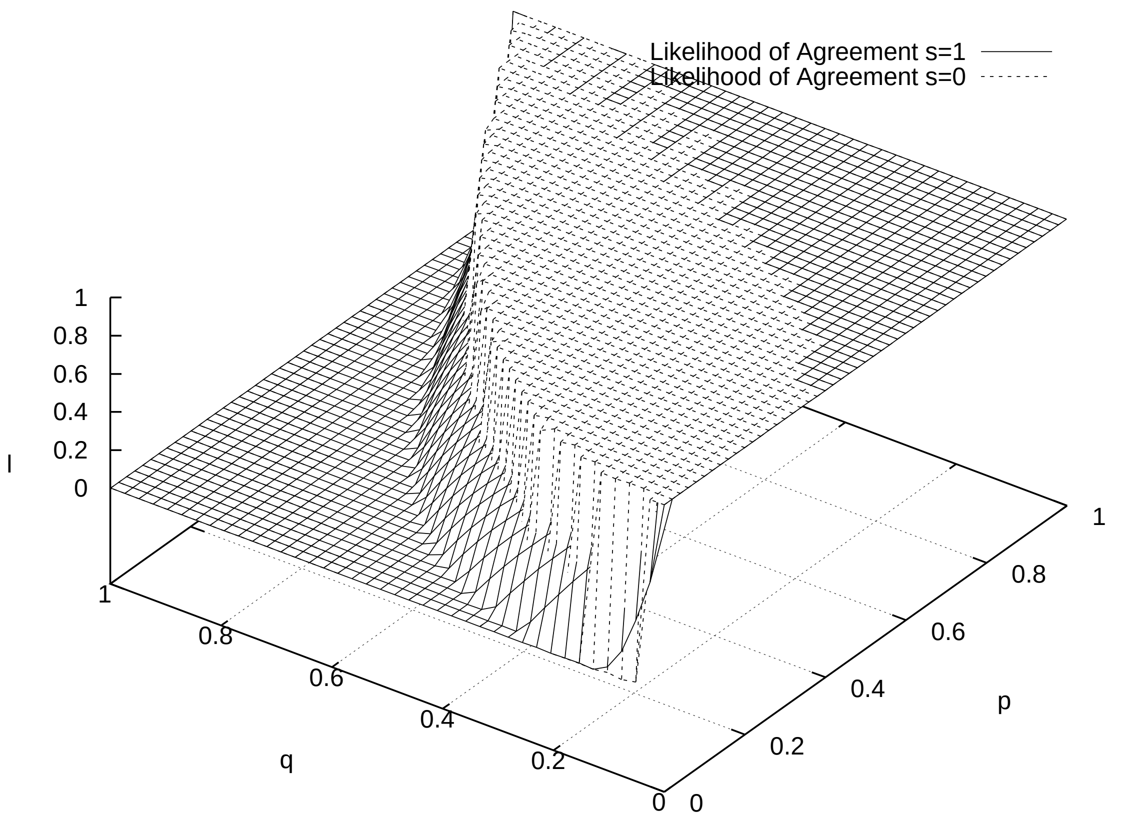


Figure 7.2: Likelihood of agreement in TUPLEBIDDING for $s \in \{0,1\}$, $r = 0$

of some graphs in Section 7.8.2 is an artifact of the simulation with a discrete increment and potential floating-point rounding errors in the runtime environment.

In the remainder of this section, we first review optimal bidding strategies, expected utility of both players, and welfare for each mechanism separately to provide an intuition in the strategic effects of the mechanisms under risk ($s > 0$) and with the consumer being risk averse ($r > 0$). In the following Section 7.7.2, a direct comparison of the three mechanisms is presented to identify which mechanism performs best in which situation and for what player.

7.7.3 TUPLEBIDDING

Obviously, $\mathcal{C}$'s bid consisting q and p has two effects: It determines the likelihood of reaching an agreement (cf. Figure 7.2) and the utility of an agreement, should it be achieved. In bidding, $\mathcal{C}$ trades-off these two directly opposed effects for maximizing her expected utility. In the border case of complete information ($s = 0$), bidding is straightforward and the numerical simulation coincides with the analytical solution presented above. Any $\{q, p\}$-tuple belongs to either of two sets: Either it leads to agreement with certainty, or to disagreement with certainty.[12] Among the tuples that lead to an agreement, $\mathcal{C}$ maximizes her utility by choosing the efficient allocation.

[12]Recall the assumption that the provider accepts when the tuple belongs to his cost function, i.e. accepts a utility of zero for himself.

Introducing risk ($s > 0$), $\mathcal{C}$'s optimization problem becomes more tricky. Numerical results (cf. Figures 7.3 and 7.4) show that for an increasing risk parameter s, $\mathcal{C}$ initially prefers bidding more conservatively, i.e. she foregoes utility from the agreement but assures reaching agreement with 100% certainty. She does so by simultaneously lowering q and increasing p (both to her disadvantage in case of agreement), i.e. by moving the bid to the upper left in Figure 7.1. At one point – called "tipping point" t from here on –, s becomes excessive from $\mathcal{C}$'s viewpoint. $\mathcal{C}$ stops retracting and starts bidding more aggressive. From $s > t$ onwards, $\mathcal{C}$ simultaneously increases q and decreases p; she demands higher utility in case of agreement and takes the risk of not reaching an agreement.

Under complete information ($s = 0$), the efficient agreement is reached, $\mathcal{C}$ claims all the value, and $\mathcal{P}$ has a utility of zero. Welfare, i.e. the sum of utilities, is at its maximum. With increasing risk ($s > 0$ and increasing), $\mathcal{C}$'s utility decreases monotonically and increasingly rapidly – up to the tipping point (Figure 7.7a). At the tipping point, $\mathcal{C}$ changes her strategy (see above): The monotonic decline of expected utility persists, but is slowed down.

As $\mathcal{C}$'s risk increases, $\mathcal{P}$'s utility increases – again, up to the tipping point, when $\mathcal{P}$'s expected utility reaches its maximum and then sharply declines with further increasing uncertainty (Figure 7.8a). The effect on welfare is straightforward: Welfare is optimal for $s = 0$. With risk, the parties on average no longer agree on the efficient allocation; welfare declines gradually. With risk s beyond the tipping point t, when $\mathcal{P}$'s utility declines, the decline of welfare increases its speed (Figure 7.9a).

All of the above holds for risk neutrality ($r = 0$). With the cost function, scoring function and utility functions chosen for numerical simulation, the tipping point is $t = 0.7$.[13] When $\mathcal{C}$ is increasingly risk averse ($r > 0$ and increasing), t increases, as the "gambling strategy" is more risky and, thus, less preferred by $\mathcal{C}$. $\mathcal{C}$'s strategy and expected utility, $\mathcal{P}$'s strategy and expected utility as well as welfare qualitatively all follow the pattern described above, only with higher t (Figures 7.7b, 7.8b, 7.9b).

7.7.4 SCORINGBIDDING

Under risk and when $\mathcal{C}$ bids a scoring function, $\mathcal{C}$ faces the same trade-off as with price-quality tuples. The trade-off is between utility in case of agreement and the likelihood of reaching agreement (cf. Figure 7.5).

Numerical results show that $\mathcal{C}$'s strategy in bidding a scoring function follows the same pattern as in bidding a price-quality tuple (cf. Figures 7.3 and 7.4): With increasing risk (s), the consumer again bids more conservatively. Technically, she decreases $\hat{a}$ and bids more truthfully. She does so sufficiently to assure certainty of reaching an agreement. Again, this holds up to the tipping point t, at which $\mathcal{C}$ starts gambling, i.e. she gradually increases $\hat{a}$ for claiming more value at the risk of not reaching agreement. With increasing risk aversion, the cost of gambling rises and, thus, the tipping point rises from $t = 0.7$ for $r = 0$ to $t = 0.9$ for $r = 1$. Interestingly,

[13]The exact tipping point (0.7) depends on the exact parameters chosen. We believe however, that the existence of such a tipping point is almost universal.

[15]Note that $\hat{a}^*$ of SCORINGBIDDING has been divided by four to improve the readability of the chart.

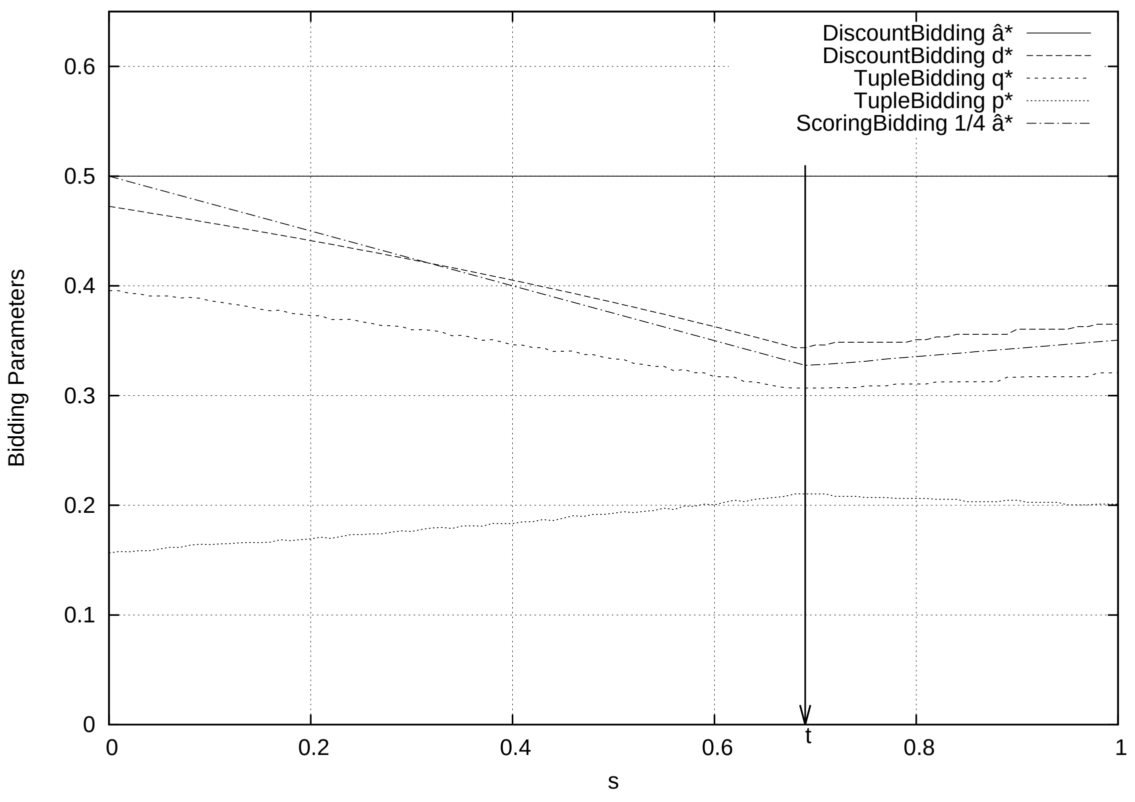


Figure 7.3: Optimal bidding parameters for all three negotiation mechanisms depending on the consumer's risk (s) and risk aversion ($r = 0$)[14]

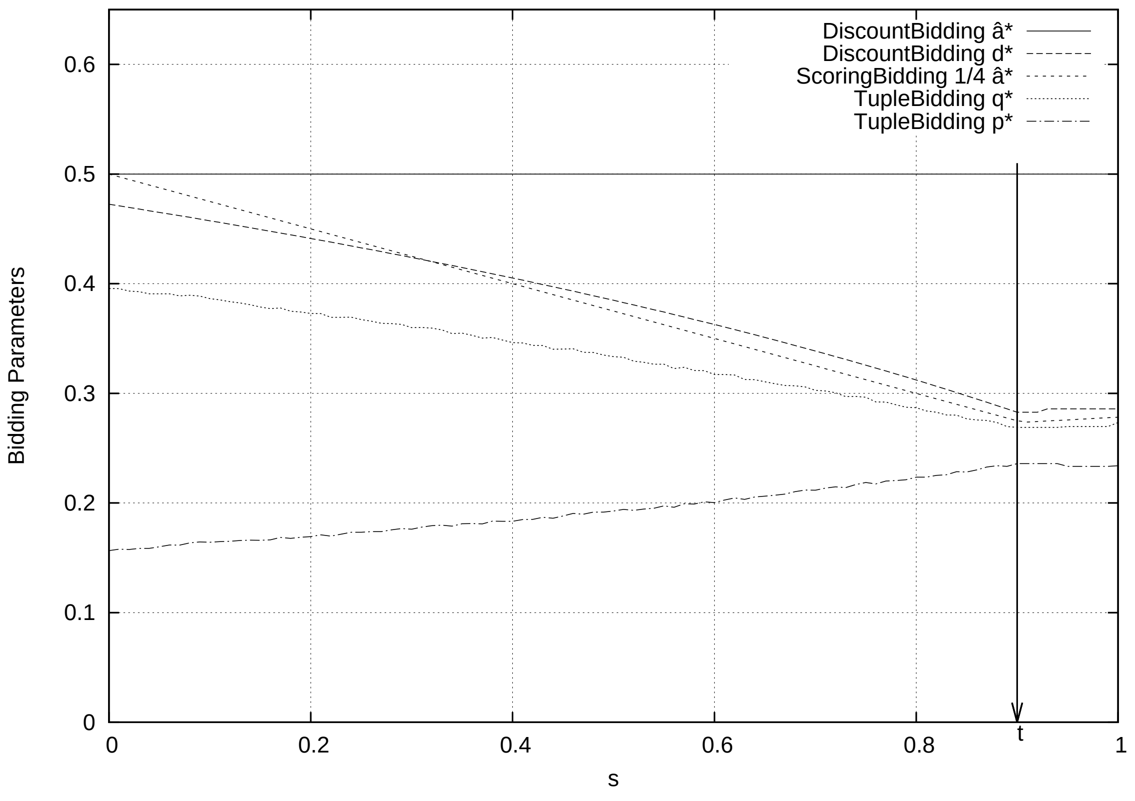


Figure 7.4: Optimal bidding parameters for all three negotiation mechanisms depending on the consumer's risk (s) and risk aversion ($r = 1$)[15]

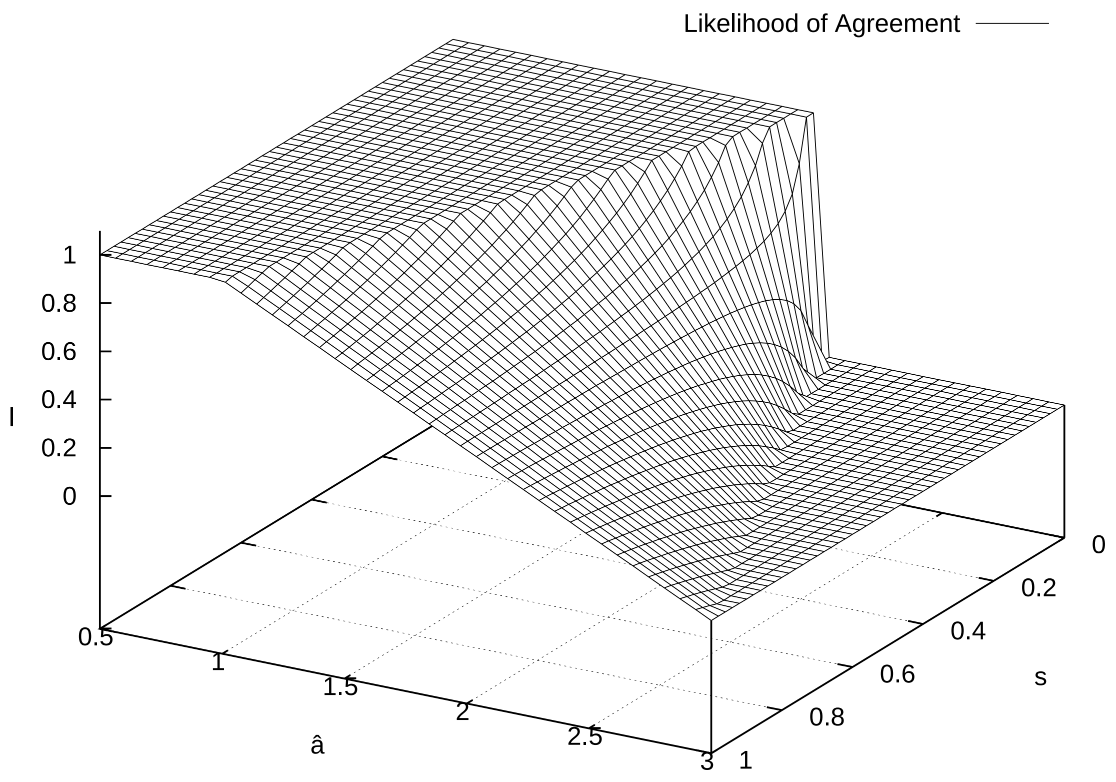


Figure 7.5: Likelihood of agreement in ScoringBidding for $s \in [0,1]$, $r = 0$

the tipping point coincides with the tipping point for bidding price-quality tuples for any r.[16]

$\mathcal{C}$'s and $\mathcal{P}$'s expected utility as well as welfare all three qualitatively resemble the patterns from TupleBidding for varying s and r. $\mathcal{C}$'s expected utility is maximal for $s = 0$ and decreases monotonically for an increasing risk parameter s (Figure 7.7a, 7.7b). The decrease speeds up for increasing s up to the tipping point and slows down from $t > s$ onwards. $\mathcal{P}$'s expected utility is zero for $s = 0$, increases monotonically for increasing s up to the tipping point. It sharply kinks at $s = t$ and decreases thereafter (Figure 7.8a, 7.8b).

Interestingly, welfare is optimal for $s = t$. The intuition is that strategic bidding by $\mathcal{C}$ hinders the integrative part of the negotiation; risk initially lowers $\mathcal{C}$'s strategic misrepresentation and thereby allows $\mathcal{P}$ to come closer to the efficient quality. At $s = 0$, $\mathcal{C}$ submits a bid of $\hat{a}$ as close as possible to b, optimizing her own utility, yet sacrificing efficiency by overbidding by almost four times the real value a (for $a = 0.5$ and $b = 2.0$). As stated before, with increasing uncertainty ($s > 0$), $\mathcal{C}$ will lower her bid, thus increasing efficiency simultaneously. After passing the utmost efficient point for $s = t$, $\mathcal{C}$ starts gambling by raising $\hat{a}$ and welfare starts to rapidly decline again (Figure 7.9a, 7.9b).

[16]For other parameters a, b the tipping point is slightly shifted, cf. Appendix B.3.

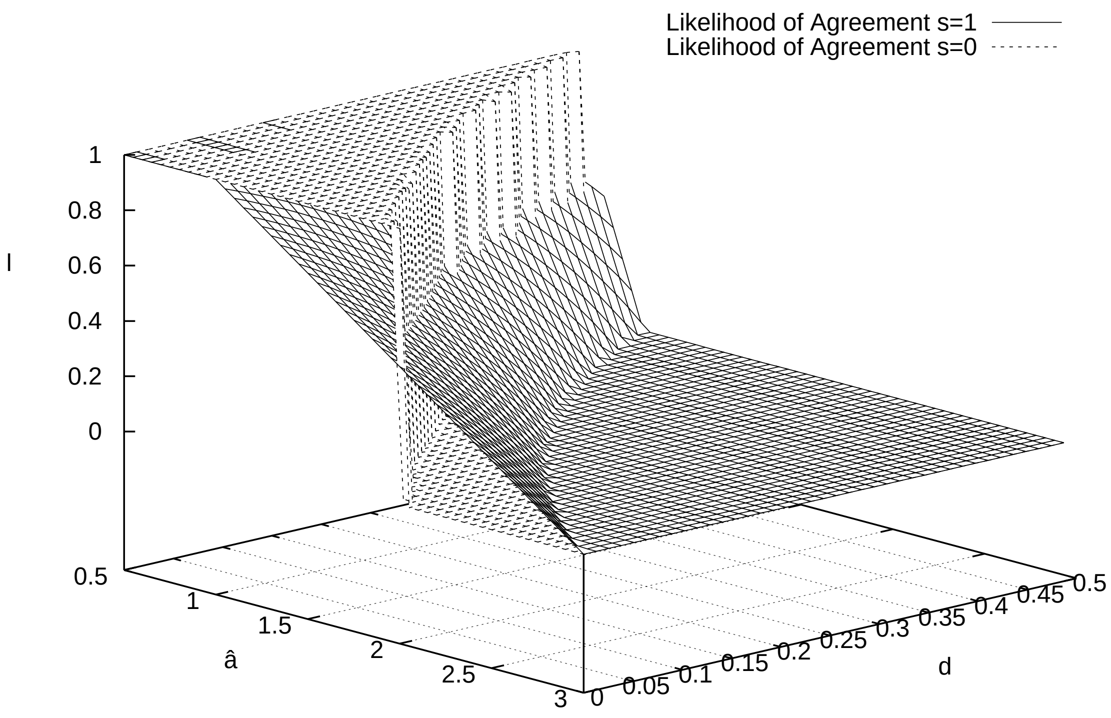


Figure 7.6: Likelihood of agreement in DISCOUNTBIDDING for $s \in \{0,1\}$, $r = 0$

7.7.5 DISCOUNTBIDDING

$\mathcal{C}$'s trade-off decision is the same as for the other mechanisms, only the vehicle of bidding more or less conservatively differs (cf. Figure 7.6). It can be implemented by either lowering scoring parameter $\hat{a}$, or by lowering discount d, or both. The interesting question is which of these vehicles $\mathcal{C}$ chooses to maximize her expected utility.

Analogously to the analytical results in case of complete information, numerical results show that bidding the scoring function truthfully ($\hat{a} = a$) and claiming value via discount d is the strategy maximizing $\mathcal{C}$'s expected utility. For the functional forms of $C(q)$ and $S(q)$ being strongly convex/concave (cf. Section 7.4), bidding the scoring function truthfully is the only optimum.[17] Considering, for instance, the border case of $s = 0$: $\mathcal{C}$ knows the efficient allocation that maximizes her utility $(q^*, p = C(q^*))$ and needs to derive a bid $(\hat{a}, d)$ to meet that very allocation. With $\hat{a} \neq a$, the provider's optimization will yield a quality different from q^*, value is lost and cannot be regained by any discount d (cf. Section 7.6.3). Numerical results show that the same holds true under uncertainty.

For more general scoring and bidding functions, however, bidding the scoring function truthfully might not be the only optimum – it is, however, always one optimum, as long as $\mathcal{P}$ chooses the efficient quality q for a truthful bid.

Again, the qualitative patterns in strategies and utilities resemble what is known from the other mechanisms: With increasing risk s, $\mathcal{C}$ initially bids more and more

[17]The fluctuations in Figures 7.3 and 7.4 for $\hat{a}^*$ and d^* are an artifact caused by the increment in our implementation, as the difference between the exact theoretical maximum welfare w^* and the second best welfare is smaller than the increment with its three decimal places.

conservative, lowering discount d. The tipping point t exists at which this pattern inverts and $\mathcal{C}$ switches to gambling, i.e. to increasing d (cf. Figures 7.3 and 7.4).

Up to t, $\mathcal{C}$'s utility decreases in s and $\mathcal{P}$'s utility increases. For increasing $s > t$, the decline of $\mathcal{C}$'s utility slows down (Figure 7.7a, 7.7b) and $\mathcal{P}$'s utility decreases as well (Figure 7.8a, 7.8b). Welfare decreases in s with a kink at t (Figure 7.9a, 7.9b). Increasing risk aversion r increases t but does not qualitatively affect behavior or outcomes. All variation in $\mathcal{C}$'s bidding depending on s and r takes place in the discount d; the revelation of the scoring function remains truthful ($\hat{a} = a$).

In summary, discount bidding effectively disentangles the integrative and distributive elements of the negotiation. Bidding the scoring function truthfully allows maximizing the consumer's expected utility and expected welfare in any given state of the world, i.e. for any combination of provider cost function (parameter b), consumer scoring function (parameter a), and consumer risk aversion (parameter r). Comparing discount bidding with bidding a pure scoring function (Section 7.7.4) shows that this truthfulness is only possible, as the discount factor d provides the consumer a vehicle to claim value, i.e. d is the distributive element of the negotiation.

Truthfulness in the scoring function is an interesting and potentially beneficial property of the discount bidding mechanism. The even more interesting question is, however, how the three mechanisms compare in terms of utility for one or the other party, as this will drive adoption in the marketplace. The following Section 7.8 presents this comparison.

7.8 Comparison of Negotiation Mechanisms

This section provides a comparison of the three mechanisms regarding utility and welfare. The first comparison is under complete information, followed by a comparison under incomplete information. The comparison is based on the findings of Sections 7.6 and 7.7.

7.8.1 Complete Information

Under complete information TUPLEBIDDING, as aforementioned, allows the consumer to choose the efficient allocation q^* while claiming the entire surplus by setting the price to the value of $\mathcal{P}$'s cost function. The same is possible when DISCOUNTBIDDING is implemented, as $\mathcal{C}$ can use the discount d to obtain the entire surplus, while ensuring efficiency by truthfully bidding $\hat{a}$. Thus, under complete information, both mechanisms result in the same allocation, utility and welfare.

SCORINGBIDDING cannot achieve the same efficiency, as $\mathcal{C}$ is always tempted to misrepresent her type by overbidding $\hat{a}$, thus leading to a quality agreement that does not match q^*. Despite the fact that it yields the same utility for $\mathcal{P}$, it turns out that SCORINGBIDDING leads to a lower utility for $\mathcal{C}$, which also implies a lower welfare than when using TUPLEBIDDING or DISCOUNTBIDDING as negotiation mechanism. In mathematical terms, this is represented by the following strict inequality:

$$e^{-\frac{a}{b}} - \frac{1}{e} < \left(\frac{b}{a}\right)^{\frac{a}{a-b}} - \left(\frac{b}{a}\right)^{\frac{b}{a-b}} \tag{7.28}$$

which holds $\forall a \in (0, b)$ and $\forall b \in [1, \infty)$.[18]

7.8.2 Incomplete Information

Figures 7.7 to 7.9 show the numerical results for the consumer's expected utility, the provider's expected utility and welfare for all three negotiation mechanisms. The upper graph of each figure – labeled with (a) – relate to risk neutrality ($r = 0$), the lower graph – labeled with (b) – to risk aversion ($r = 1$). $a = 0.5$ and $m = E(b) = 2$ is constant across all settings.[19] Qualitatively equivalent but numerically different results were obtained for other a and b combinations (cf. Appendix B.3).

Figure 7.7a shows that for any given risk s, the consumer's expected utility is equal for DISCOUNTBIDDING and TUPLEBIDDING (the lines are exactly on top of each other) and strictly lower for SCORINGBIDDING. This result is independent of the consumer's risk aversion, as Figure 7.7b exemplifies. The implication is twofold:

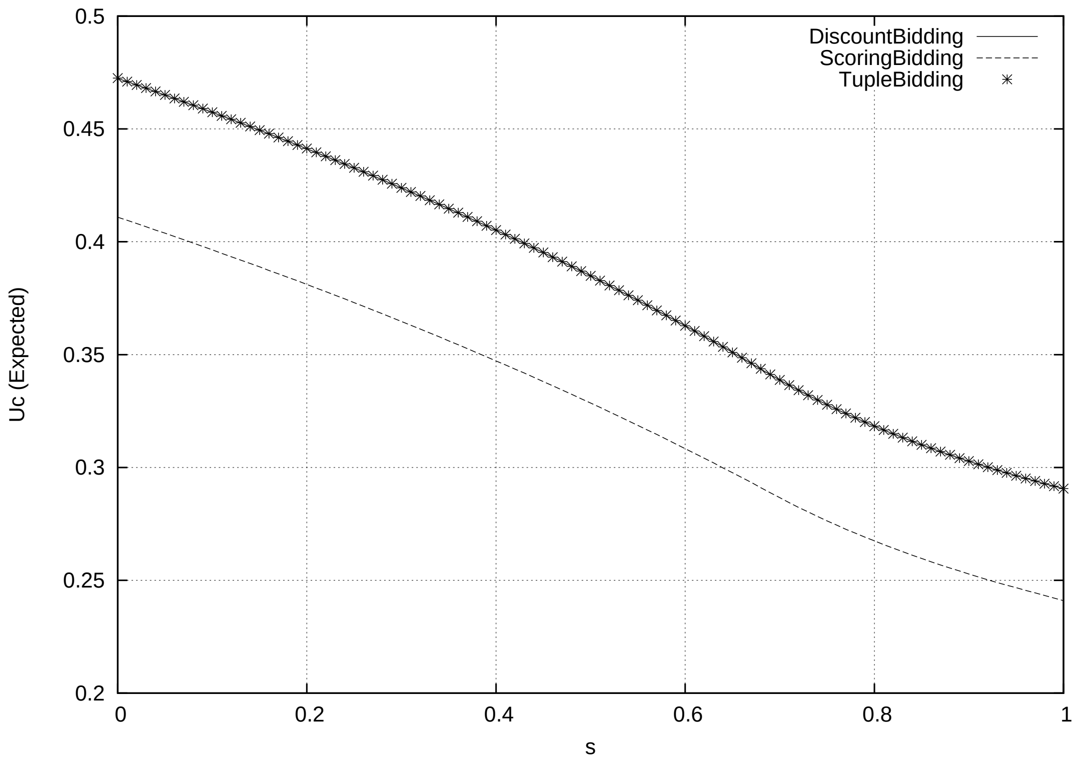
- When the consumer can choose the mechanism, she will prefer either DISCOUNTBIDDING or TUPLEBIDDING over SCORINGBIDDING. The consumer may have the ability to choose the mechanism when either there are multiple providers offering different mechanisms, or when she has sufficient purchasing power to dictate the mechanism.
- When the consumer has the chance to reduce her risk at a reasonable cost in terms of time and money, she will do so. She may have the chance to acquire information either by standard market research, or by learning from repeated negotiations with a single or with multiple providers.

Figure 7.8a shows a clear ranking of the three mechanisms from the provider's view: SCORINGBIDDING is preferred over DISCOUNTBIDDING which is preferred over TUPLEBIDDING. This holds for all $s > 0$; for $s = 0$, the provider is indifferent between the three mechanisms. The implication is, again, twofold:

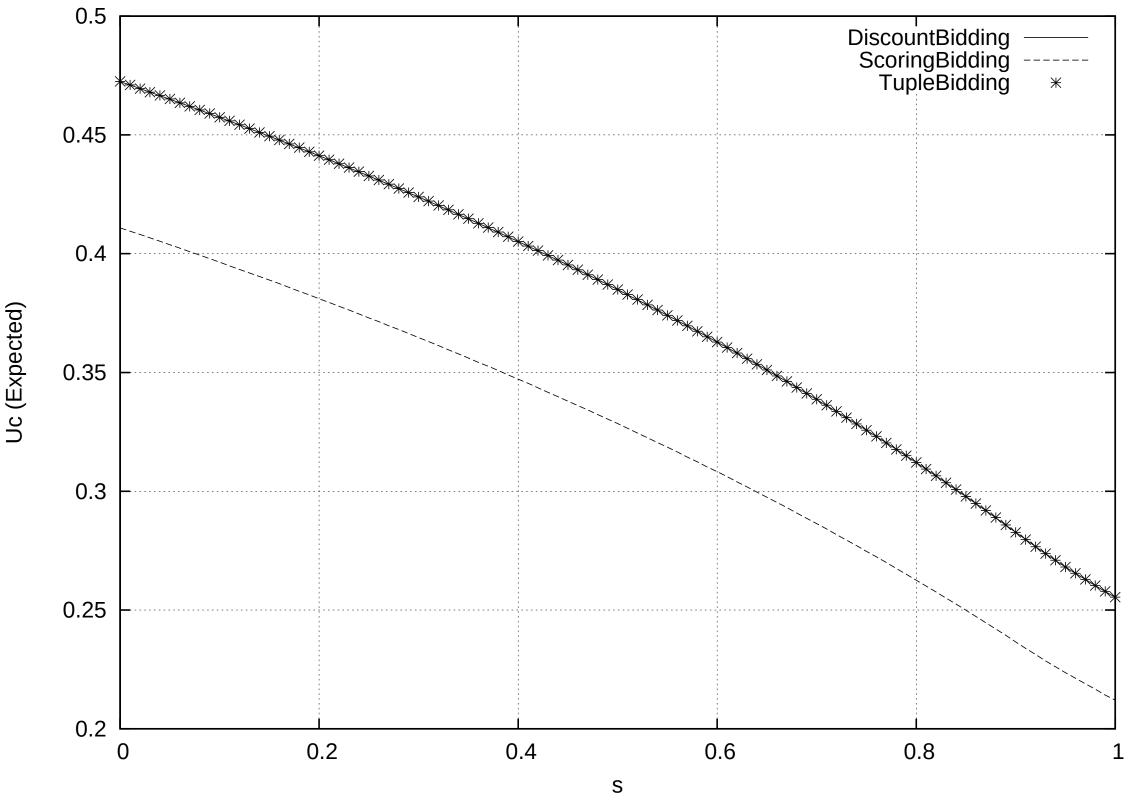
- When the provider can choose the mechanism, he will select SCORINGBIDDING.
- When the provider can influence the consumer's risk, he will do so. He will try to provoke conservative bidding by the consumer but will not exaggerate the risk to a level where the consumer starts gambling and the provider risks not reaching an agreement at all. The provider can e.g. influence risk by withholding information on the exact technology employed and on his costs. Both

[18]For mathematical proof refer to Appendix B.2.

[19]In graph b, $R^{-1}(E(U_C))$ is shown, i.e. the back transformation with respect to risk aversion. The underlying strategic decision making still considers $\mathcal{C}$'s risk aversion. However, the numerical change for this graph allows a comparison of absolute numbers and slopes across graphs a and b.



(a) Expected utility of $\mathcal{C}$ for $r = 0$



(b) Expected utility (applied R^{-1}) of $\mathcal{C}$ for $r = 1$

Figure 7.7: Comparison of negotiation mechanisms with respect to $\mathcal{C}$'s utility depending on the level of the consumer's risk s and risk aversion r

is common in real-world settings. The provider has, however, an interest on providing some information on technology and, thus, on implied costs. Again, this is a common behavior in real-world settings.

The ranking of mechanisms from the provider's viewpoint is independent of the consumer's risk aversion (cf. Figure 7.8b). The provider's strategic manipulation of the consumer's risk, e.g. by providing or withholding information on the technology for service provision and associated costs, depends on the consumer's risk aversion. This, in turn creates risk on the provider side with regards to the exact degree of the consumer's risk aversion. The analysis of an equilibrium in this extended game is beyond the scope of this thesis.

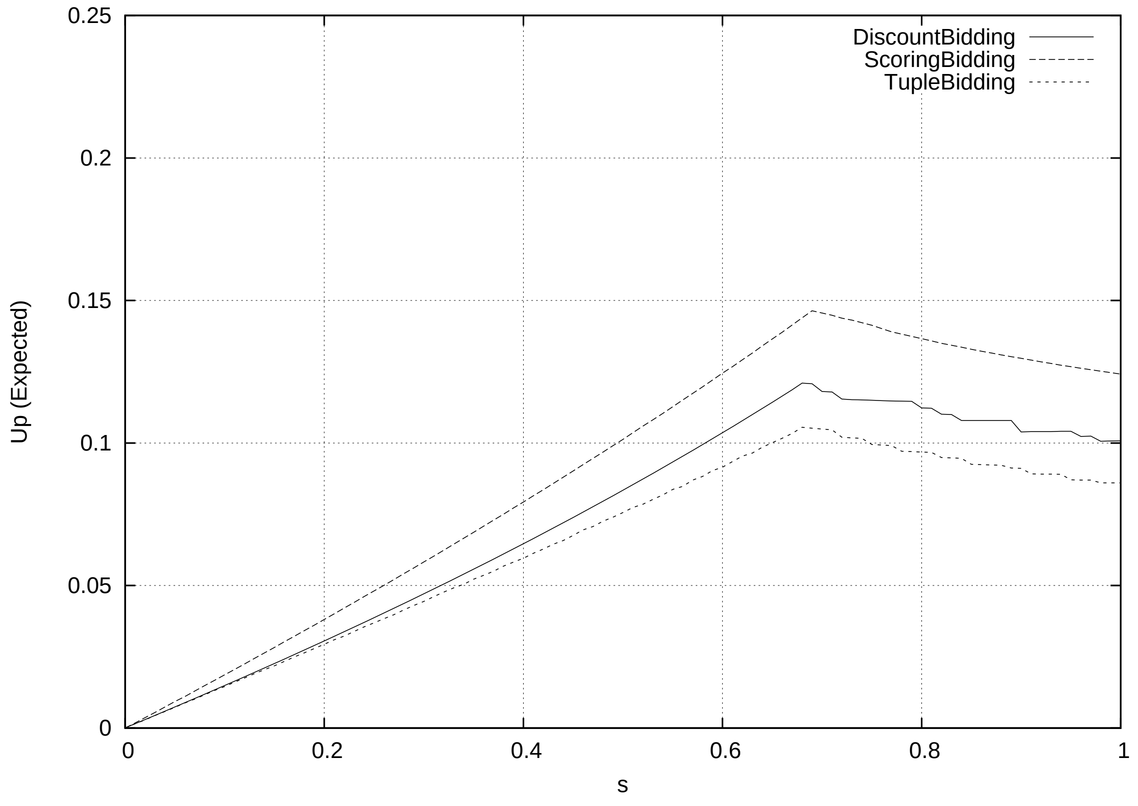
Figure 7.9a – the welfare – combines the above perspectives and, again, provides a ranking of mechanisms. Interestingly, for the case of the consumer being risk averse, this ranking depends on the level of risk (cf. Figure 7.9b). For low risk, welfare from TUPLEBIDDING is higher than from SCORINGBIDDING.[20] For high risk and risk aversion, this ranking inverts. In any case, DISCOUNTBIDDING results in at least the same welfare for the special case of $s = 0$ and strictly higher expected welfare for the general case of $s > 0$. This result leads to the following implication: When a third party can choose the mechanism, it will select DISCOUNTBIDDING as welfare maximizing mechanism among the three mechanisms, independent of risk and risk aversion. Note, however, that in the scenario studied in this chapter, it is rather unlikely that a third party like a regulator imposes a mechanism to the bilateral negotiation.

Looking beyond the scope of a single negotiation, three observations stand out: Firstly, the provider will strategically manipulate the consumer's risk prior to the negotiation, if he has the chance of doing so (which will hold true in most real-world settings, where any provider will carefully decide on how much information on his technology he is willing to give price). His challenge is to find the optimal degree of risk, as neither a low nor a high risk by the consumer is optimal for the provider.

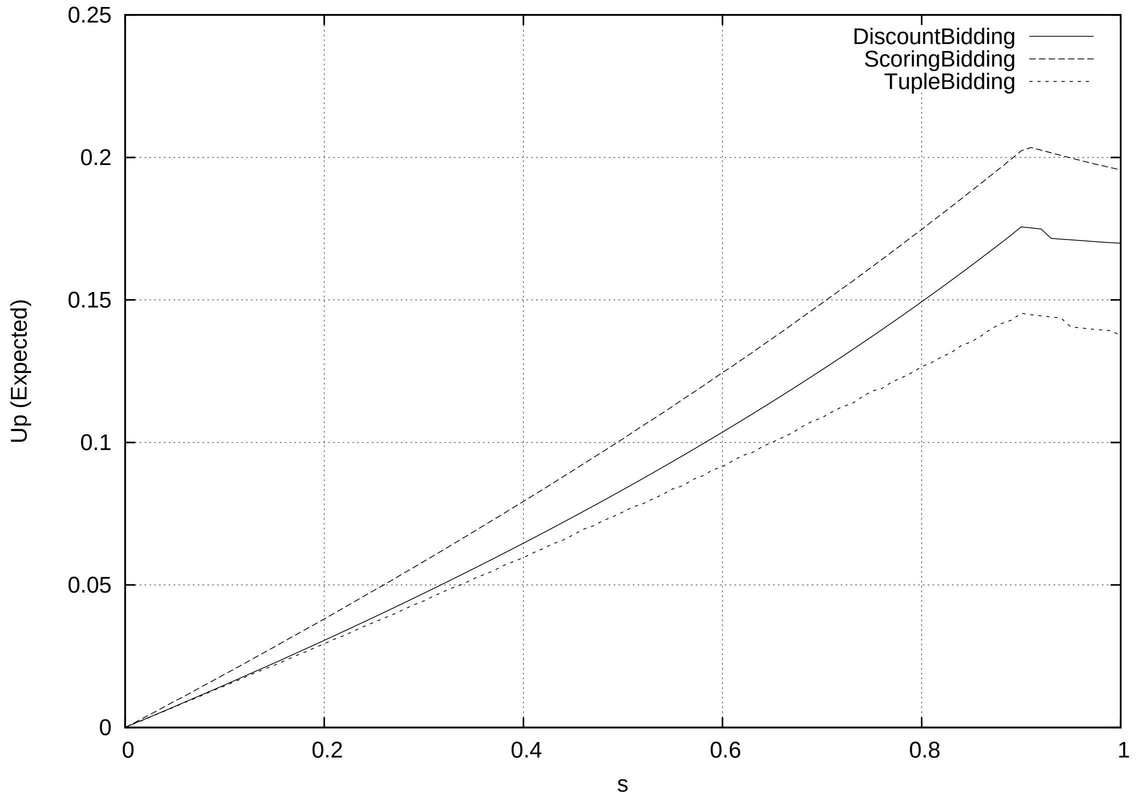
Secondly, DISCOUNTBIDDING is an attractive mechanism, but not 100% certain to be adopted in practice. In the extended game of selecting a mechanism, TUPLEBIDDING is not Pareto-optimal, it is dominated by DISCOUNTBIDDING. DISCOUNTBIDDING will prevail when either the consumer or a third party have the discretion or power to impose a mechanism. Bidding scoring functions will be adopted when the provider decides on the mechanism. A positive side effect exists that might increase the provider's utility from DISCOUNTBIDDING in the long term: DISCOUNTBIDDING promotes truthful revelation of the consumer's scoring function and, thereby, allows the provider to optimize his technology portfolio and cost structure in the long-term. This effect – that is not reflected in the provider's utility function in this chapter – might lead all parties to unanimously prefer discount bidding over the other mechanisms.

Thirdly, the results presented in the previous sections can be also be interpreted from the perspective of the contract [30] or principal-agent theory [57, 91]. Seen

[20]Note that welfare can increase for SCORINGBIDDING and potentially equal welfare of TUPLEBIDDING when allowing for more general forms of scoring functions.

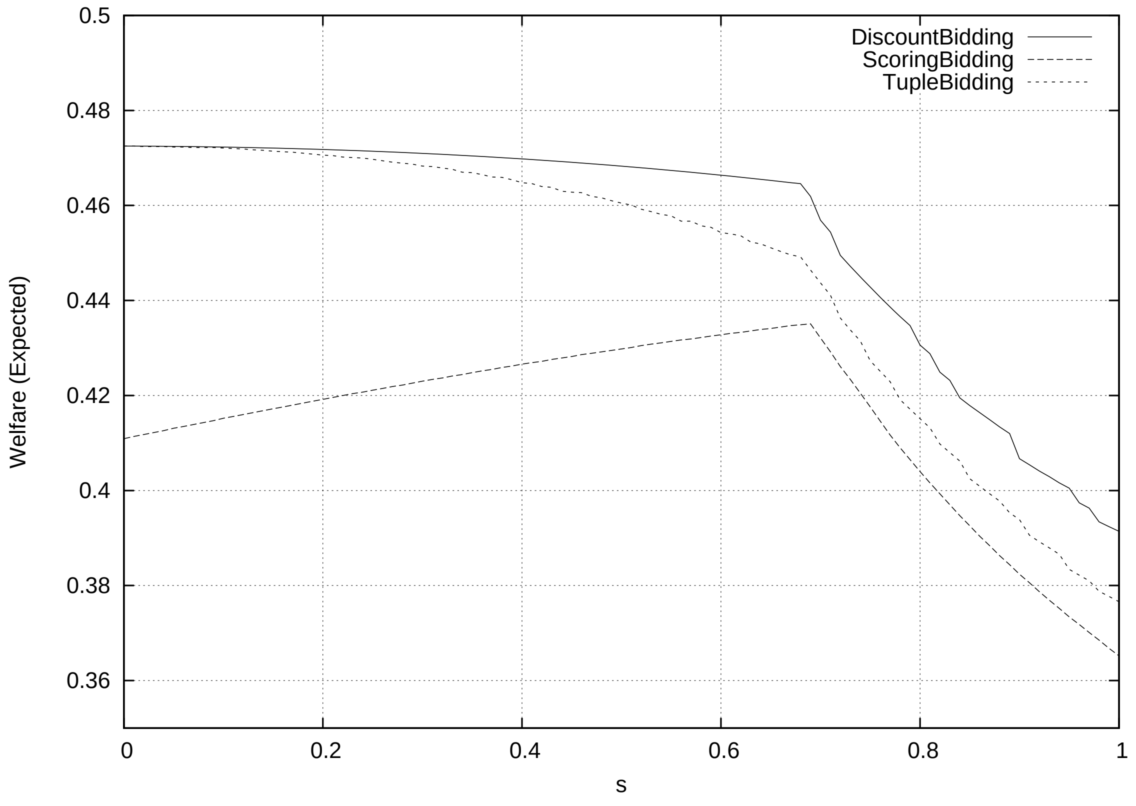


(a) Expected utility of $\mathcal{P}$ for $r = 0$

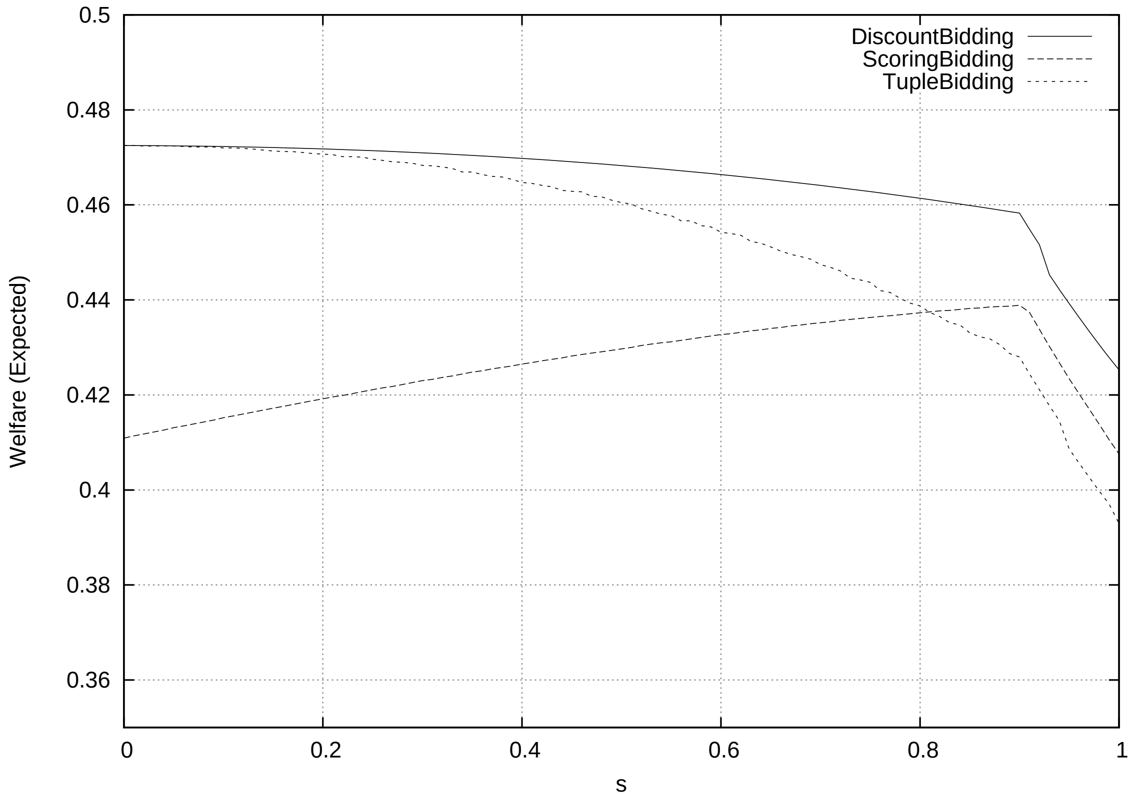


(b) Expected utility of $\mathcal{P}$ for $r = 1$

Figure 7.8: Comparison of negotiation mechanisms with respect to $\mathcal{P}$'s utility depending on the level of the consumer's risk s and risk aversion r



(a) Expected welfare for $r = 0$



(b) Expected welfare for $r = 1$

Figure 7.9: Comparison of negotiation mechanisms with respect to welfare depending on the level of the consumer's risk s and risk aversion r

from this perspective, TUPLEBIDDING can be considered as a complete non-linear pricing mechanism. In SCORINGBIDDING, the principal (consumer) is more constrained since it does not have a contract term to directly transfer (or, appropriate) surplus from the agent (provider). Therefore, it is intuitive that SCORINGBIDDING is inferior in terms of efficiency. The principal has to sacrifice efficiency to address the moral hazard of the agent (provider). In DISCOUNTBIDDING, the principal is less constrained compared to the case of SCORINGBIDDING because it can use a discount term to transfer surplus from the agent. However, the principal's contracting vehicle is still not as flexible as TUPLEBIDDING because the principal cannot directly specify the quality level, but only the functional form of the quality. Such a constraint then forces the principal to balance between the contracting terms – the bidden functional form and the discount term – in addressing the incomplete information and moral hazard. In SCORINGBIDDING, actually the agent determines the surplus transfer, i.e. the price. All in all, the results presented in Section 7.8 are congruent with these considerations.

7.9 Conclusion

In this chapter an economic model for multi-attributive negotiations was presented. The model is set in the overall scenario of this thesis – mass customized Cloud services that require not only an agreement on functional requirements, but in addition also require agents to agree on non-functional aspects. Specifically, bilateral negotiations on quality and price of a service between service provider and consumer were studied. In this setting, a negotiation has an integrative facet – many possible qualities of service are sub-optimal, but by means of communication within the negotiation mechanism, the parties can identify a Pareto-optimal quality of service. On the other hand, the negotiation has a distributive facet – either party has an interest in claiming as large a share in the value from an agreement as possible. Strategic bidding typically leads to negotiators mixing the integrative and distributive facets which results in inefficient outcomes.

In such a scenario, the selection of an "optimal" or at least "satisfying" negotiation mechanism is a challenge. In the light of the Myerson-Satterthwaite impossibility theorem [121], individual rationality and budget balance were postulated for analyzing economic properties of different negotiation mechanisms, namely in how far agents' negotiation strategies deviate from truthful revelation of their types and in how far efficiency of negotiated agreements deviates from the efficient agreement an omniscient arbitrator would define. On this theoretical background, three negotiation mechanisms were compared: TUPLEBIDDING, SCORINGBIDDING and DISCOUNTBIDDING. TUPLEBIDDING serves as a proxy for commonly used fixed price mechanisms; SCORINGBIDDING resembles the widely used approach of agents bidding a scoring function, e.g. in multi-attribute auctions. DISCOUNTBIDDING was newly introduced in this thesis – it allows to bid a scoring function and additionally a discount that the consumer demands from the provider. The intuition is that this approach disentangles the integrative and distributive facets of the negotiation and increases efficiency. The results confirmed this intuition. For complete information, these results were derived analytically from the game theoretic equilibrium;

for the extended case of incomplete information, risk and risk aversion, a numerical simulation was conducted to characterize the mechanisms.

TUPLEBIDDING is not Pareto-optimal but dominated by DISCOUNTBIDDING. Thus, it is unlikely to prevail in a marketplace for custom services as soon as this market matures. Nowadays, comparable mechanisms are used by some Cloud service providers, for example. It has the advantage of a simple bidding language and very little communication effort. However, as providers and consumers get more and more sophisticated and as automatic negotiations become more prevalent, the disadvantageous economic properties will weigh heavier and TUPLEBIDDING will become less relevant.

SCORINGBIDDING emphasizes the integrative facet and yields higher expected utility for the provider than either of the other mechanisms. When the provider can dictate the choice of the mechanism, he will presumably favor SCORINGBIDDING. However, the consumer's strategic misrepresentation of her scoring function leads to sub-optimal agreements. DISCOUNTBIDDING captures even more of the integrative facet: it promotes truthful revelation of the consumer's scoring function and thereby allows reaching an efficient agreement more often than with SCORINGBIDDING. The expected welfare from DISCOUNTBIDDING is higher than from SCORINGBIDDING for any level of risk and risk aversion. Compared to SCORINGBIDDING, the discount factor in DISCOUNTBIDDING shifts utility from the provider to the consumer. Whenever the consumer or an independent third party can dictate the negotiation mechanism, she will tend to favor DISCOUNTBIDDING. A positive long-term effect of DISCOUNTBIDDING is that truthful revelation of the consumer's scoring function allows the provider to adapt his technology and service offering. In the long run, this may even overturn the provider's favoritism for SCORINGBIDDING.

All mechanisms show a tipping point in the consumer's behavior depending on the risk: with risk below this tipping point, the consumer bids conservatively and assures reaching an agreement; beyond this risk, she bids aggressively and risks not reaching an agreement. For a given level of risk aversion, this tipping point is – somewhat surprisingly – identical for all three mechanisms.

In each of the mechanisms, the provider has an incentive to strategically manipulate the consumer's risk by determining the level of information provided on his technology and costs. Neither full transparency nor opacity are optimal for the provider, but he will have to carefully chose the level of information depending on the consumer's risk aversion. The higher the consumer's risk aversion, the less information will be given by the provider. All results hold inversely when inverting the roles of consumer and provider in the negotiations.

7.9.1 Limitations and Implications

The presented results have four main limitations: (1) Firstly, individually rational utility maximizing agents are assumed; in the real-world, such a setting is rare. (2) Secondly, the results depend on the model of preferences, especially on the functional forms of cost, scoring, and utility functions, the assumed probability distribution function and are limited to a single QoS attribute. While we believe that similar results can be obtained for other preferences and more than one QoS attribute, this has not been proven yet. (3) Thirdly, "only" three distinct mechanisms are studied without deriving an "optimal mechanism" in the mechanism design sense. However, given the complexity and impossibility theorems, this comparison of existing mechanisms and introduction of DISCOUNTBIDDING as additional mechanism is a valuable contribution to the field. (4) Fourthly, and most importantly, all three mechanisms only allow for a single offer and its acceptance or rejection by the counterparty. More complex negotiation mechanisms with an alternating offer exchange are possible, so is the introduction of a central marketplace.

For the overall scenario of this work, and in particular for the service optimization following in Chapter 8, the obtained results have two implications: Firstly, a provider offering mass customized Cloud services will prefer SCORINGBIDDING over the other two mechanisms under investigation. As direct implication, the optimization scenario in the subsequent chapter implements SCORINGBIDDING, yet allowing more than one quality attribute in the scoring function.[21] Secondly, a Cloud provider implementing any of the three mechanisms in a mass customization scenario should be well aware of how much information to reveal to his customers, as neither too much, nor too little information on his cost structure was found to be beneficial.

[21]Note that from an optimization view point DISCOUNTBIDDING could also be implemented without implications on the optimization mechanism, as finding the optimal configuration is independent of the discount factor d.

Chapter 8

Service Optimization

It is the goal of this thesis to enable Cloud providers to offer mass customized services in a profit maximizing way. So far, we introduced a semantic service description framework, an ontology update manager and a service engineering algorithm, which altogether allow to derive feasible service configurations for a given consumer request. The resulting configurations are formalized in a graph structure – a so called service configuration graph. However, it remains open how to find the optimal configuration within this graph, i.e. the configuration which yields the highest profit for the provider.

Chapter 7 presented different negotiation mechanisms which help provider and consumer in finding an agreement on both quality and price of a service. In two of the three presented mechanisms – SCORINGBIDDING and DISCOUNTBIDDING – the provider is expected to find the profit maximizing quality given a scoring function and his own cost function, which were modeled as monomials. Reality, however, is more complex. The scoring function representing the consumer preferences regarding the non-functional service properties in many cases has to capture more than one QoS attribute. The cost function cannot be represented by a monomial, as different pairs of quality and costs depend on the available service resources and thus result in a discrete step function. While the economic implications drawn in the preceding chapter are also believed to be valid in more complex models, this chapter has its focus on optimization techniques. We thereby assume a complex form of SCORINGBIDDING to be implemented by the provider, allowing to state consumer preferences on more than one QoS attribute.[1] The challenge lies in the combinatorial complexity of finding the best alternative in a service configuration graph, that can get high even for rather small graph instances. Especially, since overall Quality-of-Service levels have to be obtained by aggregation. The aggregation operators strongly influence the computational complexity. Depending on the graph structure and the QoS aggregation operators, different techniques from brute-force, over integer programming, to shortest-path algorithms are possible and imply their own advantages and disadvantages.

In this chapter the challenge of the above mentioned computational complexity is addressed by presenting optimization techniques, efficient aggregation func-

[1] The results of this chapter could also be applied to DISCOUNTBIDDING. TUPLEBIDDING is dominated by the other mechanisms, hence it is not considered in this chapter.

tions including their formulation within integer programs and heuristics. We first introduce related work from different research areas in Section 8.1. The concrete optimization model as well as the resulting challenges in this model follow in Section 8.2. The model includes a formalization of non-functional requirements, the solution space of the optimization, functions for aggregating QoS values and the objective function. Challenges arise from conjunctive edges (and nodes) in the service configuration graph and from aggregation functions that violate the Bellman property. Different optimization techniques coping with these challenges are described in Section 8.3. The presented techniques comprise brute-force algorithms, a binary integer programming approach and a graph-based approach using Dijkstra's algorithm. A detailed evaluation comes thereafter in Section 8.4. Some parts are evaluated analytically or numerically, while the main evaluation is performed through a simulation study. The chapter is concluded by a summary and recommendations given in Section 8.5. Minor parts of this chapter are based on [63].

8.1 Related Work

This chapter introduces different optimization techniques for finding optimal service configurations with respect to given consumer preferences and provider cost structures. Regarding the consumer preferences, Asker and Cantillon [7] introduce the concept of a scoring function for expressing quality preferences in the context of multi-attribute procurement auctions. The scoring function is used to rank different offers over different attributes like quality and costs. In the work by Blau et al. [27], this concept is picked up and used in the context of a newly introduced complex service auction, an auction based negotiation protocol for selecting and pricing a complex service based on an SVN. The scoring function thereby is used to rank different paths, i.e. feasible service composition alternatives of an SVN. The model presented in the following section is an adaption of these well established concepts for describing both the non-functional preferences and the economic objective function.

Further related literature concerning the challenge of QoS-aware service configuration can be found in the field of BPM and service composition research. In the context of automatic service composition different approaches have been proposed in recent literature. In the work by Lécué and and Léger [96] backward chaining is applied to derive suitable compositions starting from a central objective. Sirin et al. [143] propose the use of AI planning techniques to achieve an automated composition of Web Services. Thereby they use a hierarchical task network for planning in conjunction with OWL-S Web service descriptions. These and other approaches in this area are mainly based on formal description languages focusing on functional service characteristics as proposed by the W3C recommendation SAWSDL, OWL-S and WSMO. Other work, such as by Berardi et al. [16], also incorporates the service lifecycle aspect and time dependencies. Yet, all of these approaches mainly focus on service functionality as the only criteria for composition and largely ignore other non-functional properties like QoS or price for an economic evaluation.

Mathematical models for aggregating quality attributes of single services into complex services depending on multiple types of process patterns are investigated in [22, 79, 147, 87]. Although this stream of research considers different types of attributes and the implications for a corresponding aggregation algorithm, computational complexity and desired efficiency is not in scope of their investigation. Another research area targets a more comprehensive solution for managing functional and non-functional service characteristics across the entire life cycle based on a model of atomic and composite services [101, 118]. Although the outlined approaches also focus on automation and on-line computation of QoS aggregation, complexity aspects and efficient algorithm design is not in the focus. Zeng et al. [154] propose a linear programming approach that enables an automated Web service composition while maximizing user experience that is modeled as QoS dependent utility function. The authors also describe how different types of QoS values can be aggregated in such an optimization scenario. For aggregating multiplicative QoS values like a services availability, a similar approach to the later presented logarithm approximation is used. The optimization model itself, however, differs fundamentally to the model in this thesis, as both approaches deal with different scenarios.

8.2 Optimization Model and Challenges

This section introduces a formal model specific to the challenge of finding the profit maximizing configuration in a scenario of mass customized Cloud services. The model is closely related to the formalizations in the previous chapters, in particular to the model presented in Section 6.3. While the service engineering model is focused on deriving feasible service configurations to form a service configuration graph, we now concentrate on finding the optimal configuration within such a graph structure. The model containes the non-functional requirements of a consumer request for a mass customized service, a refined definition of the service configuration graph which spans the solution space obtained from functional engineering step, the aggregation functions for QoS attributes and the overall objective function of the economic optimization problem. The section is concluded with a presentation of the computational challenges arising in the economic optimization over different service composition alternatives.

8.2.1 Non-functional Requirements

A request for a mass customized Cloud service contains both functional and non-functional requirements. The non-functional requirements contain preferences regarding non-functional QoS attributes along with a price component, the consumer is willing to pay for the service. This is analogous to model of preferences which was presented in Section 7.4. Yet, it comprises an additional price component that distinguishes two different consumers in their general willingness to pay for the service and are more complex scoring function.

The non-functional preferences $\mathcal{P}$ are defined as the tuple $\mathcal{P} = (\alpha, \mathcal{S})$ with willingness to pay for a perfect service α and the QoS preferences represented by a scoring function $\mathcal{S} : A \to [0,1]$ of the form

$$\mathcal{S}(A) = \left(\sum_{l=1}^{L} \lambda_l \, \|a_l\| \right) \tag{8.1}$$

with L the number of QoS attributes, $\Lambda = (\lambda_1,...,\lambda_L)$ a vector of weights with $\sum \lambda_l = 1$, $A = (a_1,...,a_L)$ a vector containing the aggregated QoS attributes for service configuration alternative $\mathcal{A}$ and $\|a_l\|$ the normalization function for aggregated attribute a_l defined as

$$\|a_l\| = \begin{cases} 1 & \text{if } a_l \geq \gamma_l^T \\ \frac{a_l - \gamma_l^B}{\gamma_l^T - \gamma_l^B} & \text{if } a_l \in \left(\gamma_l^B, \gamma_l^T\right) \\ 0 & \text{if } a_l \leq \gamma_l^B \end{cases} \tag{8.2}$$

with γ_l^T, γ_l^B upper and lower boundaries for a_l. The chosen functional form assumes linearity between the boundaries parameters γ_l^T and γ_l^B. While other functional forms certainly could be more realistic, linearity is assumed for two reasons: (1) It simplifies preference elicitation on consumer side, as only two boundary parameters have to be obtained from the consumer, and (2) linearity reduces the problem complexity and allows for the usage of state of the art optimization approaches.

The mathematical form of the non-functional preferences $\mathcal{P}$ is given by the negotiation mechanism as presented above. Weights λ_l as well as boundary parameters γ_l^T and γ_l^B are assumed to be obtained from the consumer in her service request.

8.2.2 Solution Space

The solution space for describing feasible service compositions resulting from the service engineering algorithm is described in graph notation. We hereby rely on the same notion of a service configuration graph $\mathcal{G}$ (cf. Section 6.3.2), slightly adapted in its formulation to better fit in the context of optimization algorithms. Two different graph structures are distinguished, as the difference between them affects the nature of applicable optimization techniques. At first, a general concept of a service configuration graph allowing conjunctive edges is presented.

Definition 8.1 [Service Configuration Graph]. *The solution space which results from the service engineering algorithm is defined as service configuration graph $\mathcal{G} = (V, E)$, a directed, acyclic, graph with vertices $V = \{n_0, n_\infty\} \cup V^\bullet \cup V^+$ and edges E, defined as follows:*

- *$n_0 \in V$ is the source node of $\mathcal{G}$*
- *$n_\infty \in V$ is the sink node of $\mathcal{G}$*
- *nodes in $V^\bullet$ represent conjunctive dependencies, nodes in V^+ represent possible atomic service choices (disjunctive) for the preceding dependency*
- *there are at least one and at most M nodes $n^\bullet \in V^\bullet$ with edges $(n_0, n^\bullet) \in E$*

- *for each node $n^\bullet \in V^\bullet$ there are at least one and at most τ nodes $n^+ \in V^+$ with edges $(n^\bullet, n^+) \in E$*
- *for each node $n^+ \in V^+$ there are either at least one and at most M nodes $n^\bullet \in V^\bullet$ with edges $(n^+, n^\bullet) \in E$ or there is one edge $(n^+, n_\infty) \in E$*
- *N is the depth of the graph, defined as the number of nodes n^+ in the longest path from n_0 to n_∞*
- *$\mathcal{A} \subseteq V^+$ is a subset of nodes representing one feasible service configuration alternative of $\mathcal{G}$, i.e. a subgraph of $\mathcal{G}$ where the following condition holds $\exists! n^+ \in \mathcal{A} : (n^\bullet, n^+) \in E$, $\forall n^\bullet \in V^\bullet$*

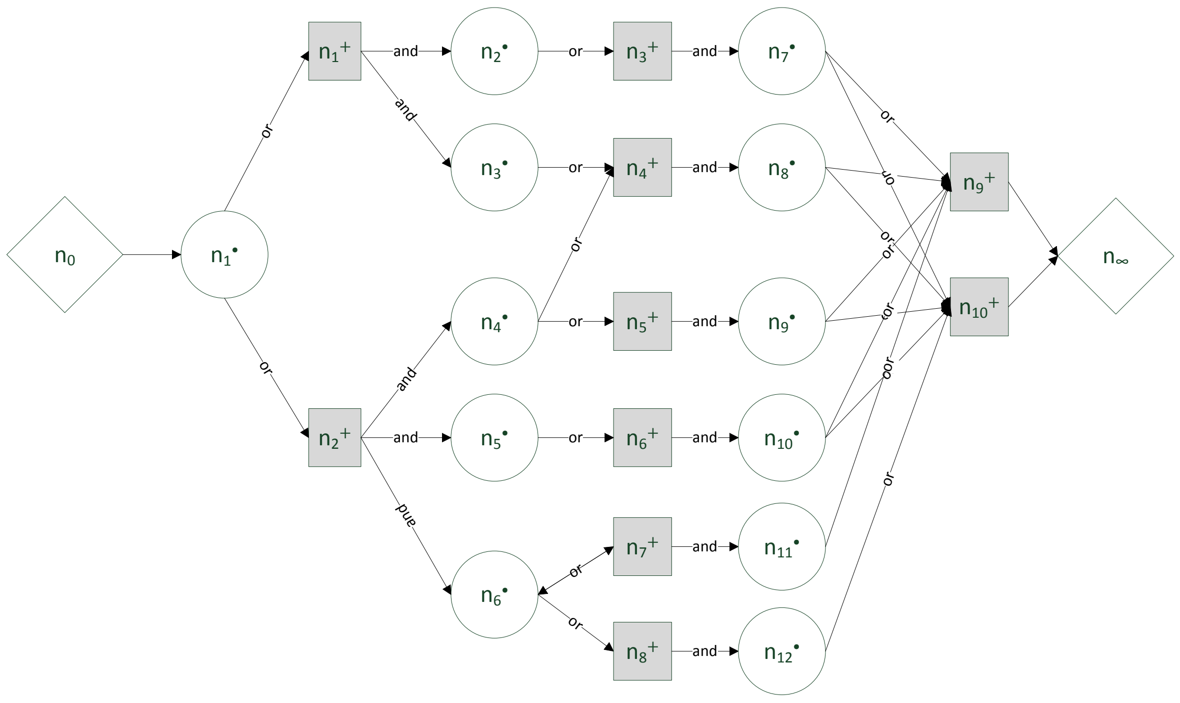
Through the nature of generic defined dependencies, the graph structure is built upon conjunctive and disjunctive edges, i.e. connections between nodes either have the semantics of *and* or *or*. Within this graph structure, feasible configurations are subgraphs of $\mathcal{G}$.

Limiting the underlying ontology to contain at most one dependency per service entity, a more simple graph structure containing only disjunctive edges is obtained. The graph is characterized by *or* connections only, with each path from source to sink representing one feasible configuration. As such, the structure is equivalent to the structure of an SVN graph (cf. Section 2.2.5).

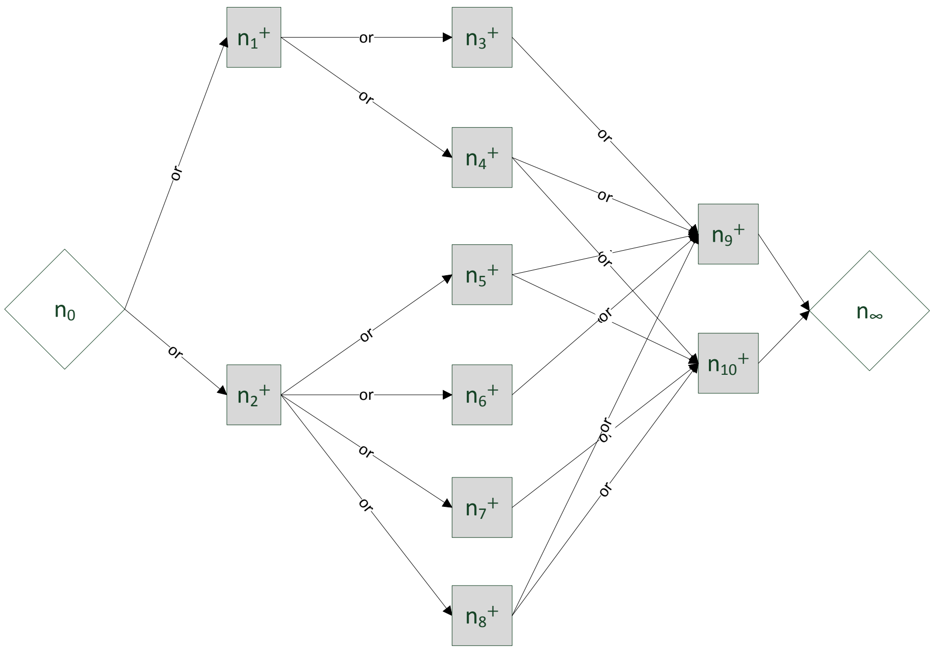
Definition 8.2 [Disjunctive Service Configuration Graph]. *The solution space resulting from the service engineering algorithm and limited to at most one dependency per service entity is defined as disjunctive service configuration graph $\mathcal{G}^+ = (V, E)$, a directed, acyclic, graph with vertices $V = \{n_0, n_\infty\} \cup V^+$ and edges E, defined as follows:*

- *$n_0 \in V$ is the source node of $\mathcal{G}^+$*
- *$n_\infty \in V$ is the sink node of $\mathcal{G}^+$*
- *nodes $n^+ \in V^+$ represent possible atomic service choices*
- *for each node $n^+ \in V^+$ there are either at least one and at most M nodes $\hat{n}^+ \in V^+$ with edges $(n^+, \hat{n}^+) \in E$ or there is one edge $(n^+, n_\infty) \in E$*
- *N is the depth of the graph, defined as the number of nodes n^+ in the longest path from n_0 to n_∞*
- *$\mathcal{A} \subseteq V^+$ is a subset of nodes representing one feasible service configuration alternative of $\mathcal{G}^+$, i.e. a path from n_0 to n_∞.*

The remainder of this chapter deals with solutions for both graph structures. Sample graphs of both variants are depicted in Figure 8.1. In (a), the depicted graph contains both conjunctive and disjunctive edges with $\mathcal{A} = \{n_1^+, n_3^+, n_4^+, n_9^+\}$ being one of the potential service configurations. Graph (b) comprises only *or* connections. In graphs of this nature, the usage of shortest-path algorithms becomes possible, which is not the case in the graph that also has conjunctive edges. $\mathcal{A} = \{n_1^+, n_3^+, n_9^+\}$ is an example path from source to sink, representing one potential service configuration in this type of graph.



(a) $\mathcal{G}$ with $M = 3$, $\tau = 2$, $N = 3$



(b) $\mathcal{G}^+$ with $\tau = 4$, $N = 3$

Figure 8.1: Example service configuration graphs

8.2.3 Aggregation Functions

Each QoS attribute that is contained in the scoring function $\mathcal{S}$ needs to be aggregated, i.e. the perceived QoS value of the entire service configuration has to be calculated based on the individual values of each atomic service resource comprised in the configuration alternative. The aggregation function is defined as

$$a_l = \bigoplus_{n \in \mathcal{A}} q_n^l \tag{8.3}$$

with q_n^l being the l^{th} QoS attribute of resource node n. Depending on the type of attribute l, the aggregation operation $\bigoplus$ is characterized by different mathematical operators. The selection of attributes under consideration is restrained to the three most important QoS attributes most often mentioned in literature, *response time* ($\sum$), *availability* ($\prod$), and *throughput* (*min*), additionally adding *error rate* (*max*) and *encryption* ($\wedge$) in order to consider two further common operators.

8.2.4 Objective Function

The following functional term is used as objective function to the optimization problem

$$U(\mathcal{A}) = \alpha \mathcal{S}(A) - C(\mathcal{A}) \tag{8.4}$$

with cost function

$$C(\mathcal{A}) = \sum_{n \in \mathcal{A}} c_n \tag{8.5}$$

and c_n the costs of component node n. It resembles a monetarized utility that is obtained by multiplying the normalized scoring function with the maximum willingness to pay α, minus the accruing costs for service configuration alternative $\mathcal{A}$.

8.2.5 Challenges

In the generic form of the service configuration graph, conjunctive edges prevent the use of well-known efficient graph algorithms. Hence, it is a challenge to find an optimization technique which is able to find the optimal subgraph representing the best available and feasible configuration, without having to enumerate all feasible configurations in a brute-force attempt.

Regarding the more simple disjunctive graph structure, a property stated by Bellman [15] becomes important: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." This policy, a necessary condition for optimality of dynamic programming approaches, is known as Bellman's principle of optimality (cf. Section 2.4.4.1). In any shortest-path graph optimization problem with an objective function fulfilling the Bellman principle, the optimal path can be computed efficiently using well-known algorithms like Dijkstra or the Bellman-Ford algorithm. This is due to the fact, that under the Bellman condition, suboptimal paths of a graph can never be part of the optimal path and thus can be cut off from the remaining search space.

Any computational challenge for finding the optimal path in a disjunctive service configuration graph $\mathcal{G}^+$, consisting of *or*-labeled edges only, arises from a lack of the Bellman optimality in the objective function. It turns out, that this also holds for the more complex case with conjunctive edges, where optimal sub graphs rather than paths have to be found.

The Bellman optimality potentially is violated at two different spots in the present model. The first violation occurs in the application of the normalization function, when dealing with more than one QoS attribute and aggregated attributes potentially cross upper or lower boundaries. The following example illustrates this violation.

Example 8.1. *Two attributes ($L = 2$) with aggregation function $\sum$, lower and upper boundaries $\gamma_l^T = 0, \gamma_l^B = 10$, both weighted equally ($\lambda_1 = \lambda_2 = 0.5$). Now consider a node n with two incoming paths. Path one with aggregated attribute values $a_1 = 2, a_2 = 3$ and path two with $a_1 = 1, a_2 = 9$. It is easy to see, that path one is superior to path two, as one obtains a score of* $0.5(2-10)/(0-10) + 0.5(3-10)/(0-10) = 0.75$*, while path two has a lower score of* $0.5(1-10)/(0-10) + 0.5(9-10)/(0-10) = 0.5$*. In any optimization approach with Bellman optimality, path two could be neglected, as any suboptimal sub path cannot be part of the optimal path. With node n adding attribute values $q_1 = 1$, $q_2 = 7$, the score of both paths are not lowered equally due to the effect of the normalization function's boundaries. Path one now has a score of* $0.5(3-10)/(0-10) + 0.5(10-10)/(0-10) = 0.35$*, while path two scores* $0.5(2-10)/(0-10) + 0.5(10-10)/(0-10) = 0.4$*. a_2 becomes irrelevant, as for both paths, the lower boundary is reached (path one) or exceeded (path two) – a_2 contributing a partial score of 0 for both paths. Path two (neglected in any Bellman optimization) suddenly is superior to path one, as its partial score for a_1 is higher.*

With other words, whenever the objective function contains elements, that possibly change the rank order of preceding paths, Bellman-optimality-based algorithms cannot guarantee optimality. This is also the case with aggregation functions like $\wedge$ and $\prod$, as these functions do not guarantee strictly monotone behavior. However, similar Bellman-like behavior can be observed for attributes like *availability*, which we will call "quasi-Bellman" behavior from here on. Despite the fact, that $\prod$ has to be used as aggregation function, the value domain is usually restricted to values above 99%, implying strictly monotone behavior while aggregating availability values.

After all, eight different settings can be characterized which are typically found in the domain of our interest, posing different challenges on the optimization technique. The first challenge arises in settings with the generic form of a service configuration graph, where the conjunctive semantic of *AND* connections massively increases the combinatorial complexity of the solution space.

Setting 8.1 [Conjunctive edges]. *In the generic form of a service configuration graph, conjunctive edges with* ***AND*** *semantic prevent the usage of well known graph algorithm, as not paths, but subgraphs represent feasible configurations.*

Further challenges arise from the different forms of QoS aggregation functions, which are required for aggregating the various types of QoS attributes. One can distinguish between three classes of attributes.

Setting 8.2 [Only Bellman attributes]. *Any setting with only attributes following a strictly monotonous aggregation function, not violating the Bellman property.*

In the following, attributes that show "quasi-Bellman" behavior are denoted as *quasi-Bellman* attributes.

Setting 8.3 [Quasi-Bellman attributes]. *Quasi-Bellman attributes require an aggregation function that violates the Bellman property when computed exactly. Nevertheless, certain value domains, as in the case of the QoS attribute availability, allow heuristic approaches yielding a very low expected error.*

Setting 8.4 [Non-Bellman attributes]. *Comprises all other attributes that require non-Bellman aggregation functions, but do not approximately behave Bellman-like (e.g. multiplicative attributes with a wide value range or Boolean attributes).*

Lastly, the normalization function can bring along a further violation of the Bellman property, if upper or lower boundaries are surpassed (cf. Example 8.1).

Setting 8.5 [Boundary violation]. *Any boundary violation in the normalization function also leads to a violation of the Bellman property in the optimization.*

The approaches presented later in this chapter address the challenges contained in these settings. They are also evaluated in different time criticality contexts – offline, online and time critical.

Setting 8.6 [Offline]. *The optimization is run in an offline scenario, for example over night, without runtime being of any great importance. Formally, we require runtime to be below 12 hours.*

Setting 8.7 [Online]. *The optimization is run in an online scenario, for example in an online order process, i.e. results have to be obtained within a reasonable waiting time at least for reasonable problem instances. Formally, we require runtime to be below 10 seconds.*

Setting 8.8 [Runtime Critical]. *Runtime is the most critical factor, for example, if thousands of simultaneous requests have to be processed and any lag is associated with high costs. Formally, we require runtime to be below 100 milliseconds.*

8.3 Optimization Techniques

This section describes different approaches for tackling the challenge of finding optimal configurations within a service configuration graph. The presented approaches have been designed to meet the challenges presented in Section 8.2.5. An evaluation on both their runtime and solution quality follows in Section 8.4. The section begins with exact procedures, followed by a description of approximative solutions dealing with quasi-Bellman attributes. Lastly, heuristic solutions allowing an efficient computation in non-Bellman-optimality scenarios are presented.

8.3.1 Exact Approaches

In the exact approaches, exactness of the obtained objective function value and optimality of the solution are guaranteed. However, depending on the problem structure, optimality comes at the price of soaring computation time. The presented brute-force algorithms – exploring the complete solution space – serve as benchmark for all other approaches, as they not only are exact, but also can cope with all challenges of Section 8.2.5. Dijkstra's algorithm and the BIP formulation offer a massively improved runtime performance, however, they are only exact for a subset of all settings.

8.3.1.1 Brute-Force-Search

Brute-force algorithms iterate through each and every possible solution of the optimization problem. For a disjunctive service configuration graph $\mathcal{G}^+$, a simple depth-first-search algorithm can be implemented (Algorithms 8.1 and 8.2). The implementation for the graph with conjunctive edges is more sophisticated, due to the increased solution complexity obtained from the *and* connections. Algorithms 8.3 and 8.4 show an approach that uses a stack S to store unvisited conjunctive nodes.[2]

Algorithm 8.1 Initialize Depth-First-Search $\mathcal{G}^+$

```
Require: graph $\mathcal{G}^+$, objective function $U$
Ensure: nodes $\mathcal{A}^*$ yielding the highest possible utility $u^*$
  $u^* \leftarrow -\infty$, $\mathcal{A} \leftarrow \emptyset$, $\mathcal{A}^* \leftarrow \emptyset$
  for all $n \in V$ with $(n_0, n) \in E$ do
    depthFirstSearchDisjunctive($n$, $\mathcal{G}$, $U$, $\mathcal{A}$, $\mathcal{A}^*$, $u^*$)
  end for
```

8.3.1.2 Dijkstra

Dijkstra's algorithm (cf. Section 2.4.4.2) is a single-source shortest-path algorithm for graphs with nonnegative edge path costs. It has a runtime of $\mathcal{O}(|V^2|)$, which

[2]Note that the presented pseudo algorithms do not claim to be most efficient.

Algorithm 8.2 depthFirstSearchDisjunctive(n, $\mathcal{G}^+$, U, $\mathcal{A}$, $\mathcal{A}^*$, u^*) – Recursive Depth-First-Search $\mathcal{G}^+$

```
Require: node $n$, graph $\mathcal{G}^+$, objective Function $U$, path $\mathcal{A}$, optimal path $\mathcal{A}^*$, maximum utility $u^*$
Ensure: path $\mathcal{A}^*$ yielding the highest possible utility $u^*$
  if $n \neq n_\infty$ then
    $\mathcal{A} \leftarrow \mathcal{A} \cap n$
    for all $n' \in V$ with $(n, n') \in E$ do
      depthFirstSearchDisjunctive($n'$, $\mathcal{G}$, $U$, $\mathcal{A}$, $\mathcal{A}^*$, $u^*$)
    end for
  else
    $u \leftarrow U(\mathcal{A})$
    if $u > u^*$ then
      $u^* \leftarrow u$
      $\mathcal{A}^* \leftarrow \mathcal{A}$
    end if
  end if
```

Algorithm 8.3 Initialize Depth-First-Search $\mathcal{G}$

```
Require: Graph $\mathcal{G}$, Objective Function $U$
Ensure: Nodes $\mathcal{A}^*$ yielding the highest possible utility $u^*$
  $u^* \leftarrow -\infty$, $\mathcal{A} \leftarrow \emptyset$, $\mathcal{A}^* \leftarrow \emptyset$, $S \leftarrow \emptyset$
  for all $n^\bullet \in V^\bullet$ with $(n_0, n^\bullet) \in E$ do
    push($S$, $n^\bullet$)
    depthFirstSearch($\mathcal{G}$, $U$, $S$, $\mathcal{A}$, $\mathcal{A}^*$, $u^*$)
  end for
```

can be further improved to $\mathcal{O}(|V| \log |V| + |E|)$ by optimized data structures, making Dijkstra's algorithm one of the most efficient single-source shortest-path algorithm available [50]. Being based on Bellman's principle of optimality, however, a prerequisite for it to work is a strictly monotonous objective function. This is the case within its typical field of application as routing algorithm. For the problem of finding optimal configuration paths in a disjunctive service configuration graph, this implies two restrictions on the objective function for it to guarantee optimality: (1) The aggregation function has to be of strictly monotonous nature, and (2) lower and upper boundaries of scoring function $\mathcal{S}$ cannot be exceeded, as exceeded boundaries also violate strict monotonicity. Regarding the first restriction, Dijkstra's algorithm ensures optimality only for additive QoS attributes. By nature, Dijkstra's algorithm cannot be used for service configuration graphs with conjunctive edges.

8.3.1.3 Binary Integer Program

Despite the graph notation for the service configuration problem in this chapter, finding the best feasible service composition alternative can also be solved using a binary integer programming (BIP) approach (cf. Section 2.4.3).

Algorithm 8.4 depthFirstSearch($\mathcal{G}$, U, S, $\mathcal{A}$, $\mathcal{A}^*$, u^*) – Recursive Depth-First-Search $\mathcal{G}$

```
Require: graph $\mathcal{G}$, objective function $U$, stack $S$, subgraph $\mathcal{A}$, optimal subgraph $\mathcal{A}^*$,
  maximum utility $u^*$
Ensure: recursively adding nodes to $\mathcal{A}$ yielding the highest possible utility $u^*$
  if $S \neq \emptyset$ then
    $n^\bullet \leftarrow$ pop($S$)
    for all $n^+ \in V^+$ with $(n^\bullet, n^+) \in E$ do
      if $n^+ \notin A$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cap n$
        for all $n^\bullet \in V^\bullet$ with $(n^+, n^\bullet) \in E$ do
          push($S$, $n^\bullet$)
        end for
      end if
      depthFirstSearch($\mathcal{G}$, $U$, $S$, $\mathcal{A}$, $\mathcal{A}^*$, $u^*$)
    end for
  else
    if $u > u^*$ then
      $u^* \leftarrow u$
      $\mathcal{A}^* \leftarrow \mathcal{A}$
    end if
  end if
```

Objective Function. The objective function presented in Section 8.2.4 has to be adapted to meet the prerequisites of BIP solvers. As current state-of-the-art integer programming algorithms[3] only support linear or quadratic forms, the objective function has to be linear or at most of quadratic nature. Through introducing binary decision variables $x \in \mathcal{X}$ with value domain $\{0,1\}$, for each node $n \in V$ representing a service resource, one obtains the following binary optimization problem:

$$\max_{x \in \mathcal{X}} \alpha \cdot S(\mathcal{X}) - C(\mathcal{X}) \tag{8.6}$$

Cost Function. The BIP formulation of the cost function is straight forward using

$$C(\mathcal{X}) = \sum_{x \in \mathcal{X}} c \cdot x \tag{8.7}$$

Modeling the scoring function $\mathcal{S}$ is more sophisticated. The piecewise defined normalization function can be reformulated using both *min* and *max* function:

$$\min(\max(\frac{a_l - \gamma_l^B}{\gamma_l^T - \gamma_l^B}, 0), 1) \tag{8.8}$$

The obtained *minimax* objective can be transformed by adding additional decision variables and constraints ensuring minimality and maximality, respectively.[4]

[3]As implemented in industry standard solving engines like CPLEX or Gurobi (cf. Section 2.4.2).

[4]Solvers like CPLEX support *min* and *max* functions by performing the required transformations (additional variables and constraints) automatically, potentially increasing the BIP formulation's computational complexity.

Aggregation Function. Regarding the aggregation functions (cf. Section 8.2.3), the aggregation operators $\sum$, *min*, *max* and $\wedge$ can be formulated within a linear integer program. Additive QoS attributes are formulated analogously to the cost function. *min* and *max* functions are more tricky, as QoS attributes are multiplied with decision variables, which requires a transformation of all QoS values to be below zero or above one, respectively. Using transformation value $t = \max(q_1,...,q_n)$, the *min* function is transformed by

$$\min((q_1 - t) \cdot x_1,...,(q_n - t) \cdot x_n) + t \tag{8.9}$$

and the *max* function, analogously. Linearization is then achieved using the same transformation as described for the normalization function. Boolean QoS attributes like encryption can be aggregated through the term

$$1 - \max(x_1 \cdot (1 - q_1),...,x_n \cdot (1 - q_n)) \tag{8.10}$$

When dealing with more than two decision variables, multiplicative QoS attributes cannot be formulated within a binary integer program, as the $\prod$ function would lead to a nonlinear and nonquadratic objective function. However, quasi-Bellman attributes like *availability* can be approximated using the approaches presented in the succeeding Section 8.3.2.

Graph Structure. The graph structure of the solution space is formulated by means of constraints to the BIP. Constraints can be derived in a standardized manner from the graph. For a disjunctive service configuration graph with disjunctive edges only, the term

$$\sum_i x_i = 1 \quad \forall i : (n_0, n_i) \in E \tag{8.11}$$

constrains all nodes connected to the source, while

$$\sum_j x_j \geq x_i \quad \forall i,j : (n_i, n_j) \in E \tag{8.12}$$

constrains all other nodes. A major advantage of a BIP formulation over Dijkstra's algorithm is its easy adaptivity to graphs with conjunctive edges. Nodes connected to the source are constrained as above. All other nodes are modeled by means of the constraint formulation:

$$\sum_k x_k \geq \sum_i x_i \quad \forall i,j,k : (n_i^+, n_j^\bullet) \in E \wedge (n_j^\bullet, n_k^+) \in E \tag{8.13}$$

Further, the BIP formulation can be extended by adding additional constraints, e.g. excluding certain undesired resource combinations.

8.3.2 Approximations

Apart from the brute-force algorithms, neither of the exact mechanisms can solve the optimization when dealing with multiplicative QoS attributes. However, QoS attributes with very limited value domain can show quasi-Bellman behavior. With values typically ranging from 0.99 to 0.999, this is especially true for *availability* – one of the most important QoS attributes in the IT service domain. Despite the multiplicative nature of this attribute, two different linear approximations proposed later in this section allow its inclusion into linear programming approaches.[5]

8.3.2.1 Subtractive Approximation

To achieve an efficient computation (and for aggregating quasi-Bellman attributes in a BIP formulation), a linear approximation can be of great benefit. Recapturing the typical characteristics of QoS attribute *availability* (and others with similar value domain) that are aggregated by multiplication, the linear approximation function in the following offers computational efficiency. Simultaneously, it only yields a small error, depending on the concrete problem instantiation:

$$a_l = 1 - \sum_{n \in \mathcal{A}} \delta \cdot (1 - q_n) \tag{8.14}$$

Parameter δ can be used to parametrize the function according to the domain of attribute values, which allows to reduce the expected error of the approximation function. Values for δ that achieve a minimum expected error typically lie within the unit interval, as shown experimentally later in Section 8.4.1.

8.3.2.2 Logarithmic Approximation

A second approximation can be achieved by means of the logarithm function. Hereby, true values of the QoS attribute are replaced by their logarithmic values. The $\sum$ operator is then used instead of the $\prod$ operator. For back-transformation, the logarithm is applied on the upper and lower boundaries of the normalization function. However, the resulting value is not exact, as these steps are no mathematically equivalent transformations. The following linear term is obtained[6]

$$\|a_l\| = \frac{\sum_{n \in \mathcal{A}} \ln(q_n) - \ln(\gamma_l^B)}{\ln(\gamma_l^T) - \ln(\gamma_l^B)} \tag{8.15}$$

with all logarithmic values pre-computed.

8.3.3 Heuristics

Based on the considerations prior in this section and the approximations from above, several heuristics can be derived addressing the problem of quasi-Bellman attributes as described in Setting 8.3.

[5]Note that the formulation of the approximation functions in the following looks slightly different in a binary integer program, as the aggregation function in such a program must iterate over all decision variables, while multiplying the QoS value with the corresponding decision variable $x \in \mathcal{X}$. For readability reasons, the graph notation is therefore chosen, iterating only over the set $\mathcal{A}$ representing one feasible service configuration alternative.

[6]For a better readability, parts of the piecewise function above/below the upper/lower boundary are omitted.

8.3.3.1 Dijkstra*

The first heuristic introduced in this thesis will be called *Dijkstra**. Dijkstra* is not exactly a heuristic in a sense, that it is different from the exact algorithm. Yet, this notation is used whenever Dijkstra's algorithm is applied despite the violation of Bellman's principle of optimality. This violation occurs when cutting off paths in the graph, that are dominated by a higher valued path from a local perspective and could become optimal later on while iterating through the graph, due to some non-monotonous aggregation function.

8.3.3.2 Dijkstra with Approximation

For the quasi-Bellman attribute *availability*, Dijkstra's algorithm can be applied using the approximations described in Section 8.3.2. Its application will lead to no additional error on top of the error originating from the used approximation function, unless strict monotonicity is violated by another aggregation function or through crossing the normalization function's boundaries.

8.3.3.3 Binary Integer Program with Approximation

Through the linearity of the two approximation functions presented above, the BIP approach is also applicable to settings with multiplicative quasi-Bellman-like attributes. As with the above mentioned Dijkstra-based heuristic, the inexactness of the optimization originates from the error of the approximation function.

8.4 Evaluation

So far, different approaches to tackle the challenge of efficiently computing optimal service configurations in different settings were introduced. This section covers both efficiency, i.e. runtime of the presented approaches, and effectiveness, i.e. the solution quality, under different circumstances.

The expected error of the subtractive approximation is evaluated context-free, i.e. independent of the optimization model, both analytically and numerically in Section 8.4.1. As the other approximations and all other techniques and heuristics cannot be evaluated independently of a given scoring function and service configuration graph, a simulation study is conducted as evaluation, following in Section 8.4.2.

8.4.1 Subtractive Approximation Error

As measure for the obtained approximation error the following function is defined

$$e(a^*,\hat{a}) = \frac{|a^* - \hat{a}|}{a^*} \tag{8.16}$$

with a^* being the exact aggregation value and $\hat{a}$ the approximated one.[7] Based on this error function, various sensitivity analyses are conducted with different parameter settings. For reducing the mathematical complexity and a better understanding of the reader, δ is set to a value of one for most parts of the evaluation, as one can draw the same conclusions without losing generality. A simplified aggregation function with δ equaling one is obtained

$$a = 1 - N + \sum_{n \in \mathcal{A}} q_n \tag{8.17}$$

with $N = |\mathcal{A}|$ the number of service resource nodes in $\mathcal{A}$.

8.4.1.1 Deterministic Values

For a first analytic error estimation we assume the attribute values to be deterministic and parameter δ to be one. Thus the error can be calculated straightforward without the use of stochastics. If all attribute values are equal, i.e. $\forall n, n' \in \mathcal{A} : q_n = q_{n'} = q$, a simplified equation is obtained:

$$e(a^*, \hat{a}) = \frac{|q^N - 1 + N - N \cdot q|}{q^N} \tag{8.18}$$

The resulting error as a function of N and q is depicted in Figure 8.3a. One can see that the error increases with N and decreases with q. However, in reality, not all values are equal. One way to measure this effect in a deterministic manner, is by spreading the N attribute values equally in the interval $[q - \epsilon, q + \epsilon]$, with

$$\begin{aligned} q_1 &= q - \epsilon \\ &\vdots \\ q_N &= q + \epsilon \end{aligned}$$

The following error function can be derived, showing the influence of the values being spread apart by two times ϵ:

$$e(a^*, \hat{a}) = \frac{\left| \prod_{i=1}^{N} \left(q + \epsilon \left(\frac{2 \cdot i - 2}{N-1} - 1 \right) \right) - 1 + N - N \cdot q - \epsilon \sum_{i=1}^{N} \left(\frac{2 \cdot i - 2}{N-1} - 1 \right) \right|}{\prod_{i=1}^{N} \left(q + \epsilon \left(\frac{2 \cdot i - 2}{N-1} - 1 \right) \right)} \tag{8.19}$$

Looking at Figure 8.2, the error depending on q and ϵ can be analyzed. For q values below one and starting from $\epsilon = 0$, one can notice that the error of the approximation first decreases in ϵ. When a certain value of ϵ is reached, the approximation evaluates to the exact value obtained by the product of all aggregated values, i.e. approximation and exact value are equal. After passing this point, the error increases again. The exact location of this root depends on the number of aggregated

[7]Note that index l for the l's attribute of $\mathcal{S}$ is omitted in this Section.

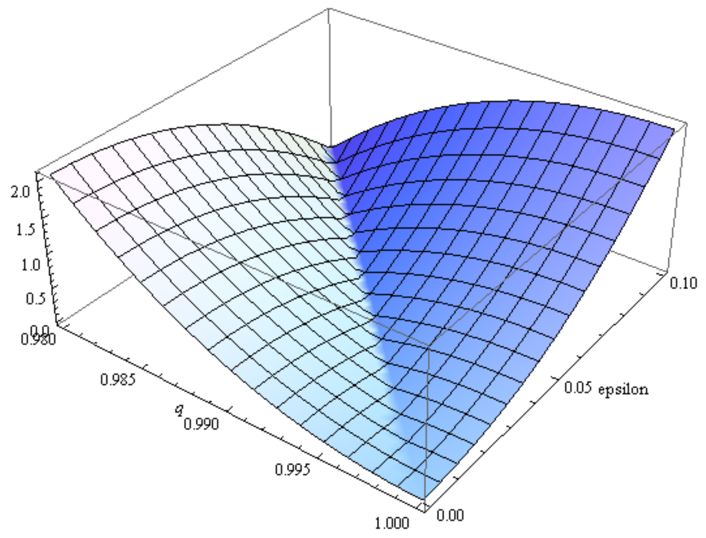


Figure 8.2: Deterministic error of subtractive approximation for spread values depending on ϵ [%]

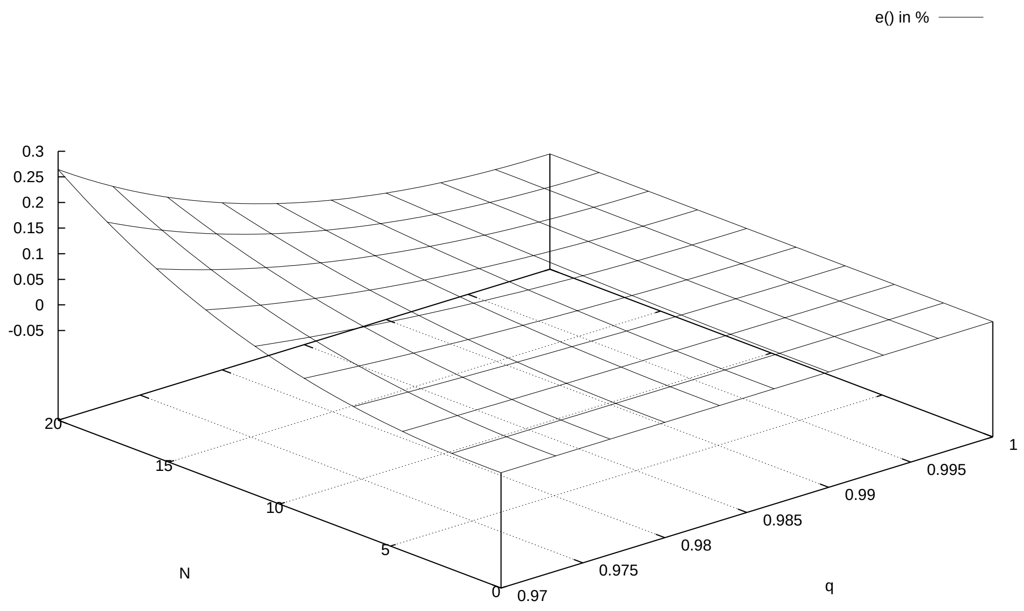
attributes N and their mean value q. The location of the root forms a valley where a decreasing value of q corresponds to an increasing value of ϵ. From a practical point of view, this is an advantageous property, as realistic values commonly "fall" into this valley, i.e. q-values are below one and spread within a certain interval.

8.4.1.2 Normalization Function

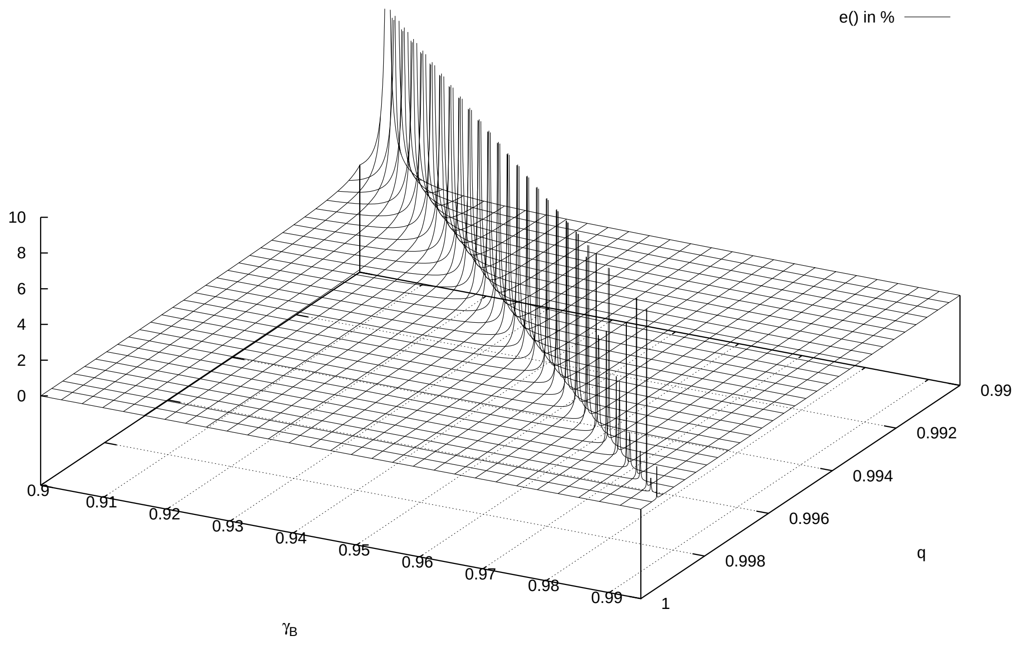
In the context of a normalization function, as presented in Section 8.2.1, the perceived error increases. This is due to the augmentation effect of the normalization, scaling up small differences in the attribute value, thus also raising the expected error. For $q \in [\gamma_l^B, \gamma_l^T]$ the deterministic error equals

$$e(\|a^*\|, \|\hat{a}\|) = \frac{\left| \frac{1-\gamma_l^B - N - N\cdot q}{\gamma_l^T - \gamma_l^B} - \frac{q^N - \gamma_l^B}{\gamma_l^T - \gamma_l^B} \right|}{\frac{q^N - \gamma_l^B}{\gamma_l^T - \gamma_l^B}} = \frac{|1 - N + N \cdot q - q^N|}{q^N - \gamma_l^B} \tag{8.20}$$

Obviously, the error is actually independent of the upper boundary γ_l^T, yet requires a limit analysis where a^* is close to γ_l^B. Figure 8.3b shows the error in dependency on γ_l^B and q. In most cases, it is not much higher than the error without using a scoring function, yet with a close proximity of q^N to γ_l^B, the error rises towards infinity. This implies that the proposed approximation function is not suitable, where the exact value of the aggregation function is very close to the lower boundary of the scoring function. However, from a practical point of view, this case can be considered unlikely, unless quality expectations are very high compared to the available configurations.



(a) Equal q values without normalization



(b) Equal q values with normalization, $N = 10$

Figure 8.3: Deterministic error of subtractive approximation [%]

8.4.1.3 Stochastic Values

Up to now, evaluation results were relying on deterministic values for the QoS attributes. In reality one is confronted with values that origin from some stochastic or random process with an underlying probability distribution[8]. Nevertheless, the prior deterministic considerations allowed for some much simpler analytic conclusions, that become far more complex in the stochastic case. As more sophisticated analytic considerations require a numerical integral evaluation with an increasing complexity in the number of variables, Monte Carlo simulations are used for computing error values for larger values of N.

In the center of interest is again the error arising from using an approximation instead of the exact multiplication function. In the stochastic case this error is not a deterministic value, but a probability function. Thereby, the expected value of this function resembles a comparable result to the deterministic error value.

This expected error, which is based on N independently distributed quality values, can be calculated by building the integral over all random variables. To do so, density functions of all random variables have to be multiplied (i.e. the actual probability of obtaining a given value) with the error that results from the given set of random variable instantiations. With $f_i(q_i)$ being the probability density function of variable q_i, the following equation is obtained

$$E(e(a^*,\hat{a})) = \int_{\dots} \int f_1(q_1) \cdot \dots \cdot f_N(q_N) \frac{|a^* - \hat{a}|}{a^*} dq_1 \dots dq_N \tag{8.21}$$

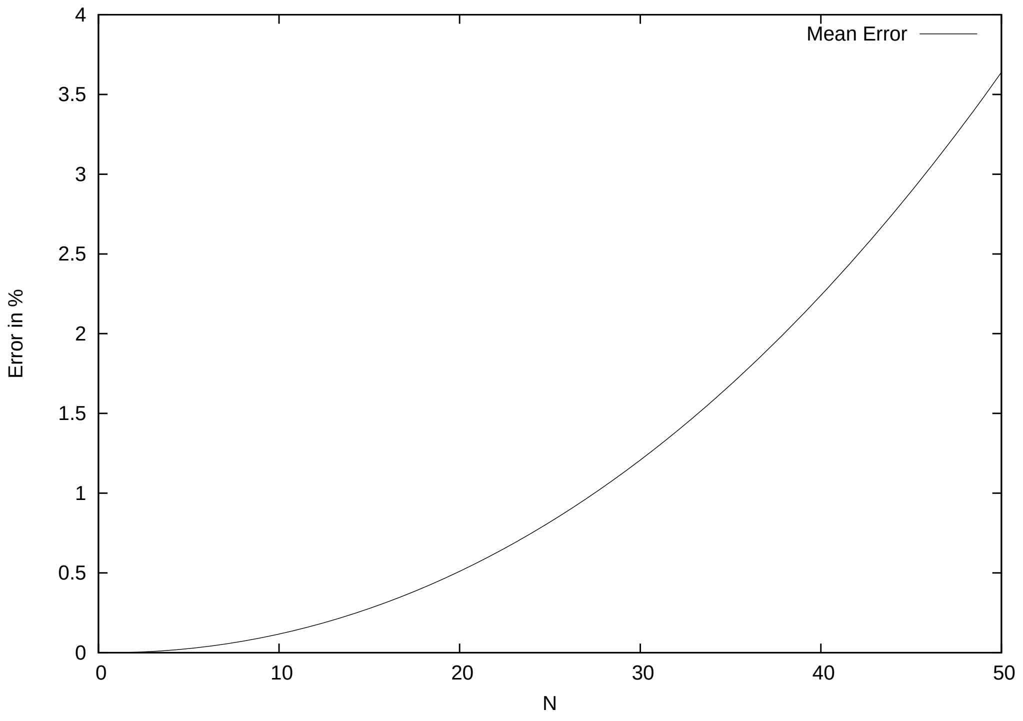
for calculating the expected error. Table 8.1 gives an overview on concrete values – (a) contains values for two and three normally distributed variables[9] with different distribution parameters μ and σ^2, (b) shows values for two to five uniformly distributed variables with different parameters for the distribution $\mathcal{U}(a,b)$. All error values are calculated by means of numerical integration. For large values of μ or a,b, respectively, i.e. quality values above 99%, the expected error is well below or around 1%. Concerning the QoS attribute *availability*, such a result seems acceptable, given that *availability* values in most Cloud computing SLAs are commonly around or above 99% (e.g. Amazon EC2 99.95%, IBM SmartCloud Enterprise+ 98.5% for Bronze to 99.9% for Platin, ...).

As a further evaluation, a Monte Carlo simulation study to expose the relationship of mean value, variance and parameter δ on the expected error was conducted. All simulations are based on the average of 100,000 runs. The study reveals, that the error slightly increases with the variance σ^2 and decreases with the mean value μ, in accordance with Table 8.1. Figure 8.4a, shows the influence of a growing graph structure $\mathcal{G}$, i.e. an increasing value of N for a typical setting with a mean value of 99.5% and a variance of 0.001.

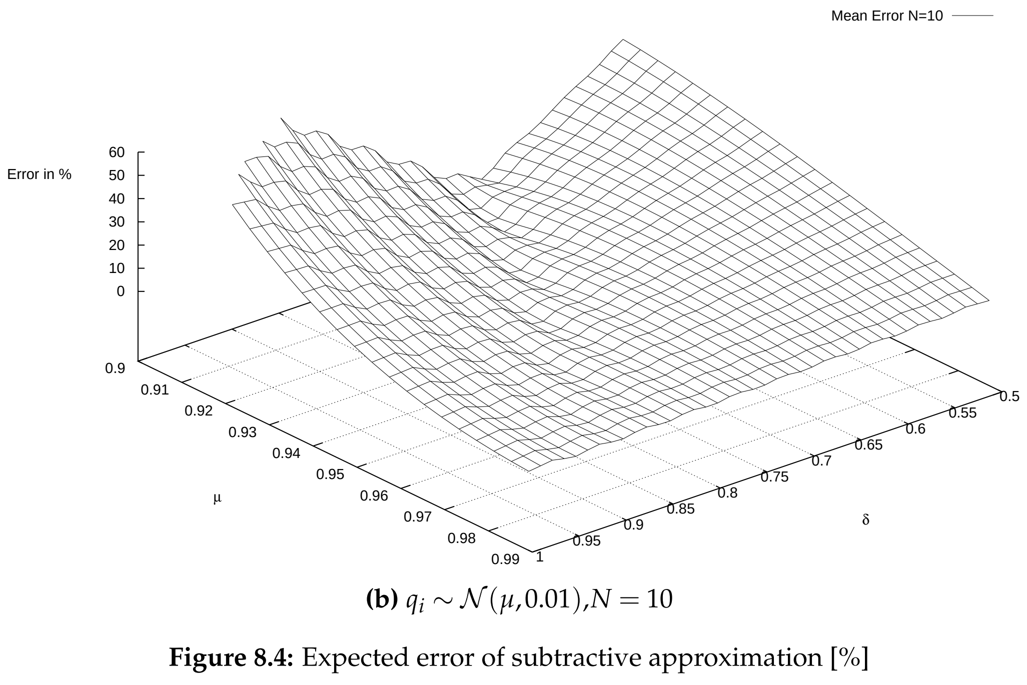
It remains open, whether the approximation can be improved by adjusting parameter δ. Figure 8.4b shows its influence on the expected error depending on mean

[8]For evaluating the expected error when coping with stochastic processes, the probability distributions are assumed to be known.

[9]Note that normally distributed values for availability can exceed a value of one, which has been neglected in the considerations, but only has a small influence on results. Future work will address this issue by using a beta distribution instead.



(a) $q_i \sim \mathcal{N}(0.995, 0.001)$



(b) $q_i \sim \mathcal{N}(\mu, 0.01), N = 10$

Figure 8.4: Expected error of subtractive approximation [%]

N	μ	σ^2	$\mathbf{E(e(a^*,\hat{a}))}$
2	0.9	0.1	1.979
2	0.9	0.01	1.238
2	0.99	0.1	0.698
2	0.99	0.01	0.014
2	0.999	0.1	0.666
2	0.999	0.01	0.006
3	0.9	0.1	5.560
3	0.9	0.01	3.988
3	0.99	0.1	1.296
3	0.99	0.05	0.317
3	0.999	0.1	1.216
3	0.999	0.01	0.282

(a) $q_i \sim \mathcal{N}(\mu,\sigma^2)$

N	a	b	$\mathbf{E(e(a^*,\hat{a}))}$
2	0.9	0.99	0.348
2	0.99	0.999	0.003
3	0.9	0.99	1.085
3	0.99	0.999	0.009
4	0.9	0.99	2.257
4	0.99	0.999	0.019
5	0.9	0.99	3.911
5	0.99	0.999	0.031

(b) $q_i \sim \mathcal{U}(a,b)$

Table 8.1: Expected error of subtractive approximation [%]

value μ. In fact, parameter δ can be effectively used to reduce the error even for low mean values, by lowering δ when QoS attributes are low as well.

An extensive evaluation on the effective performance of this approximation in the context of service configuration graphs will follow in Section 8.4.2.2.

8.4.2 Simulation Study

For evaluating the different approaches within the context of the custom service process, a simulation-based numerical evaluation is chosen for two reasons: (1) To ensure comparability over all scenarios, challenges and approaches, and (2) as analytically predicting the runtime of an optimization problem formulated as binary integer program is hardly feasible. The simulations are based on Java implementations for both algorithms and BIP formulations, with CPLEX (cf. Section 2.4.5) as solver for the BIP without any optimizations of the solver's out-of-the-box parameters.

In all simulations, both non-functional preferences $\mathcal{P}$ and graph $\mathcal{G}$ are randomly generated in each round. With $L = 2$, *availability* and *response time* serve as exemplary QoS attributes. Each node is assigned with random values drawn from normal distribution $\mathcal{N}(0.995, 0.005)$ for *availability* and from normal distribution $\mathcal{N}(15,5)$ for *response time*. Concerning $\mathcal{S}$, weights $\lambda \in \Lambda$ are randomized, while the normalization boundaries are constant values of a function. The function ensures a reasonable relationship of the boundaries to the parameters' mean value, standard deviation and boundary tightness ϕ – with a higher value of ϕ leading to a tighter interval $[\gamma_l^B, \gamma_l^T]$. Further parameters describing the structure of the input graph $\mathcal{G}$ are de-

Parameter	Default Value	Description
τ	3	Number of n^+-nodes per $n^\bullet$-node/hierarchy
N	3	Depth of the graph from n_0 to n_∞
β	2/3	Connectivity (ratio)
ϕ	1	Tightness of the normalization boundaries with $\gamma_l^B, \gamma_l^T = \mu \pm (\sigma/\phi)$
L	2	Number of QoS attributes

Table 8.2: Simulation parameters

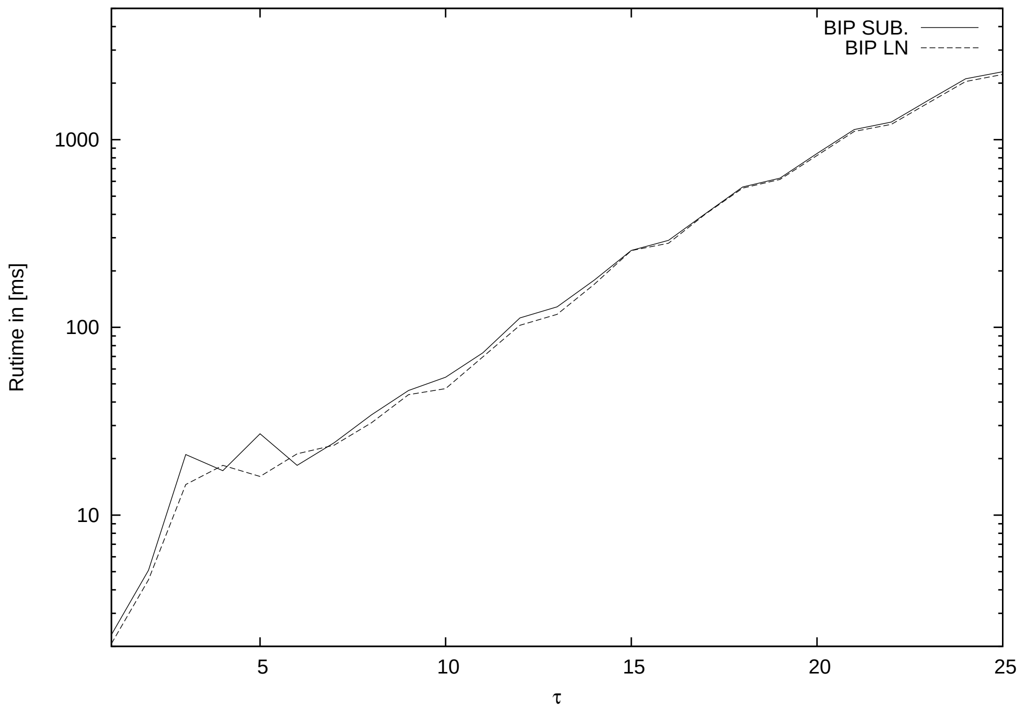
picted in Table 8.2. Apart from β, all parameters originate from the the optimization model in Section 8.2. β denotes the connectivity of the graph, i.e. the ratio between actual and maximum number of connections between two dependency layers. All runtime simulations are conducted for both types of graph, $\mathcal{G}^+$ and $\mathcal{G}$, whereas solution quality was evaluated on disjunctive service configuration graphs only, without loss of generality.

8.4.2.1 Runtime

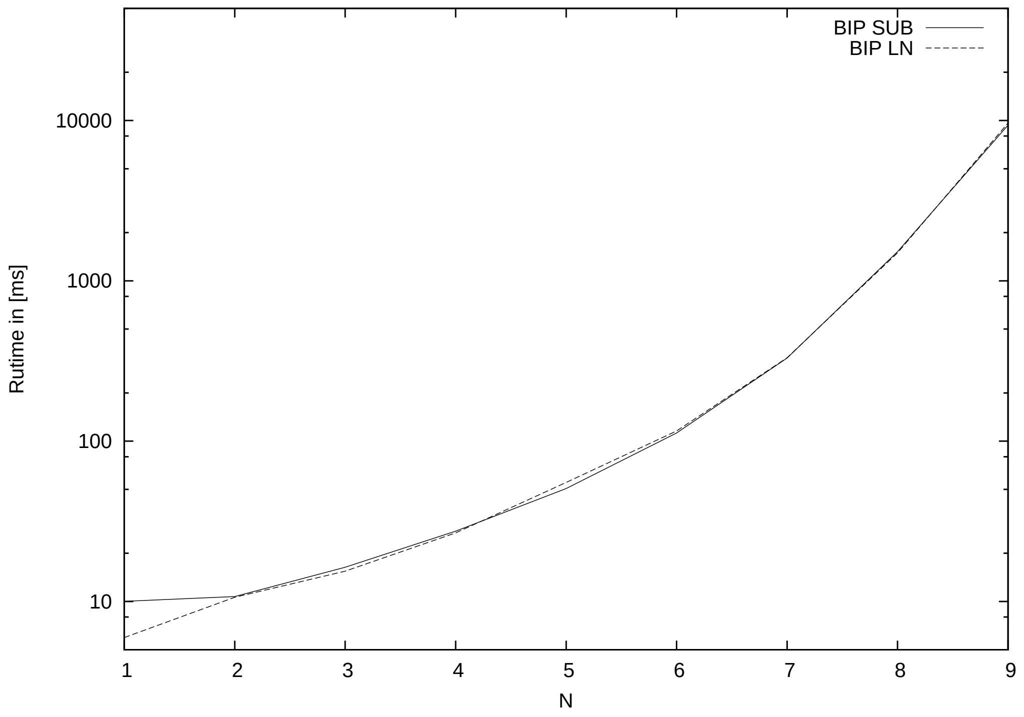
For evaluating the runtime of the different approaches, a simulation study with 100 rounds per setting is conducted. Main drivers for an increasing problem complexity are parameters τ and N. In the scenario of this thesis, a growing ontology will increase the number of nodes τ. The dependency depth N can be assumed constant, as the number of dependencies is closely related to the number of layers in the service architecture, which typically is limited to a few layers. Figure 8.5 shows the influence of τ (a) and N (b) on runtime for regular service configuration graphs, Figure 8.6 for the more simple disjunctive variant.[10] Average runtime is depicted for all graph-based and BIP approaches presented in Section 8.3. Average brute-force runtime is shown for low complexity settings in Table 8.3 for the generic variant $\mathcal{G}$ and Table 8.4 for the disjunctive variant $\mathcal{G}^+$, with $|V|$ denoting the number of vertices in the graph.

Looking at the generic service configuration graph allowing conjunctive edges (cf. Figure 8.5), one recognizes exponential growth in both parameters. Nevertheless, realistic problem sizes with up to 20 resource options per dependency node $n^\bullet$ still can be computed within one second (cf. Figure 8.5a). Concerning the disjunctive service configuration graph, polynomial runtime can be achieved by both the Dijkstra-based mechanism and the BIP approach with respect to parameter τ (cf. Figure 8.6a), opposed to the exponential growth of the brute-force algorithm (cf. Table 8.4). However, runtime shows an exponential growth rate in dependency depth N (cf. Figure 8.6b). As stated before, the dependency depth can be assumed constant even for large ontologies, as a single service configuration will not grow in the number of resources.

[10]The runtime is depicted in a logarithmic scale to achieve a better readability of the graphs.

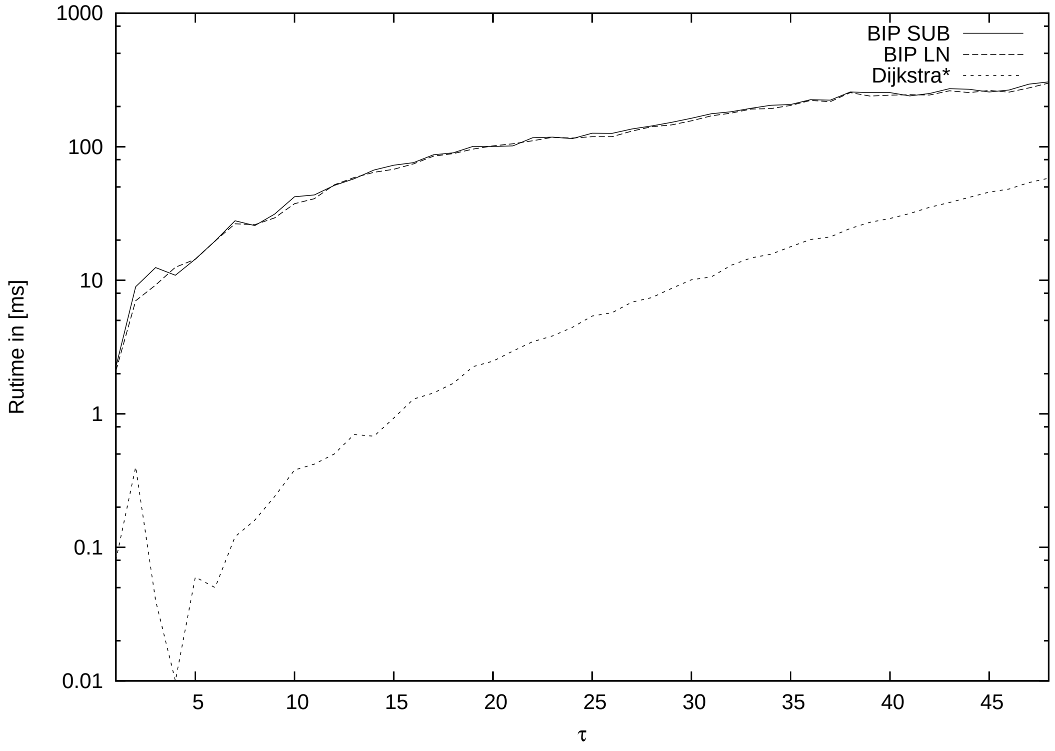


(a) Number of nodes τ

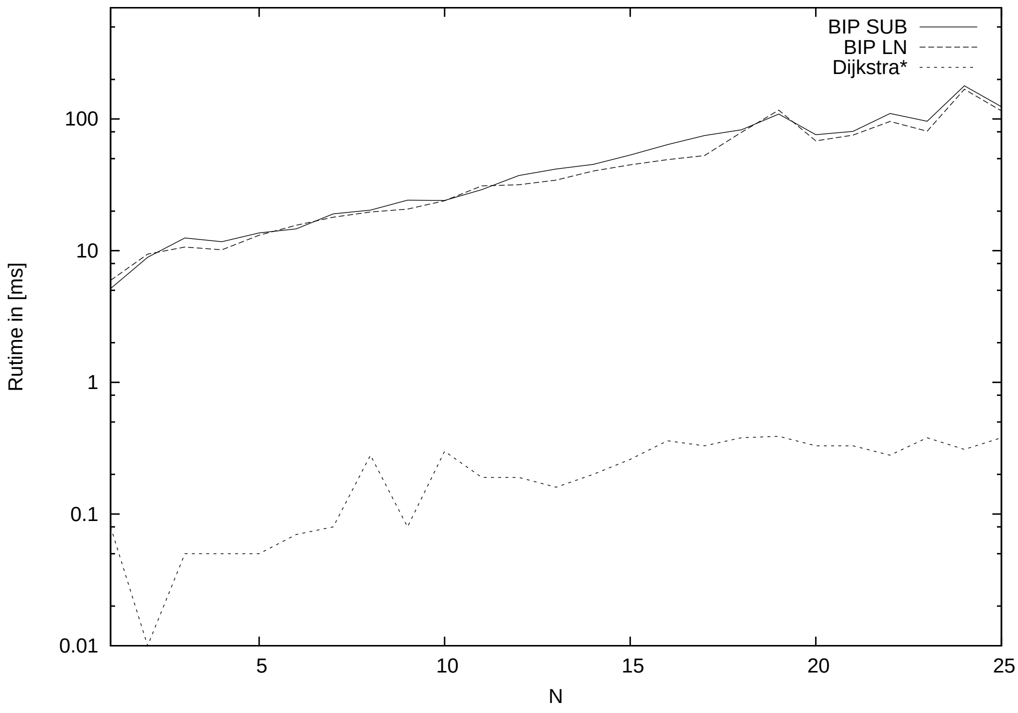


(b) Graph depth N

Figure 8.5: Average runtime [ms] – service configuration graph ($\mathcal{G}$)



(a) Number of nodes τ



(b) Graph depth N

Figure 8.6: Average runtime [ms] – disjunctive service configuration graph ($\mathcal{G}^+$)

τ	|V|	**Solutions**	**Brute-Force**	**BIP SUB**	**BIP LN**
1	18	1	0.02	2.31	2.07
2	32	4	0.09	5.09	4.53
3	46	11247	45.24	21.04	14.56
4	60	29470	133.92	17.25	18.42
5	74	5190360	22828.41	27.14	16.07

(a) Number of nodes τ

N	|V|	**Solutions**	**Brute-Force**	**BIP SUB**	**BIP LN**
3	46	–	–	16.36	15.47
6	100	–	–	112.52	115.76
9	154	–	–	9349.00	9643.20

(b) Graph depth N

Table 8.3: Average runtime [ms] – service configuration graph ($\mathcal{G}$)

In general, runtime is found to grow faster in a setting with conjunctive edges (generic graph $\mathcal{G}$) than in settings with disjunctive edges (disjunctive graph $\mathcal{G}^+$). The selection of a specific approximation function shows no significant effect on runtime, as both approximations are of additive nature.

All in all, graph-based optimization approaches achieve a better runtime than the BIP formulations. However, the constant amount of runtime for initialization (independent of the problem size) could not be separated in the the simulation study and is larger for the BIP formulation in comparison to the Dijkstra implementation, due to the overhead of using a complex software library like CPLEX. For disjunctive service configuration graphs, however, the runtime advantage of the Dijkstra* algorithm is significant, especially since a linear growth in runtime with respect to graph depth N is recognizable, whereas the BIP approach shows polynomial growth.

8.4.2.2 Solution Quality

To evaluate the deviation of heuristic approaches from the exact solution, two different measures are used:

Equality E measures how often the exact solution was matched by the heuristic proportional to the number of considered observations.

Deviation Δ measures the deviation of utility from the exact solution's utility in proportion to the exact solution's utility, as average over the considered observations. It can be interpreted as percental loss of welfare due to inexactness of the heuristic approach.

The results of a simulation (200 rounds for each setting) showing the effects of a growing network (parameter N) are depicted in Figure 8.7, for a setting with wide

τ	|V|	Solutions	Brute-Force	BIP SUB	BIP LN	Dijkstra*
5	16	45	0.01	14.39	14.37	0.06
10	32	490	1.21	42.22	37.49	0.38
15	47	1500	6.91	72.80	67.83	0.93
20	62	3380	25.54	100.34	101.48	2.48
25	77	7225	78.44	126.38	119.06	5.40
30	92	12000	178.15	163.34	156.45	10.08
35	107	18515	369.52	207.23	203.66	17.82
40	122	27040	694.65	254.30	243.00	28.99
45	137	40500	1309.76	257.02	263.65	45.77

(a) Number of nodes τ

N	|V|	Solutions	Brute-Force	BIP SUB	BIP LN	Dijkstra*
5	16.4	48	0.11	13.66	13.12	0.05
10	30.6	1536	3.47	24.05	23.98	0.30
15	45.05	49152	141.78	53.30	44.84	0.26
20	59.28	1572864	6353.43	76.05	68.23	0.33

(b) Graph depth N

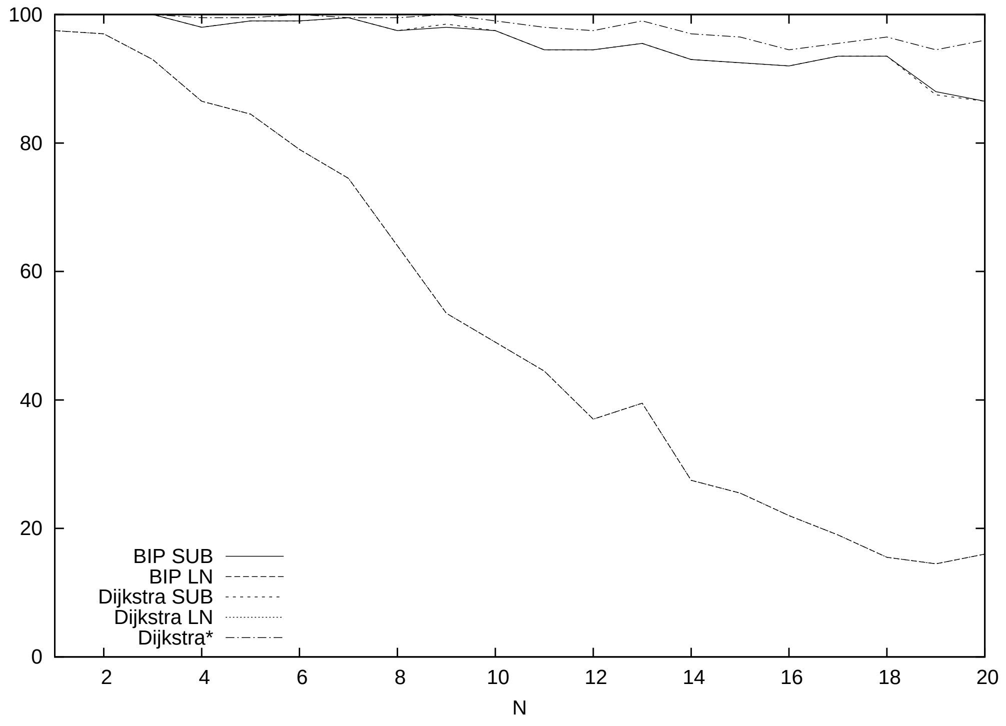
Table 8.4: Average runtime [ms] – disjunctive service configuration graph ($\mathcal{G}^+$)

boundaries ($\phi = 0.3$), and in Figure 8.8, for tight boundaries ($\phi = 1.0$). In all cases and for all heuristics, an increase in N leads to lower value of E and a higher deviation Δ.

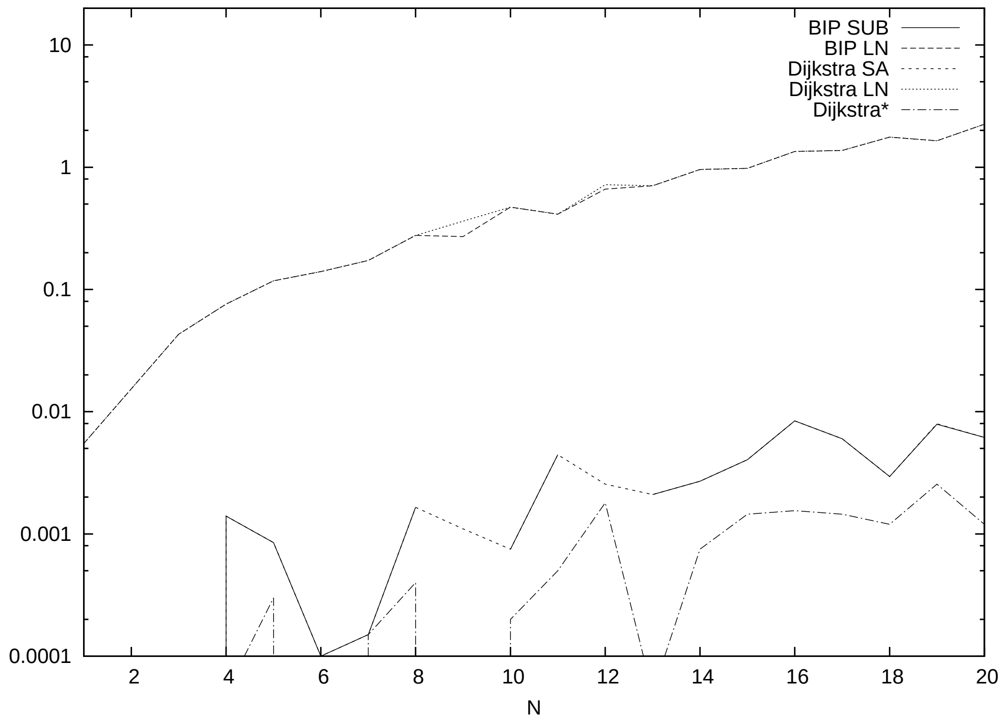
For the case of wide boundaries (cf. Figure 8.7), the Dijkstra* algorithm dominates all other approaches. As wide boundaries ensure that Bellman optimality is not violated by the normalization function – this violation only affects graph-based approaches, not BIP formulations – both Dijkstra and BIP approach relying on the same approximation function show identical error values. The subtractive approximation (SUB)[11] hereby clearly offers better results than the logarithm-based approximation (LN). Nevertheless, all heuristics apart from the logarithm-based ones guarantee a high solution quality with welfare losses below 0.1%.

In cases where aggregated QoS attribute values exceed the boundaries of the normalization function, the BIP approach's strength becomes visible (cf. Figure 8.8). Again, the subtractive approximation outruns the logarithm-based one, both with high solution quality in combination with BIP formulation ($\Delta < 0.1\%$). The Dijkstra-based approaches experience high welfare-losses ($\Delta > 25\%$) for larger values of N, thus are not suitable for a scenario with tight boundaries.

[11]With $\delta = 1$, no further tuning of the subtractive approximation was performed.



(a) Equality E



(b) Deviation Δ

Figure 8.7: Solution quality [%] depending on N – Wide boundaries $\phi = 0.3$

	Mean Δ S.A.	**Mean Δ LN A.**
BIP & Dijkstra, $\phi = 0.3$	0.00262	0.69133
BIP, $\phi = 1.0$	0.00302	0.10274

Table 8.5: Mean values of deviation Δ in %, $N \in [0,20]$

For validating the predominance of the subtractive approximation over the logarithm-based one, a comparison of mean values is shown in Table 8.5.[12] In all scenarios and in combination with each potential algorithm, the subtractive approximation shows significantly lower mean welfare losses than the logarithm-based approximation.

8.5 Conclusion

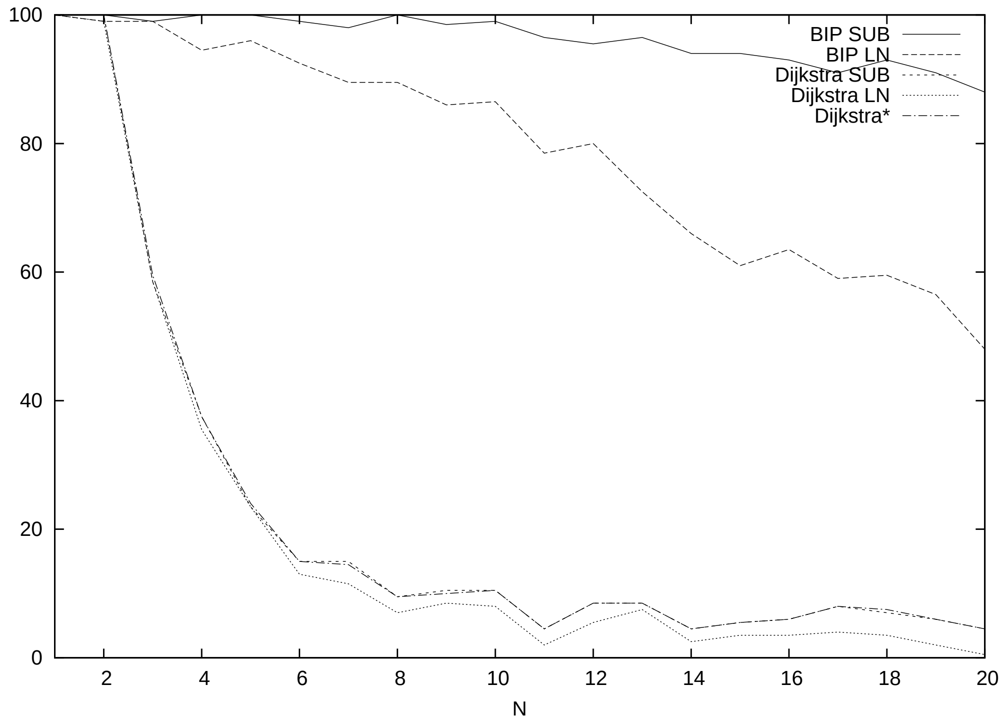
This chapter was dedicated to computational efficient optimization techniques for finding optimal service configurations in a graph structure that, for example, results from the service engineering algorithm. A more complex form of SCORINGBIDDING was assumed to be implemented by the service provider as negotiation mechanism.

In the presented model, the functional requirements from Chapter 6, that lead to a certain service configuration graph, are extended by non-functional requirements. They contain the consumer preferences regarding certain QoS attributes and a willingness to pay for a service configuration offering maximal preference fulfillment. Through introducing lower and upper bounds for each QoS attribute and a parameter for weighting the various attributes, a linear scoring function is obtained which allows the usage of either binary integer programming or graph algorithms like Dijkstra to be used for solving the optimization problem.

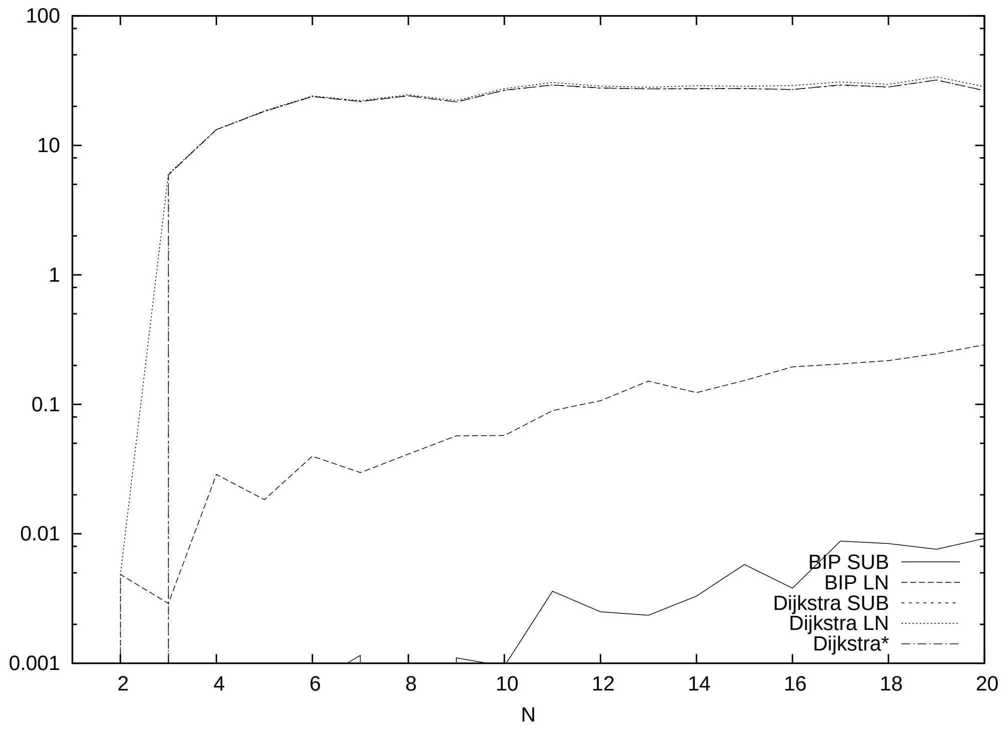
The major challenges within the presented model arise from the required aggregation of QoS attributes, their normalization and from conjunctive nodes. Regarding the first, we recognized that nonadditive aggregation operators violate the Bellman property, hence hinder the use of efficient graph algorithms without losing exactness. In a BIP approach, additivity is also required to obtain a linear objective function. Regarding the latter, *and* instead of *or* semantics require more sophisticated solutions as not paths, but subgraphs are feasible service configurations within the solution space.

When dealing with non-Bellman attributes, a special case was identified when aggregating high percentage values by multiplication, as in the case of *availability*, a QoS attribute commonly of high interest. For these types of attribute, the notion of a "quasi-Bellman" attribute was introduced. It turned out, that two rather simple approximations can handle quasi-Bellman attribute in an additive way. One of the two – the subtractive approximation – was found to be superior to the other one – the logarithmic approximation.

[12]The comparison is based on all simulation runs over all variations of N. Statistical tests are omitted given the effect size and number of total observations.



(a) Equality E



(b) Deviation Δ

Figure 8.8: Solution quality [%] depending on N – Tight boundaries $\phi = 1.0$

In the evaluation section of this chapter, the expected error for this approximation was derived analytically. For reasonably sized graphs – which already become computationally complex through their combinatorial nature – the subtractive approximation was found to have acceptable small errors, as long as dealing with availability values in the range above 90%. With values of 99% or above, the error was even found to be well below 1%.

Additionally, a broad simulation study was conducted to investigate on both runtime and solution quality of all techniques in randomized service configuration graphs. In graphs with disjunctive edges only, Dijkstra or Dijkstra* are dominating both regarding runtime and solution quality, unless boundaries of the normalization function are violated. In this case, the slower binary integer programming approach using the subtractive approximation offers the least error. The same technique also is capable of dealing with conjunctive edges, where the graph based Dijkstra algorithm is no longer suitable. However, this problem instance grows exponentially in number of nodes per dependency and in dependency depth, making it only a viable concept for smaller graphs. In case absolute correctness is required or non-Bellman attributes are used, brute-force algorithms can provide a solution. However, they yield exponential time complexity with exploding runtime even for rather small graph instances.

All of the results also hold for a provider implementing DISCOUNTBIDDING, as the discount parameter d merely implies a linear transformation and hence neither influences solution quality, nor runtime of the presented optimization techniques.

The presented results have three major limitations. Firstly, one cannot claim that the presented techniques are the only suitable mechanisms for solving the problem at hand. Secondly, it is not proven that the presented approximation and optimization techniques are optimal regarding runtime or solution quality. Thirdly, we do not provide a solution for non-Bellman attributes apart from the brute-force algorithms. Especially regarding Boolean attributes, optimization algorithms smarter than brute-force are thinkable, as the combinatorial explosion is limited due to true/false values only. Nevertheless, through the findings of this chapter, an efficient optimization of service configuration problems can be achieved in many standard settings.

8.5.1 Practical Implications

Based on the above made observations, a recommendation on when to use which approach can be given. Dominated heuristics and approximation functions – Dijkstra in conjunction with either approximation and the logarithm-based approximation in general – are not considered. Table 8.6 summarizes the evaluation results and indicates the eligibility of the different techniques under different circumstances. Brute-force algorithms can cope with all challenges, but are only suitable in offline settings. Dijkstra and Dijkstra* are the best choice in disjunctive graphs and when normalization boundaries are unlikely to be violated. Regarding runtime, they outperform all other presented approaches. Dealing with tight boundaries or conjunctive edges, the BIP formulation can be a remedy, yet, at the cost of worse expected runtime behavior. In order to cope with quasi-Bellman attributes, the subtractive

	Exact Mechanisms			Heuristics	
	Brute-Force	**Dijkstra**	**BIP**	**Dijkstra***	**BIP SUB**
8.1 Conjunctive edges	●	○	●	○	●
8.2 Only Bellman attributes	●	●	●	●	●
8.3 Quasi-Bellman attributes	●	○	○	●	●
8.4 Non-Bellman attributes	●	○	○	○	○
8.5 Boundary violation	●	○	●	○	●
8.6 Offline	◐	●	●	●	●
8.7 Online	○	●	●[13]	●	●[13]
8.8 Runtime Critical	○	●	◐[13]	●	◐[13]
Extensibility	○	○	●	○	●

Table 8.6: Overview on findings

approximation is required in the BIP formulation, offering an acceptable error rate for reasonable graph depths.

Another aspect important from a practical perspective included in this overview is extensibility. Due to the generic formulation of binary integer programs consisting of any type of linear objective function along with linear constraints, further restrictions can be included. In addition, it is possible to adapt the BIP formulation in order to optimize over several service requests simultaneously, which can be beneficial when dealing with service requests competing for scarce resources.

[13]Only for small service configuration graph sizes when dealing with conjunctive edges.

Part IV

Finale

Chapter 9

Conclusion and Outlook

In the last decade, mass customization has found its way into many areas of traditional manufacturing. While this trend offers great business opportunity by addressing the "long tail" of consumers, it has not reached current Cloud service offers. Many obstacles, from technological to economic challenges, have to be overcome in order to enable mass customized service offers in the field of Cloud computing. This thesis is dedicated to these obstacles, coping with both technical and economical problems in a holistic manner. With knowledge-based engineering, negotiation and optimization three distinct areas have been addressed. The chapter at hand summarizes the obtained results, discusses limitations and provides an overview on future work as well as complementary research.

This chapter is structured as follows: A summary of all contributions is given in Section 9.1, followed by limitations and future work in Section 9.2. Lastly, complementary research topics are presented in Section 9.3.

9.1 Contribution

The contributions of the work at hand are spread among three parts: Part I introduced the reader to the thesis, its topic – mass customization in the context of Cloud service offers – and gave a broad overview on basic concepts and technologies which build the thesis's foundation. Part II covered the technical aspects of engineering mass customized Cloud services, while Part III had its focus on the economic perspective – multi-attributive negotiation mechanisms and service optimization.

Concerning the functional engineering of a Cloud service, the need to externalize knowledge on available service resources and the functionalities which can be provided by them was identified. The necessary knowledge should be stored in a machine readable form and in a such a manner, that it serves as a catalog of functionalities to the consumer and that it can be used to derive all feasible resource configurations which match the consumer's selected functionalities. This requires a framework that supports both the modeling of dependencies and interoperability between resources. Research Question 1 addresses this challenge.

Research Question 1 ≺KNOWLEDGE MANAGEMENT FOR SERVICE RESOURCES≻. *Is it possible to design a semantic service description framework where one can model knowledge on service resources so that technically feasible service configurations matching the functional requirements of a consumer request can be derived automatically?*

Addressing this question, fundamental knowledge management concepts were introduced in Section 2.1, including ontologies and formalisms like OWL, SWRL and others. Characteristics and definitions of tangibles, intangibles and more specific information on service concepts like Cloud services, SOAs, mass customized services and SVNs were given in Section 2.2. A broad understanding of these concepts helps to comprehend the particularities when modeling knowledge on service resources.

The main contribution addressing this question is the semantic service description framework provided in Chapter 4. Apart from deriving distinct requirements and covering the related literature, Section 4.3 presented a three-fold ontology framework, comprising a generic service ontology, a domain ontology and a result ontology. The generic concepts defined in the service ontology are used within the domain ontology to store information on service resources and functionalities, along with their dependencies and compatibility information. The framework is built upon OWL in conjunction with SWRL rules as formal languages, fostering standard compliance, extensibility and exchangeability across the borders of institutions. A major concern when designing this framework was a reduced modeling overhead, i.e. knowledge should be modeled on higher abstraction levels, if possible. The goal was achieved by using description logic expressiveness in conjunction with the horn-like rules of SWRL, allowing to model information on both dependencies and compatibility on class rather than instance level.

In order to use such a framework in the engineering and optimization phase of the Cloud service offer creation process, the knowledge base has to contain up-to-date knowledge. In a scenario with constantly changing infrastructure conditions, price and quality information associated with a service resource requires continuous updates. This issue is addressed in Research Question 2.

Research Question 2 ≺ONTOLOGY UPDATE MECHANISM≻. *Is it possible to design an automated ontology update mechanism which integrates information on QoS or price from manifold data sources in an automated and structured manner, so that the ontology's data values are kept up-to-date and can be accessed in a shared environment, and can such a mechanism be scalable?*

Again, the necessary foundations were provided in Chapter 2, with ontology persistence in Section 2.1.3 being particularly relevant to this research question. The ontology update mechanism itself was described in Chapter 5. After refining the general use case to cover all relevant details for this research topic, requirements were derived and related work was presented. As main contribution, both a conceptual software architecture and its prototypical implementation followed in Sections 5.4 and 5.5 respectively. The presented concept is capable of updating data values in any OWL ontology from manifold sources from the Web, supporting different data formats. Examples were given for HTML, XML and RESTful Web services as data source. Through the modular architecture, the software can easily be extended by further types. The prototypical implementation was then used to evaluate the software performance. As major result, the approach was found to scale well with the number of update rules defined in the ontology.

Once having designed a semantic service description framework and a mechanism to keep the knowledge within up-to-date, an algorithm was sought which is capable of deriving feasible service configurations for a particular consumer request. Such an algorithm needs to resolve dependencies beginning from the functional requirements defined by the consumer in her request, while simultaneously ensuring compatibility among the selected service resources. The challenge of designing such an algorithm is formulated in Research Question 3.

Research Question 3 ≺SERVICE ENGINEERING ALGORITHM≻. *Is it possible to design an algorithm that is capable of deriving all feasible service configuration alternatives based on a set of abstract functional requirements originating from a semantic service description framework, and can that algorithm be scalable?*

The relevant foundations to this question overlap the foundations required by the previous contributions to a large extend. In addition to OWL and SWRL, knowledge on OWL reasoning and SPARQL as query language are requirements for this contribution. Both, reasoning and SPARQL were briefly introduced in Section 2.1.2. In order to derive service configurations from the presented ontology framework, a formal model to represent both input and output of the designated algorithm is required. After presenting requirements and related work, a formal model for functional requirements as input and a service configuration graph as output for the service engineering algorithm was given in Section 6.3. Based on the functional requirements and a domain ontology built upon the concepts of the generic service ontology, the algorithm derives a graph over feasible service configuration by recursively resolving dependencies for each functionality and service resource, until no more unresolved dependencies are left. The concept of the algorithm was described in Section 6.4, a prototypical implementation followed in Section 6.5. It was evaluated against the posed requirements and found to be scalable even for complex configuration scenarios with many different resource alternatives and longer dependency hierarchies.

After addressing these technical research questions, the focus of Part III is on economic problems. Provider and consumer of a mass customized Cloud service not only need to agree on the functionalities provided, but also on non-functional properties like quality and price of a particular service. Naturally, quality and price go hand-in-hand with the available service configurations. In order for both parties to find an agreement – especially in an online scenario – different negotiation mechanisms can help them to find an adequate solution. Different economic desiderata like individual rationality, budget balance, incentive compatibility and allocation efficiency play an important role in judging the eligibility of a certain mechanism. Yet, theory has proven that not all of these goals can be achieved at the same time. In Chapter 7, we sought a negotiation mechanism that fulfills individual rationality and budget balance and, at the same time, keeps efficiency as well as truthful information content high, while maintaining a simple bidding language. This challenge is addressed in Research Question 4.

Research Question 4 $\prec$MULTI-ATTRIBUTIVE NEGOTIATION MECHANISMS$\succ$. *What are possible budget balanced and individual rational mechanisms for bilaterally negotiating quality and price of a service and how do they perform with respect to the economic desiderata incentive compatibility and ex-post efficiency?*

Important economic foundations mainly in the field of mechanism design, like the above mentioned desiderata, were presented in Section 2.3. To answer the research question, a simplified model of preferences allowing analytical conclusions was introduced in Section 7.4. We scrutinized three different negotiation mechanisms, presented in Section 7.5, all of them being individual rational and budget balanced, yet suffering from a lack of incentive compatibility and allocation efficiency: TUPLEBIDDING serves as proxy for the take-it-or-leave-it mechanisms commonly found in practice. SCORINGBIDDING relates to the concept of bidding a scoring function as it is often proposed in the multi-attributive auction theory. A third mechanism, DISCOUNTBIDDING, was newly introduced in this thesis as extension to SCORINGBIDDING, offering the advantage of having separate bidding parameters for disconnecting the integrative facet from the distributive facet of the negotiation.

All three mechanisms were evaluated regarding the truthfulness, the resulting utilities for both parties and the obtained efficiency. This was done analytically in the case of complete information in Section 7.6 and with the consumer facing risk on the providers cost structure in Section 7.7 by relying on a extensive simulation study. Summarizing, one can conclude that TUPLEBIDDING is not Pareto-optimal, but dominated by the other two mechanisms. If the provider can choose between the three mechanisms, he will favor SCORINGBIDDING over the other mechanisms independent of both level of risk and risk aversion. For the consumer, DISCOUNTBIDDING or TUPLEBIDDING are the mechanisms of choice yielding the highest expected utility. From a welfare perspective, DISCOUNTBIDDING was found to perform best as soon as the consumer faces some risk. In the model and in analogy to the overall scenario of the work at hand, the consumer was assumed to go first in all mechanisms and thus confronted with risk in his decision making. Through the symmetry in the model, the results, however, also hold in the opposite direction.

Lastly, making use of the technical contributions and the considerations on an adequate negotiation mechanism, the question of choosing the optimal, i.e. profit maximizing, configuration remains open. Assuming a provider that is implementing a form of SCORINGBIDDING adapted to multi-dimensional QoS, the optimal configuration choice requires a suitable optimization technique to find the best alternative with respect to a given service configuration graph and the submitted non-functional requirements. The first spanning the solution space and the latter representing the obtainable revenue for a given service quality. This final challenge is addressed in Research Question 5.

Research Question 5 $\prec$OPTIMIZATION TECHNIQUES FOR SERVICE CONFIGURATION$\succ$. *What are possible optimization techniques for finding the profit maximizing service configuration with respect to QoS and price from a set of alternatives* *(given as a graph structure) and how do they perform with respect to runtime and allocative optimality?*

A brief overview on the foundations of mathematical optimization was given in Section 2.4. The main contribution followed in Chapter 8, which is dedicated to different optimization techniques in the context of service configuration graphs and QoS-aware consumer preferences. Again, a formal model adapted to the peculiarities of the optimization scenario was presented in Section 8.2. Within this model, the consumer preferences are stated as a combination of a linear scoring function and a maximum willingness to pay which is multiplied by the obtained score. QoS values are aggregated from the individual service resources through various aggregation operations, impacting the computational tractability of the resulting optimization problem. We investigated on different techniques for two different kinds of service configuration graphs: a generic one allowing conjunctive nodes, as it results from the service engineering algorithm, and a simplified graph structure, when service resources are constrained to at most one dependency each.

Three different solution techniques have been developed: brute-force algorithms that iterate through the entire search space, a binary integer programming approach where service resources are binary decision variables and a graph based approach using Dijkstra's algorithm. To allow an efficient computation also in the context of the very often used attribute *availability*, two different approximations were proposed, allowing to aggregate multiplicative QoS values in the range above 90% in a linearized manner. A detailed evaluation of the various techniques in combination with and without the approximations under different circumstances was conducted in a broad simulation study. All of the optimization approaches thereby showed advantages and disadvantages, making each of them more or less adequate in different contexts.

9.2 Limitations and Future Work

In the following, the above mentioned results are critically analyzed in order to expound their limitations. In addition, future research directly linked to the recognized open issues is presented.

Regarding the semantic service description framework presented in Chapter 4, several reasonable extensions and shortcomings to the approach are recognized. Firstly, a more complex cost and quality model would be beneficial. Secondly, compatibility can be ensured through complex rules, yet is limited to dependent resources only. The approach currently offers no possibility to ensure compatibility between resources in the same configuration without having a direct link. However, a concept for modeling further constraints could easily address this issue. Thirdly, we recognize the need for a better evaluation, qualitatively, through relying on an industry use case, and quantitatively, by analyzing both the economic benefit of externalized knowledge in conjunction with the other contributions, versus the costs that accrue from the creation and maintenance of the ontology. Lastly, a stronger integration with the standardization efforts of the Semantic Web community and the WS-*

stack would improve its standard compliance. Grounding the presented framework to a foundational ontology like DOLCE [104] could prevent conceptual ambiguity or poor axiomatization when integrating the presented concepts into other ontological frameworks. All of these shortcomings are subject to future work.

The second contribution of the work at hand addresses the issue of information up-to-dateness. While the ontology update mechanism presented in Chapter 5 offers a scalable, transparent and consistent automatism to update information stored in an ontology from manifold data sources, the approach is limited to updating data values. Additional service resources, compatibility and dependency information or any other structural changes to the ontology have to be made manually. Further, the obtained transparency of the mechanisms could be further improved by introducing an update history and providing a user interface for monitoring update activities. Lastly, proof-of-concept implementation and evaluation of the update functionality are based on rather simple example update instructions. An application of the proposed solution in a real business context might require more sophisticated update instructions, especially regarding data extraction and transformation, potentially also influencing the systems scalability. Again, all three issues will be addressed in future research.

In Chapter 6, an algorithm capable of deriving feasible service configurations by resolving dependencies defined on an abstract level while ensuring compatibility between dependent resources was presented. In this chapter, two main shortcomings can be identified. Firstly, in close relation to the criticism on the semantic framework, the presented approach ensures compatibility between dependent resources only. Secondly, the algorithm was analytically evaluated regarding its runtime. However, queries to the ontology were assumed to be of constant time. A simulation study with different ontology sizes and various complexity levels regarding the used expressiveness needs to conducted in order to find out the influence of reasoning query time. It is subject to future work.

The results regarding the multi-attributive negotiation mechanisms presented in Chapter 7 have four main limitations: Firstly, individually rational utility maximizing agents are assumed, which is rare in any real-world setting. Secondly, the results depend on the model of preferences, especially on the functional forms of cost, scoring, and utility functions, the assumed probability distribution function and are limited to a single QoS attribute. While it is believed that similar results can be obtained for other preferences and more than one QoS attribute, the proof is subject to future work. Thirdly, "only" three distinct mechanisms were studied without deriving an "optimal mechanism" in the mechanism design sense. Nevertheless, the obtained results are a valuable contribution for the practical scenario in the context of mass customization of the work at hand. Fourthly, all three mechanisms only allow for a single offer and its acceptance or rejection by the counterparty. More complex negotiation mechanisms with an alternating offer exchange are possible, so is the introduction of a central marketplace.

The last contribution of this thesis was dedicated to the challenge of finding optimal configurations from the perspective of a profit-maximizing Cloud provider. Regarding the results of Chapter 8, three shortcomings can be identified: Firstly, only a distinct set of the most important aggregation operators is considered. While the introduction of two approximation functions is a valuable contribution to an efficient optimization with quasi-Bellman attributes like availability, no solution is provided that can deal with multiplicative QoS attributes of generic nature. Secondly, the binary integer programming approach offers acceptable runtime for smaller problem instances of service configuration graphs containing conjunctive nodes, yet it does not scale for large problem instances due to its exponential growth in runtime. While it is believed that the problem itself lies within an exponential runtime complexity class, this proof is subject to future work. Thirdly, the aggregation of QoS attributes is restricted to a simple hierarchy, aggregating with the same aggregation function over the complete stack of resources. This approach does not cover the potential process nature of service configurations, which would imply more complex aggregation functions. Research in this area can be found for example in [87].

9.3 Complementary Research

This section provides an overview on research directions which are related, but do not contribute a direct extension to this thesis. The goal is to briefly introduce the reader to complementary topics that further promote mass customization in the context of Cloud computing.

From a technical point of view, the research areas ontology versioning, maintenance and collaborative evolution have to be mentioned. Ontology versioning [86, 124] has its focus on the creation and management of different versions of a given ontology, which could also be beneficial in the context of the mass customization scenario presented in the work at hand. By adding additional service resources and functionalities, a centralized versioning system becomes essential, especially in shared environments with more than one person performing changes. An often mentioned issue when dealing with ontologies is the complexity of their creation, maintenance and evolution. Without the use of tools engineered to the needs of the specific use case, only highly skilled semantic technology experts are able to perform these tasks. Yet, given appropriate tools and monetary incentives, a collaborative ontology evolution approach would be possible, allowing 3rd party service resource providers to maintain the information on their service offers themselves. In such a scenario, the 3rd party resource providers could use the integrating Cloud provider as additional distribution channel. Thus, supplying companies would have an incentive to collaborate in the ontology's evolution.

The contributions of this thesis result in an optimized service offer, comprising the service resource configuration, quality and price of the offer. An autonomic deployment of the service, however, is not in the scope of the thesis. Current research in computer science, mainly in the field of autonomic computing [84], addresses this issue, going closely at hand with the other self-* properties proclaimed in the autonomic computing community.

Turning to the economic perspective, it is often assumed that agents are aware of their true preferences and valuations. The same assumption is made in the economic part of this thesis. However, even for professional consumers, stating one's preferences is not an easy task. Especially in multi-attributive scenarios, elicitation of preferences can become complex. Regarding the model presented in Section 8.2, consumers must be aware of both their willingness to pay for a perfect service and their upper and lower boundaries for each and every QoS attribute of interest. Prominent solutions to the preference elicitation problem are conjoint analyses [56] and analytical hierarchy processing [136]. Yet, for the scenario of the work at hand, these approaches are not appropriate due to their complexity, especially in settings with many QoS attributes.

Complementary to the thesis's scenario would be the introduction of a central marketplace where several providers can interact with potential consumers, negotiating over quality and price of a service in an auction setting. The concept of a complex service auction [28] takes this approach even further, having numerous individual providers offer a collaborative service in a coopetition scenario.

Lastly, Myerson and Satterthwaite [121] provided two seminal theorems in the context of bilateral trading: Theorem 1 proving the impossibility of achieving all four classical mechanism design desiderata at the same time, Theorem 2 providing the means to compute mechanisms that maximize expected total gains from trade for a wide class of problems. Extending these theorems to a multi-attributive setting with valuations depending on further attributes without doubt would be a seminal breakthrough in this research area.

Part V

Appendix

Appendix A

Appendix to Part II

A.1 Service Ontology

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<rdf:RDF xmlns="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#"
     xml:base="http://research.steffenhaak.de/ontologies/ServiceOntology.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl"
        />



    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Datatypes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->




    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->




    <!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#choice -->
```

```
<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#choice">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#OrNode"/>
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#ServiceEntity"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#connectsTo -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#connectsTo">
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                    ServiceOntology.owl#ServiceEntity"/>
                <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                    ServiceOntology.owl#SourceNode"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
    <rdfs:range>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                    ServiceOntology.owl#ResultNode"/>
                <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                    ServiceOntology.owl#ServiceEntity"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:range>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#contains -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#contains">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#Alternative"/>
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#ServiceEntity"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    hasAggregationFunction -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasAggregationFunction">
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#AggregationFunction"/>
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#QualityMetricType"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#hasQualityMetric -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasQualityMetric">
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#QualityMetric"/>
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
</owl:ObjectProperty>
```

```
<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#hasQualityMetricType
    -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasQualityMetricType">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#QualityMetric"/>
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#QualityMetricType"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#inverseOfChoice -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#inverseOfChoice">
    <owl:inverseOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#choice"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#inverseOfConnectsTo
    -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#inverseOfConnectsTo">
    <owl:inverseOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#connectsTo"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#isCompatibleTo -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#isCompatibleTo">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#ServiceEntity"/>
    <owl:propertyDisjointWith rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#isRequiredBy"/>
    <owl:propertyDisjointWith rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#requires"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#isRequiredBy -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#isRequiredBy">
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#ServiceEntity"/>
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
</owl:ObjectProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#requires -->

<owl:ObjectProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#requires">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
    <rdfs:range rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology
        .owl#ServiceEntity"/>
    <owl:inverseOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#isRequiredBy"/>
</owl:ObjectProperty>



<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////////////////////////////////////
 -->




<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#hasDefaultValue -->

<owl:DatatypeProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasDefaultValue">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#QualityMetricType"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#hasFixCosts -->

<owl:DatatypeProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasFixCosts">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#Resource"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    hasQualityMetricValue -->

<owl:DatatypeProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasQualityMetricValue">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#QualityMetric"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#hasVariableCosts -->

<owl:DatatypeProperty rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#hasVariableCosts">
    <rdfs:domain rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#Resource"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>



<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////
 -->
```

```
<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#AggregationFunction
    -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    AggregationFunction">
    <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Alternative -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    Alternative">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ResultNode"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Functionality -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    Functionality">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#OrNode -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    OrNode">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ResultNode"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#QualityMetric -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    QualityMetric">
    <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#QualityMetricType --
    >

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    QualityMetricType">
    <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Resource -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    Resource">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ServiceEntity"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#ResultNode -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
    ResultNode">
    <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#ServiceEntity -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
     ServiceEntity">
    <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#SinkNode -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
     SinkNode">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ResultNode"/>
</owl:Class>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#SourceNode -->

<owl:Class rdf:about="http://research.steffenhaak.de/ontologies/ServiceOntology.owl#
     SourceNode">
    <rdfs:subClassOf rdf:resource="http://research.steffenhaak.de/ontologies/
        ServiceOntology.owl#ResultNode"/>
</owl:Class>



<!-- http://www.w3.org/2002/07/owl#Class -->

<owl:Class rdf:about="&owl;Class"/>



<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Individuals
//
///////////////////////////////////////////////////////////////////////////////////////
 -->




<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Addition -->

<owl:NamedIndividual rdf:about="http://research.steffenhaak.de/ontologies/
     ServiceOntology.owl#Addition">
    <rdf:type rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology.
        owl#AggregationFunction"/>
</owl:NamedIndividual>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Maximum -->

<owl:NamedIndividual rdf:about="http://research.steffenhaak.de/ontologies/
     ServiceOntology.owl#Maximum">
    <rdf:type rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology.
        owl#AggregationFunction"/>
</owl:NamedIndividual>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Minimum -->
```

```
<owl:NamedIndividual rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#Minimum">
    <rdf:type rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology.
        owl#AggregationFunction"/>
</owl:NamedIndividual>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Multiplication -->

<owl:NamedIndividual rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#Multiplication">
    <rdf:type rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology.
        owl#AggregationFunction"/>
</owl:NamedIndividual>



<!-- http://research.steffenhaak.de/ontologies/ServiceOntology.owl#Subtraction -->

<owl:NamedIndividual rdf:about="http://research.steffenhaak.de/ontologies/
    ServiceOntology.owl#Subtraction">
    <rdf:type rdf:resource="http://research.steffenhaak.de/ontologies/ServiceOntology.
        owl#AggregationFunction"/>
</owl:NamedIndividual>




<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// General axioms
//
///////////////////////////////////////////////////////////////////////////////////////
 -->

<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDifferent"/>
    <owl:distinctMembers rdf:parseType="Collection">
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Addition"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Maximum"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Minimum"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Multiplication"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Subtraction"/>
    </owl:distinctMembers>
</rdf:Description>
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#AggregationFunction"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#QualityMetric"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#QualityMetricType"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#ResultNode"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#ServiceEntity"/>
    </owl:members>
</rdf:Description>
<rdf:Description>
    <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
    <owl:members rdf:parseType="Collection">
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#Alternative"/>
        <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
            ServiceOntology.owl#OrNode"/>
            <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                ServiceOntology.owl#SinkNode"/>
            <rdf:Description rdf:about="http://research.steffenhaak.de/ontologies/
                ServiceOntology.owl#SourceNode"/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>



<!-- Generated by the OWL API (version 3.3.1957) http://owlapi.sourceforge.net -->
```

Appendix B

Appendix to Part III

B.1 Proof of Strict Monotonic Increase

Proof of strictly monotonic increase: By introducing the substitutions $a = x \cdot b$ and $\hat{a} = y \cdot b$ with $0 < x < y < 1$, one obtains the following term

$$y^* = \underset{y}{\operatorname{argmax}} \left(\frac{b}{yb}\right)^{\frac{xb}{yb-b}} - \left(\frac{b}{yb}\right)^{\frac{yb}{yb-b}} = \underset{y}{\operatorname{argmax}} \left(\frac{1}{y}\right)^{\frac{x}{y-1}} - \left(\frac{1}{y}\right)^{\frac{y}{y-1}} \tag{B.1}$$

Without loss of generality we can replace $(1/y)$ by some factor $c > 1$ as $0 < y < 1$ and obtain

$$y^* = \operatorname{argmax}_y c^{\frac{x}{y-1}} - c^{\frac{y}{y-1}} \quad \text{with} \quad c^{\frac{x}{y-1}} - c^{\frac{y}{y-1}} > 0 \quad \text{as} \quad c^{\frac{x}{y-1}} > c^{\frac{y}{y-1}} \tag{B.2}$$

$$\text{and} \quad c > 0 \quad \text{and} \quad \tfrac{x}{y-1} > \tfrac{y}{y-1} \quad \text{for} \quad 0 < x < y < 1 \tag{B.3}$$

It remains to be shown that $c^{\frac{x}{y-1}} - c^{\frac{y}{y-1}}$ grows strictly monotonically in y. This is the case, if

$$\frac{d\left(c^{\frac{x}{y-1}} - c^{\frac{y}{y-1}}\right)}{dy} > 0 \quad \text{or} \quad \frac{d\left(c^{\frac{x}{y-1}}\right)}{dy} > \frac{d\left(c^{\frac{y}{y-1}}\right)}{dy} \tag{B.4}$$

or, as $c > 1$, more simply, if

$$\frac{d\left(\frac{x}{y-1}\right)}{dy} > \frac{d\left(\frac{y}{y-1}\right)}{dy} \quad \Leftrightarrow -\frac{x}{(-1+y)^2} > -\frac{1}{(-1+y)^2} \tag{B.5}$$

$$\Leftrightarrow \quad \frac{x}{(-1+y)^2} < \frac{1}{(-1+y)^2} \quad \Leftrightarrow x < 1 \tag{B.6}$$

which is true $\forall x \in (0,y)$. (q.e.d.)

B.2 Numerical Proof of Strict Inequality

Proof of strict inequality: By introducing the substitution $a = c \cdot b$ with $c \in (0,1)$, capturing the relationship between $a \in (0,b)$ and $b \in [1,\infty)$, one obtains the following term:

$$e^{-\frac{c\cdot b}{b}} - \frac{1}{e} < \left(\frac{b}{c\cdot b}\right)^{\frac{c\cdot b}{c\cdot b-b}} - \left(\frac{b}{c\cdot b}\right)^{\frac{b}{c\cdot b-b}} \tag{B.7}$$

Simplifying this equation one can eliminate parameter b and the following equation is obtained:

$$e^{-c} - \frac{1}{e} < \left(\frac{1}{c}\right)^{\frac{c}{c-1}} - \left(\frac{1}{c}\right)^{\frac{1}{c-1}} \tag{B.8}$$

Figure B.1 numerically shows that this condition holds for $\forall c \in (0,1)$, as both terms are continuous and differentiable functions in c.

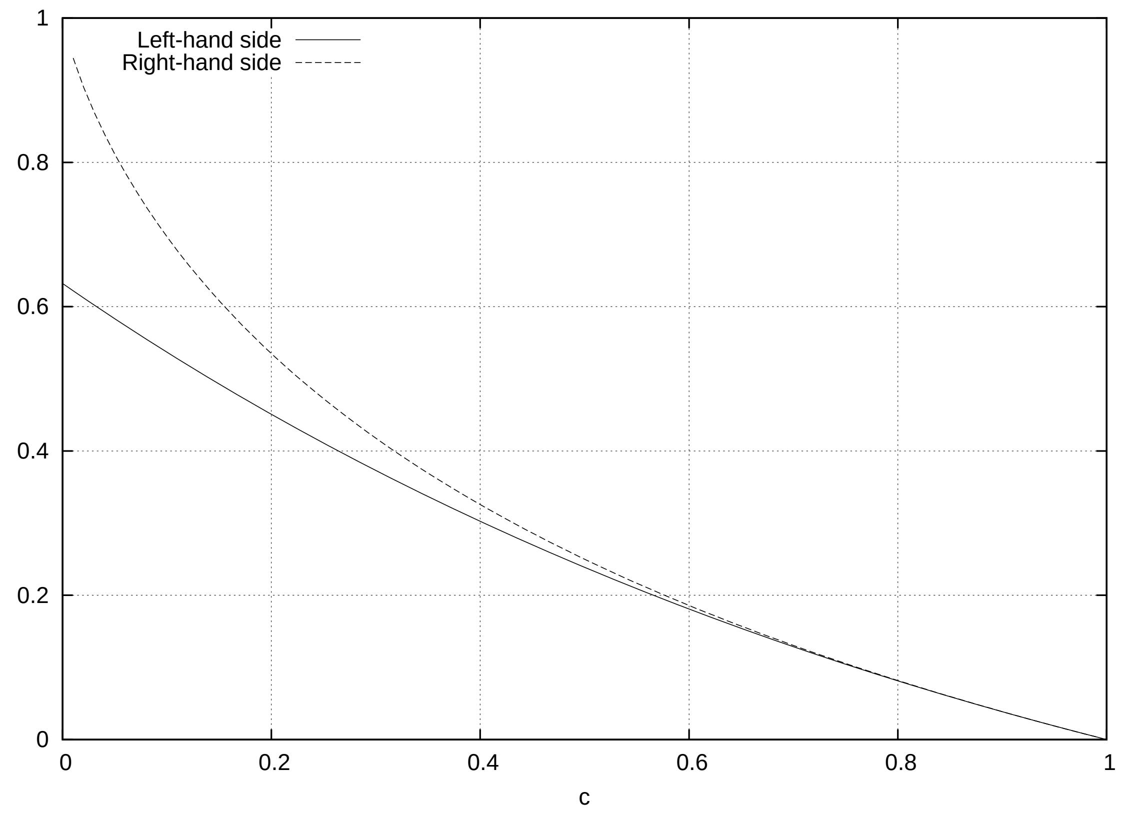


Figure B.1: Numerical proof of strict inequality

B.3 Simulation Results for Various Parameters a and b

In the following, further simulation results underpinning the robustness of Section 7.7 are presented. In all results, consumer $\mathcal{C}$ is risk neutral ($r = 0$).

B.3.1 Results for a=0.25 and b=2

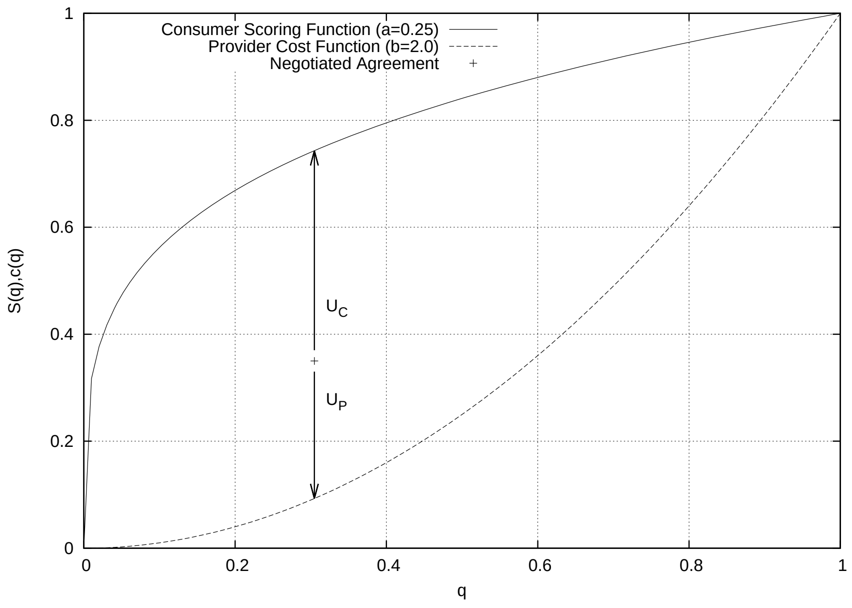


Figure B.2: Scoring and cost functions, optimal quality, price and utilities for $a = 0.25$, $b = 2$

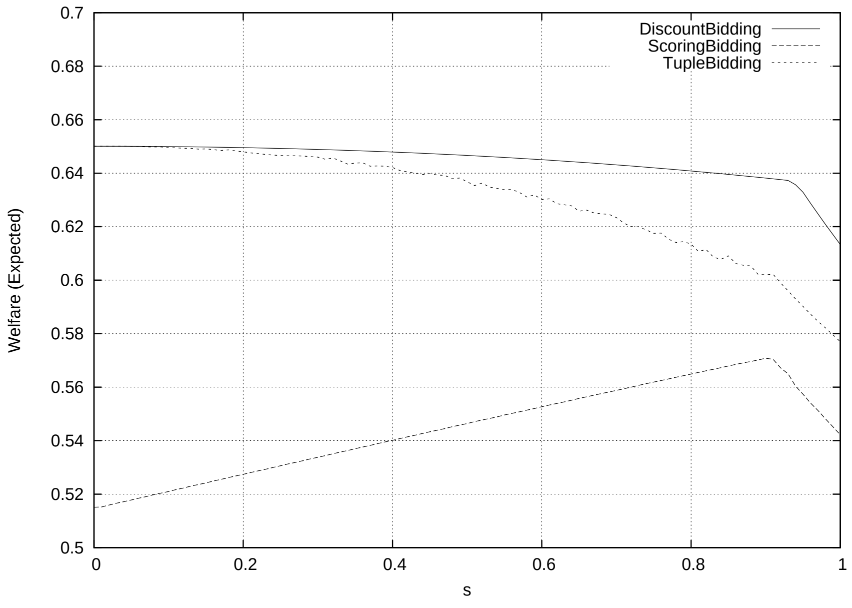


Figure B.3: Comparison of negotiation mechanisms with respect to welfare depending on the level of the consumer's risk s for $a = 0.25$, $b = 2$

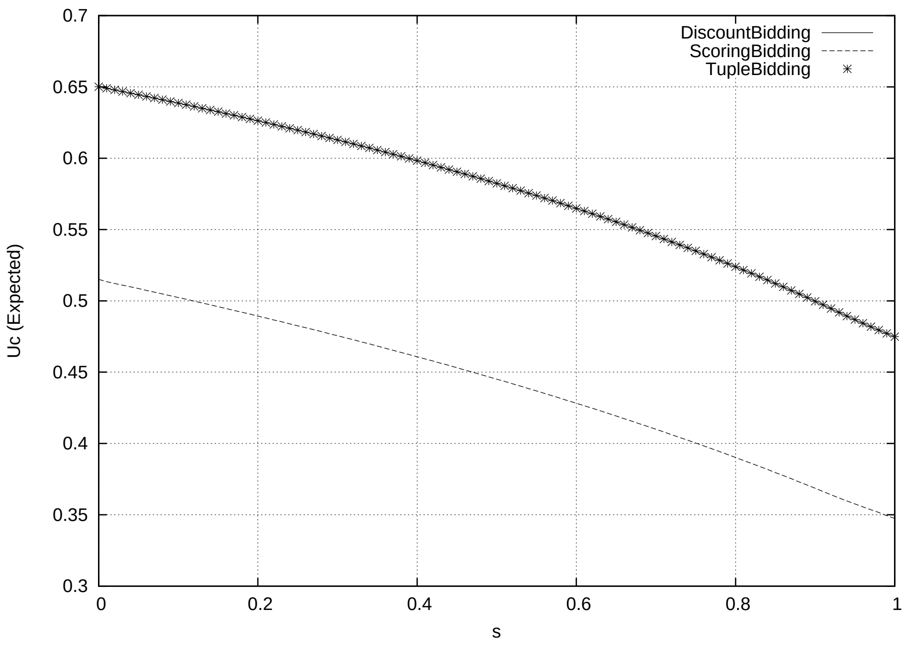


Figure B.4: Comparison of negotiation mechanisms with respect to $\mathcal{C}$'s utility depending on the level of the consumer's risk s for $a = 0.25$, $b = 2$

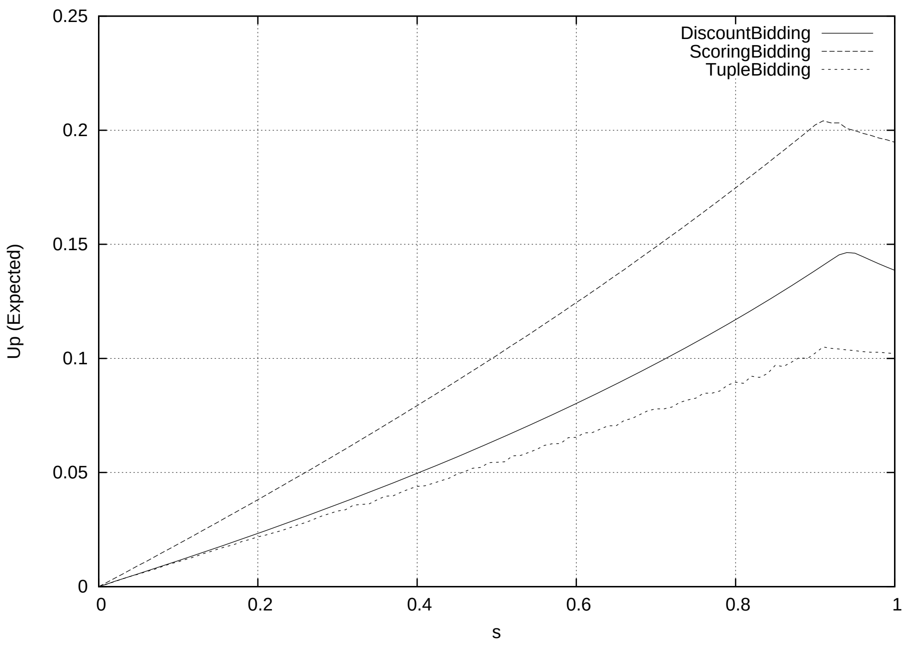


Figure B.5: Comparison of negotiation mechanisms with respect to $\mathcal{P}$'s utility depending on the level of the consumer's risk s for $a = 0.25$, $b = 2$

B.3.2 Results for a=0.5 and b=4

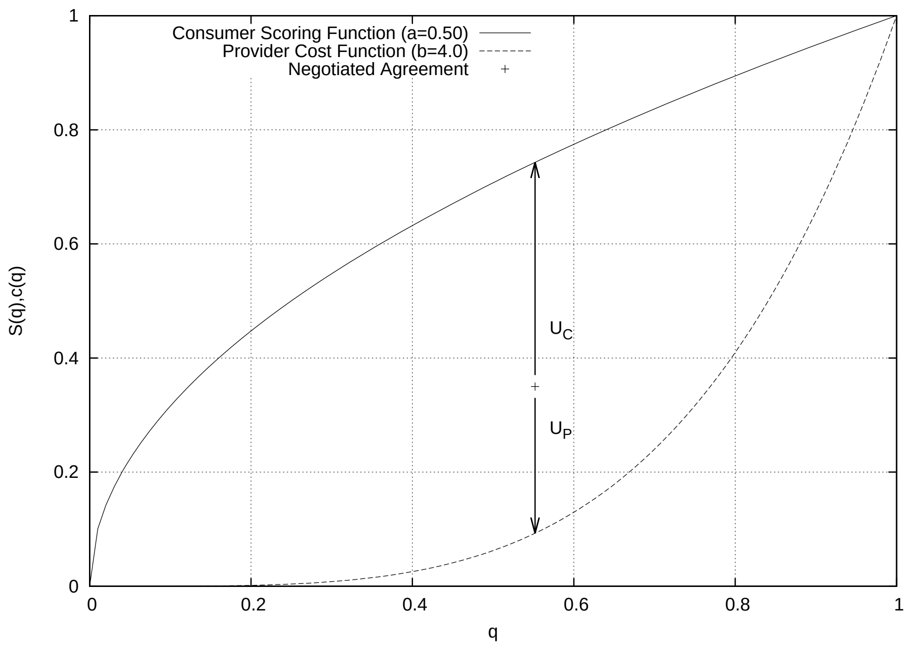


Figure B.6: Scoring and cost functions, optimal quality, price and utilities for $a = 0.5$, $b = 4$

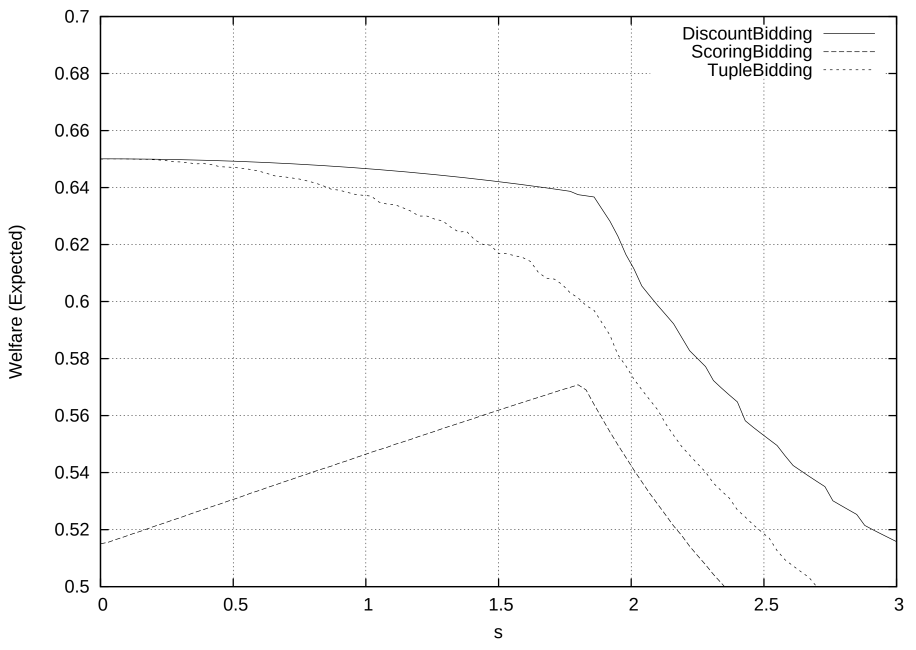


Figure B.7: Comparison of negotiation mechanisms with respect to welfare depending on the level of the consumer's risk s for $a = 0.5$, $b = 4$

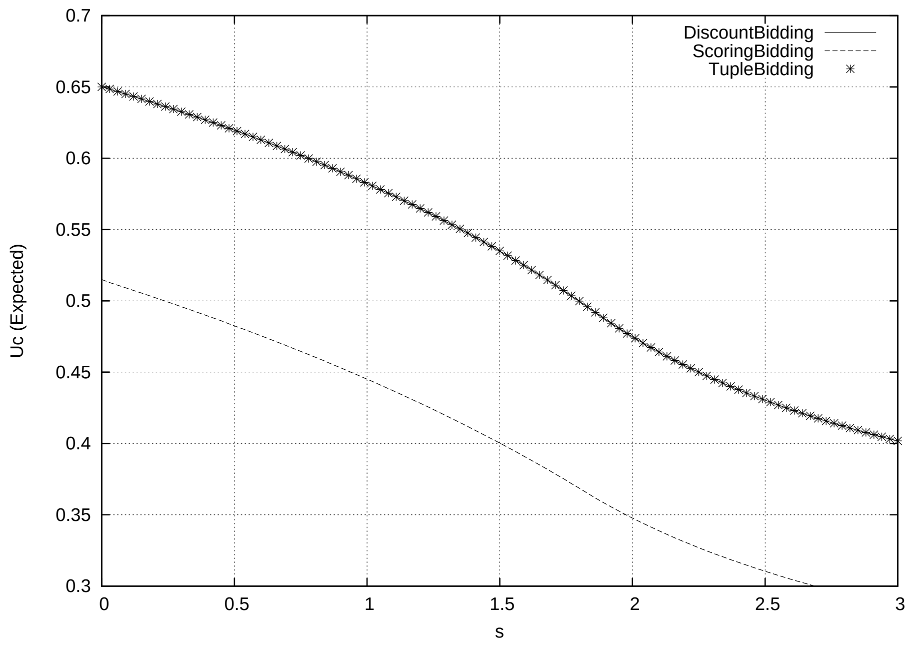


Figure B.8: Comparison of negotiation mechanisms with respect to $\mathcal{C}$'s utility depending on the level of the consumer's risk s for $a = 0.5$, $b = 4$

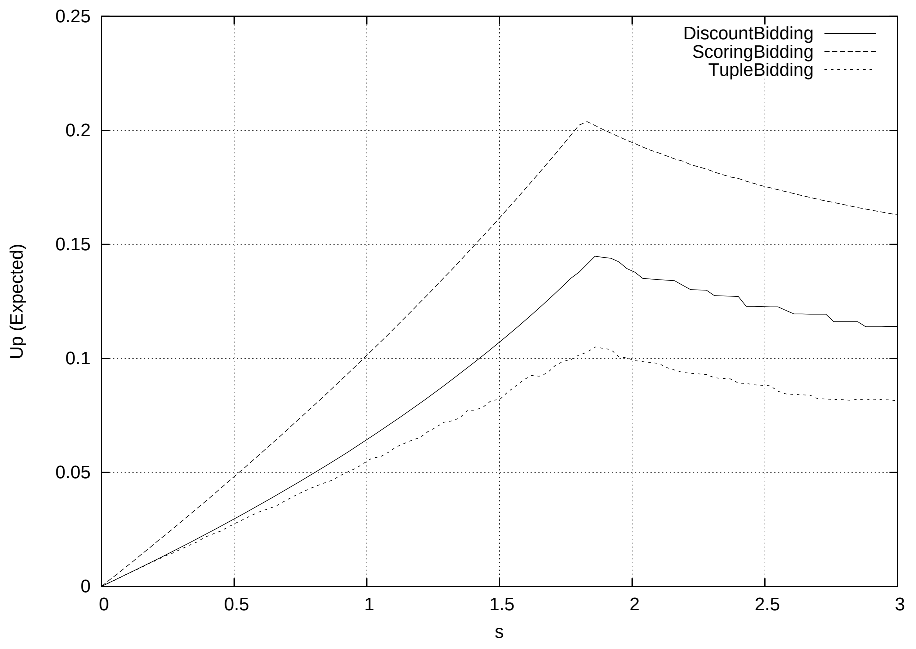


Figure B.9: Comparison of negotiation mechanisms with respect to $\mathcal{P}$'s utility depending on the level of the consumer's risk s for $a = 0.5$, $b = 4$

B.3.3 Results for a=0.25 and b=4

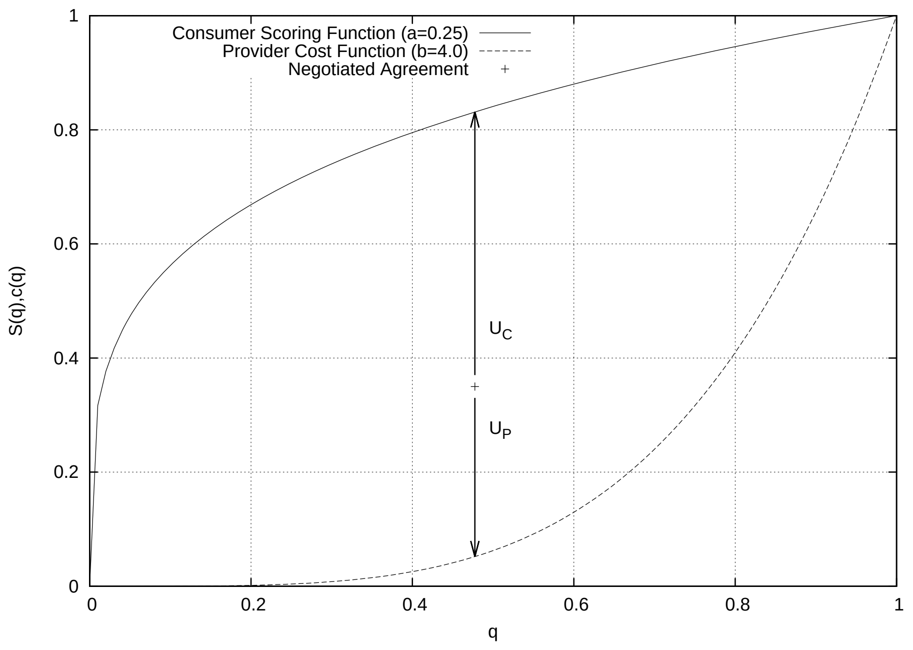


Figure B.10: Scoring and cost functions, optimal quality, price and utilities for $a = 0.25$, $b = 4$

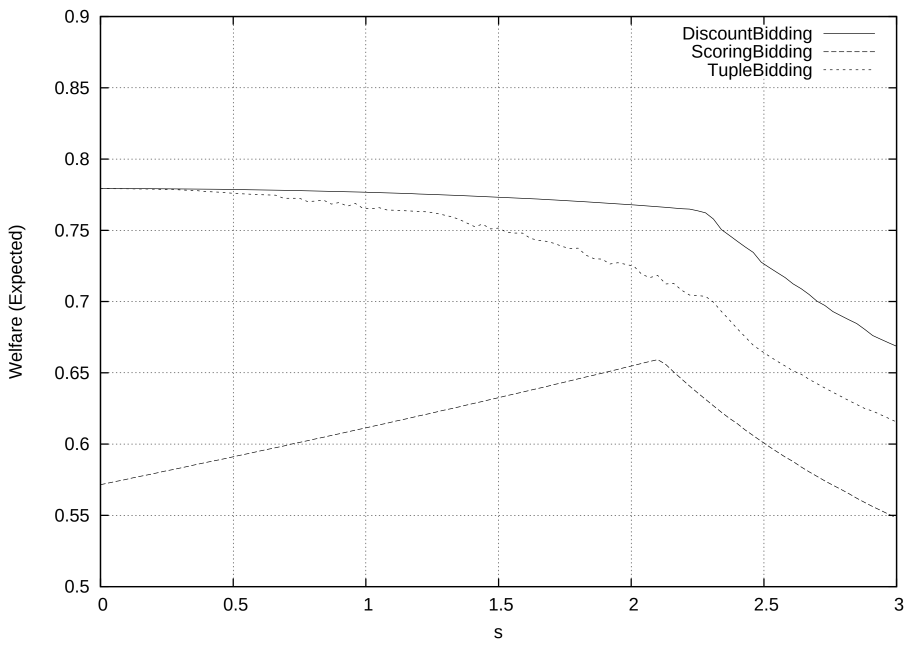


Figure B.11: Comparison of negotiation mechanisms with respect to welfare depending on the level of the consumer's risk s for $a = 0.25$, $b = 4$

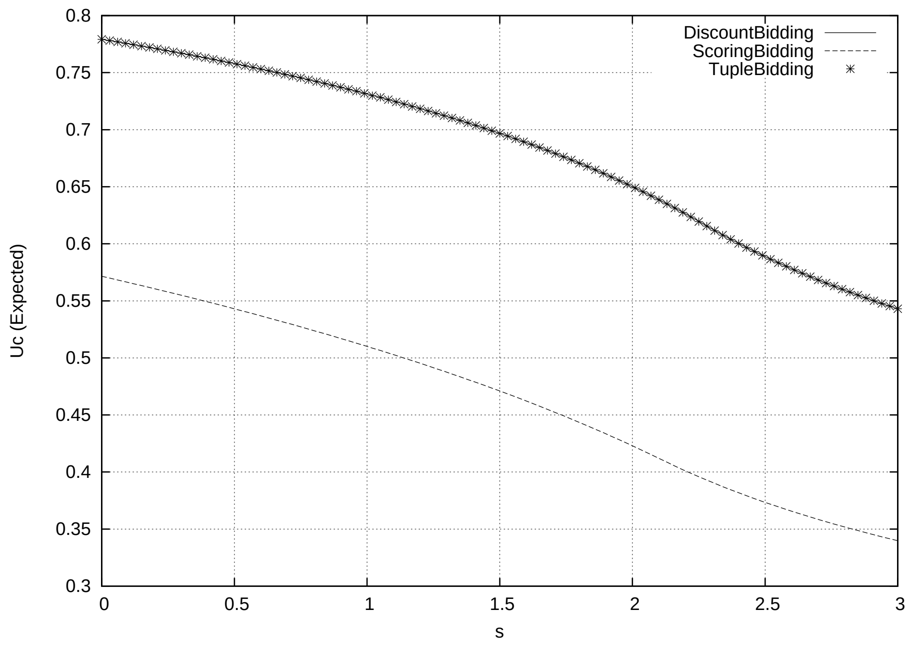


Figure B.12: Comparison of negotiation mechanisms with respect to $\mathcal{C}$'s utility depending on the level of the consumer's risk s for $a = 0.25$, $b = 4$

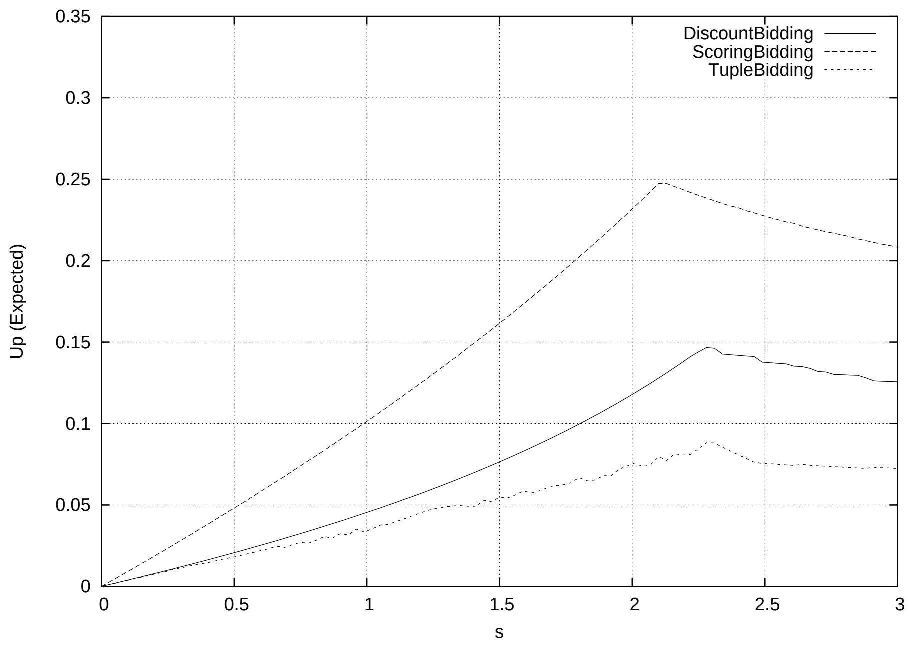


Figure B.13: Comparison of negotiation mechanisms with respect to $\mathcal{P}$'s utility depending on the level of the consumer's risk s for $a = 0.25$, $b = 4$

B.3.4 Parameters and Corresponding Optima

a	b	q^*	w^*
0.5	2	0.3969	0.4725
0.25	2	0.3048	0.6501
0.5	4	0.5520	0.6501
0.25	4	0.4774	0.7793

References

[1] Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint driven web service composition in METEOR-S. In: In Proceedings of IEEE International Conference on Services Computing. pp. 23–30 (2004)

[2] Alavi, M., Leidner, D.: Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. MIS quarterly pp. 107–136 (2001)

[3] Anderson, C.: The Long Tail: Why the Future of Business Is Selling Less of More. Hyperion (2006)

[4] Ardito, C., Barricelli, B., Buono, P., Costabile, M., Lanzilotti, R., Piccinno, A., Valtolina, S.: An ontology-based approach to product customization. Proceedings of the 3rd International Conference on End-User Development pp. 92–106 (2011)

[5] Aristotle: The complete works of Aristotle: The revised Oxford translation, vol. 1. Bollingen Foundation (1984)

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010)

[7] Asker, J., Cantillon, E.: Properties of Scoring Auctions. The RAND Journal of Economics 39(1), 69–85 (2008)

[8] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The description logic handbook: theory, implementation, and applications. Cambridge University Press (2003)

[9] Baader, F., Horrocks, I., Sattler, U.: Description logics. Foundations of Artificial Intelligence 3, 135–179 (2008)

[10] Barberà, S., Jackson, M.: Strategy-proof exchange. Econometrica 63(1), 51–87 (1995)

[11] Basole, R., Rouse, W.: Complexity of service value networks: Conceptualization and empirical investigation. IBM Systems Journal 47(1), 53–70 (2008)

[12] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref/` [Last visited: 15.10.12] (Feb 2004)

[13] Bechhofer, S., Volz, R., Lord, P.W.: Cooking the Semantic Web with the OWL API. In: The Semantic Web – ISWC 2003: Second International Semantic Web Conference, Sanibel Island, FL, USA. pp. 659–675 (2003)

[14] Behrendt, M., Blau, B., Breiter, G., Haak, S., Studer, R., Van Dinther, C., Weinhardt, C.: Semantic-and Preference-Based Planning of Cloud Service Templates (Oct 12 2011), US Patent App. 13/271,575

[15] Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)

[16] Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Service Composition Based on Behavioral Descriptions. Int. J. of Cooperative Information Systems 14(4), 333–376 (2005)

[17] Bichler, M., Kalagnanam, J.: Software Frameworks for Advanced Procurement. Communications of the ACM 49(12), 104–108 (2006)

[18] Bichler, M., Kaukal, M., Segev, A.: Multi-attribute auctions for electronic procurement. In: Proceedings of the First IBM IAC Workshop on Internet Based Negotiation Technologies. Yorktown Heights (March 1999)

[19] Bichler, M., Kersten, G., Strecker, S.: Towards a structured design of electronic negotiations. Group Decision and Negotiation 12(4), 311–335 (2003)

[20] Bichler, M., Pikovsky, A., Setzer, T.: An Analysis of Design Problems in Combinatorial Procurement Auctions. Business and Information Systems Engineering 1(1), 111–117 (2009)

[21] Bichler, M., Kalagnanam, J.: Configurable offers and winner determination in multi-attribute auctions. European Journal of Operational Research 160(2), 380–394 (January 2005)

[22] Blake, M., Cummings, D.: Workflow Composition of Service Level Agreements. In: Proceedings of the IEEE International Conference on Services Computing. pp. 138–145. Salt Lake City (2007)

[23] Blau, B., Krämer, J., Conte, T., van Dinther, C.: Service Value Networks. In: Hofreiter, B., Werthner, H. (eds.) Proceedings of the 11th IEEE Conference on Commerce and Enterprise Computing (CEC). pp. 194–201. Vienna (2009)

[24] Blau, B., Lamparter, S., Haak, S.: remash! - Blueprints for RESTful Situational Web Applications. In: Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009) (2009)

[25] Blau, B., Lamparter, S., Neumann, D., Weinhardt, C.: Planning and Pricing of Service Mashups. In: 10th IEEE Joint Conference on E-Commerce Technology (CEC 2008) and Enterprise Computing, E-Commerce and E-Services (EEE 2008). Washington, D.C. (July 2008)

[26] Blau, B.: Coordination in Service Value Networks - A Mechanism Design Approach. PhD dissertation, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften (2009)

[27] Blau, B., Conte, T., Weinhardt, C.: Incentives in Service Value Networks – On Truthfulness, Sustainability, and Interoperability. In: ICIS 2010 Proceedings. Saint Louis, Missouri, USA (12 2010), paper 8

[28] Blau, B., van Dinther, C., Conte, T., Xu, Y., Weinhardt, C.: How to Coordinate Value Generation in Service Networks – A Mechanism Design Approach. Business and Information Systems Engineering (BISE) 1(5), 343–356 (October 2009)

[29] Bock, J., Haase, P., Ji, Q., Volz, R.: Benchmarking OWL reasoners. In: Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense (ARea08), CEUR-WS. p. 119 (2008)

[30] Bolton, P., Dewatripont, M.: Contract theory. MIT press (2005)

[31] Brachman, R., Levesque, H.: The Tractability of Subsumption in Frame-Based Description Languages. Proceedings of the National Conference on Artificial Intelligence (AAAI'84), Austin, TX, USA pp. 34–37 (1984)

[32] Brachman, R., Schmolze, J.: An overview of the KL-ONE knowledge representation system. Cognitive science 9(2), 171–216 (1985)

[33] Branco, F.: The design of multidimensional auctions. RAND Journal of Economics 28(1), 63–81 (1997)

[34] Buyya, R., Yeo, C., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. pp. 5–13 (2008)

[35] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 74–83. ACM, New York, NY, USA (2004)

[36] Chandrashekar, T., Narahari, Y., Rosa, C., Kulkarni, D., Tew, J., Dayama, P.: Auction-Based Mechanisms for Electronic Procurement. IEEE Transactions on Automation Science and Engineering 4(3), 297–321 (2007)

[37] Che, Y.: Design competition through multidimensional auctions. RAND Journal of Economics 24, 668–680 (1993)

[38] Chvátal, V.: Linear programming. WH Freeman (1983)

[39] Clarke, E.: Multipart pricing of public goods. Public Choice 11(1), 17–33 (September 1971)

[40] Conte, T., Blau, B., Knapper, R.: Networked Mechanism Design – Incentive Engineering in Service Value Networks as Exemplified by the Co-Opetition Mechanism. In: Proceedings of the 16th Americas Conference on Information Systems (AMCIS). Lima (2010)

[41] Corcho, O., Gómez-Pérez, A.: A Roadmap to Ontology Specification Languages. In: Dieng, R., Corby, O. (eds.) EKAW. Lecture Notes in Computer Science, vol. 1937, pp. 80–96. Springer (2000)

[42] Da Silveira, G., Borenstein, D., Fogliatto, F.: Mass customization: Literature review and research directions. International Journal of Production Economics 72(1), 1–13 (2001)

[43] Dakin, R.: A tree-search algorithm for mixed integer programming problems. The Computer Journal 8(3), 250–255 (1965)

[44] Dijkstra, E.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1(1), 269–271 (1959)

[45] Dong, M., Yang, D., Su, L.: Ontology-based service product configuration system modeling and development. Expert Systems with Applications (2011)

[46] Eclipse Foundation: SMILA - Unified Information Access Architecture, `http://www.eclipse.org/smila/`, [Last visited: 15.10.12]

[47] Fay, S.: Selling an opaque product through an intermediary: The case of disguising one's product. Journal of Retailing 84(1), 59–75 (2008)

[48] Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California (2000)

[49] Fitzsimmons, J.A., Fitzsimmons, M.J.: Service management. McGraw-Hill, 6. international edn. (2008)

[50] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (Jul 1987)

[51] Gadrey, J.: The characterization of goods and services: an alternative approach. Review of Income and Wealth 46(3), 369–387 (2000)

[52] Gibbard, A.: Manipulation of Voting Schemes: A General Result. Econometrica 41(4), 587–601 (July 1973)

[53] Gimpel, H., Mäkiö, J., Weinhardt, C.: Multi-Attribute Double Auctions in Financial Trading. In: Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC 2005). pp. 366–369. IEEE Computer Society, Los Alamitos (2005)

[54] Gomory, R.: Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society 64(5), 275–278 (1958)

[55] Green, J., Laffont, J.: Characterization of Satisfactory Mechanisms for the Revelation of Preferences for Public Goods. Econometrica 45(2), 427–438 (1977)

[56] Green, P., Rao, V.: Conjoint measurement for quantifying judgmental data. Journal of Marketing Research pp. 355–363 (1971)

[57] Grossman, S., Hart, O.: An analysis of the principal-agent problem. Econometrica: Journal of the Econometric Society pp. 7–45 (1983)

[58] Groves, T.: Incentives in Teams. Econometrica 41(4), 617–631 (1973)

[59] Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5(2), 199–220 (1993)

[60] Guitart, J., Beltran, V., Carrera, D., Torres, J., Ayguade, E.: Characterizing Secure Dynamic Web Applications Scalability. In: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. p. 108a (2005)

[61] Güth, W., Schmittberger, R., Schwarze, B.: An Experimental Analysis of Ultimatum Bargaining. Journal of Economic Behavior and Organization 3, 367–388 (1982)

[62] Haak, S.: Custom Cloud Services – An economic model for optimizing service configuration and provisioning. In: Eymann, T. (ed.) Proceedings of the Doctoral Consortium of the Wirtschaftsinformatik 2011. Lehrstuhl für Wirtschaftsinformatik, Univ. Bayreuth, Bayreuth (2011)

[63] Haak, S., Blau, B.: Efficient QoS Aggregation in Service Value Networks. In: Proceedings of the 45th Annual Hawaii International Conference on System Sciences. Grand Wailea, Maui (2012)

[64] Haak, S., Gimpel, H.: Individualized Quality-Differentiated Services – A Market Model and Comparison of Negotiation Mechanisms. In: Proceedings of the 11. Internationale Tagung Wirtschaftsinformatik. Leipzig (2013), (forthcoming)

[65] Haak, S., Grimm, S.: Towards Custom Cloud Services - Using semantic technology to optimize resource configuration. In: Proceedings of the 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece. Springer (2011)

[66] Haak, S., Menzel, M.: Autonomic Benchmarking for Cloud Infrastructures - An Economic Optimization Model. In: Proceedings of the 1st IEEE/ACM Workshop on Autonomic Computing for Economics. Karlsruhe, Germany (2011)

[67] Haase, P., Stojanovic, L.: Consistent evolution of OWL ontologies. In: Proceedings of the Second European Semantic Web Conference, Heraklion, Greece (2005)

[68] Henss, J., Kleb, J., Grimm, S.: A Database Backend for OWL. In: Hoekstra, R., Patel-Schneider, P.F. (eds.) OWLED. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2008)

[69] Hesse, W., Kaschek, R., Mayr, H.C., Thalheim, B.: Ontologien in der und für die Softwaretechnik. In: Rumpe, B., Hesse, W. (eds.) Modellierung. LNI, vol. 45, pp. 269–270. GI (2004)

[70] Hill, T.: On goods and services. Review of Income and Wealth 23(4), 315–338 (1977)

[71] Hill, T.: Tangibles, intangibles and services: a new taxonomy for the classification of output. Canadian Journal of Economics 32(2), 426–446 (1999)

[72] van Hoeve, W.: Operations Research Techniques in Constraint Programming. Ph.D. thesis, Tepper School of Business (2005)

[73] Holm, D., Eriksson, K., Johanson, J.: Business Networks and Cooperation in International Business Relationships. Journal of International Business Studies 27(4), 1033–1053 (1996)

[74] Horrocks, I., Motik, B., Wang, Z.: The HermiT OWL Reasoner. In: OWL Reasoner Evaluation Workshop (ORE 2012) (2012)

[75] Hurwicz, L.: On informationally decentralized systems. In: McGuire, C., Radner, R. (eds.) Decision and Organization, pp. 297–336. North-Holland, Amsterdam (1972)

[76] Hurwicz, L.: On the existence of allocation systems whose manipulative Nash equilibria are pareto-optimal. In: 3rd World Congress of the Econometric Society (1975)

[77] Hurwicz, L., Walker, M.: On the Generic Nonoptimality of Dominant-Strategy Allocation Mechanisms: A General Theorem That Includes Pure Exchange Economies. Econometrica 58(3), 683–704 (May 1990)

[78] Jackson, M.: Mechanism theory. In: Derigs, U. (ed.) The Encyclopedia of Life Support Systems. EOLSS Publishers, Oxford (2003)

[79] Jaeger, M., Rojec-Goldmann, G., Muehl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC). pp. 149–159. Monterey (2004)

[80] Jerath, K., Netessine, S., Veeraraghavan, S.: Revenue management with strategic customers: Last-minute selling and opaque selling. Management science 56(3), 430–448 (2010)

[81] Junker, U., Mailharro, D.: The Logic of ILOG (J)Configurator: Combining Constraint Programming with a Description Logic. In: Proceedings of the IJCAI Workshop on Configuration. vol. 3, pp. 13–20. Citeseer (2003)

[82] Karimi, V.: Semantic Web Rule Language (SWRL). `http://www.cs.uwaterloo.ca/~gweddell/cs848/Vahid.pdf` [Last visited: 15.10.12] (2008)

[83] Kay, M.: XSL Transformations (XSLT) Version 2.0 (Januar 2007), `http://www.w3.org/TR/2007/REC-xslt20-20070123/` [Last visited: 14.07.12]

[84] Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)

[85] Klein, M., Noy, N.: A Component-based Framework for Ontology Evolution. In: Proc. Workshop on Ontologies and Distributed Systems, IJCAI 2003 (Acapulco, Mexico) (2003)

[86] Klein, M., Fensel, D.: Ontology Versioning on the Semantic Web. In: Proc. 1st Int. Semantic Web Working Symp. pp. 75–91. Stanford University, CA, USA (2001)

[87] Knapper, R., Blau, B., Speiser, S., Conte, T., Weinhardt, C.: Service Contract Automation. In: Proceedings of the 16th Americas Conference on Information Systems (AMCIS). Lima (2010)

[88] Kolovski, V., Parsia, B., Sirin, E.: Extending the SHOIQ(D) Tableaux with DL-safe Rules: First Results. In: Parsia, B., Sattler, U., Toman, D. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 189. CEUR-WS.org (2006)

[89] Kotha, S.: From mass production to mass customization: the case of the national industrial bicycle company of Japan. European Management Journal 14(5), 442–450 (1996)

[90] Kothari, A., Parkes, D., Suri, S.: Approximately-strategyproof and tractable multiunit auctions. Decision Support Systems 39(1), 105–121 (2005)

[91] Laffont, J., Martimort, D.: The theory of incentives: the principal-agent model. Princeton University Press (2001)

[92] Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: Proceedings of the 16th international conference on World Wide Web. pp. 1013–1022. ACM Press New York, NY, USA (2007)

[93] Lamparter, S.: Policy-based Contracting in Semantic Web Service Markets. Ph.D. thesis, Universität Karlsruhe (TH) (2007)

[94] Land, A., Doig, A.: An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society pp. 497–520 (1960)

[95] Lanthaler, M., Gutl, C.: Towards a RESTful service ecosystem. In: Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on. pp. 209–214. IEEE (2010)

[96] Lécué, F., Léger, A.: A formal model for Web service composition. In: Proceeding of the 2006 conference on Leading the Web in Concurrent Engineering. pp. 37–46. IOS Press, Amsterdam, The Netherlands, The Netherlands (2006)

[97] Lerman, K., Minton, S., Knoblock, C.A.: Wrapper maintenance: A machine learning approach. J. Artif. Intell. Res. (JAIR) 18, 149–181 (2003)

[98] Leser, U., Naumann, F.: Informationsintegration. dpunkt.verlag (2007)

[99] Lösch, U., Rudolph, S., Vrandecic, D., Studer, R.: Tempus Fugit - Towards an Ontology Update Language. In: 6th European Semantic Web Conference (ESWC 09). Lecture Notes on Computer Science, vol. 5554, pp. 278–292. Springer-Verlag (Juni 2009)

[100] LSDIS, Large Scale Distributed Information Systems, University of Georgia: METEOR-S: Semantic Web Services and Processes (2004), http://lsdis.cs.uga.edu/projects/meteor-s/, [Last visited: 15.10.12]

[101] Ludwig, A., Franczyk, B.: COSMA – an approach for managing SLAs in composite services. Service-Oriented Computing–ICSOC 2008 pp. 626–632 (2008)

[102] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: OWL-S: Semantic Markup for Web services (2004), http://www.w3.org/Submission/OWL-S/, [Last visited: 15.10.12]

[103] Mas-Colell, A., Whinston, M., Green, J.: Microeconomic Theory. Oxford University Press, New York (1995)

[104] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: Ontology Library. Wonderweb Deliverable D18. (DOLCE) (2003), http://wonderweb.semanticweb.org/

[105] McAfee, R.: A Dominant Strategy Double Auction. Journal of Economic Theory 56(2), 434–450 (1992)

[106] McGuinness, D., van Harmelen, F.: OWL Web Ontology Language Overview (2004), http://www.w3.org/TR/owl-features/, [Last visited: 08.11.12]

[107] Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology 53(6), 50 (2009)

[108] Meng, X., Hu, D., Li, C.: Schema-guided wrapper maintenance for web-data extraction. In: Proceedings of the 5th ACM international workshop on Web information and data management. pp. 1–8. ACM (2003)

[109] Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction. In: Proceedings Eighth National Conference on Artificial Intelligence. pp. 25–32 (1990)

[110] Motik, B.: On the properties of metamodeling in OWL. In: Proceedings of the 4th international conference on The Semantic Web. pp. 548–562. Springer-Verlag (2005)

[111] Motik, B., Patel-Schneider, P., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., et al.: OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation 27 (2009)

[112] Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic aboxes. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 227–241. Springer (2006)

[113] Motik, B., Studer, R.: KAON2–A Scalable Reasoning Tool for the Semantic Web. In: Proceedings of the 2nd European Semantic Web Conference (ESWC), Heraklion, Greece (2005)

[114] Motik, B., Sattler, U., Studer, R.: 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)

[115] Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, 165–228 (2009)

[116] Motta, E., Domingue, J., Cabral, L., Gaspari, M.: IRS–II: A Framework and Infrastructure for Semantic Web Services. The Semantic Web-ISWC 2003 pp. 306–318 (2003)

[117] Muslea, I., Minton, S., Knoblock, C.A.: Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems 4, 93–114 (2001)

[118] Muthusamy, V., Jacobsen, H., Chau, T., Chan, A., Coulthard, P.: SLA-driven business process management in SOA. In: Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research. pp. 86–100. ACM (2009)

[119] Myerson, R.: Incentive Compatibility and the Bargaining Problem. Econometrica 47(1), 61–73 (1979)

[120] Myerson, R.: Optimal auction design. Mathematics of operations research pp. 58–73 (1981)

[121] Myerson, R., Satterthwaite, M.: Efficient mechanisms for bilateral exchange. Journal of Economic Theory 28, 265–281 (1983)

[122] Myerson, R.: Mechanism design. Discussion Papers 796, Northwestern University, Center for Mathematical Studies in Economics and Management Science (September 1988)

[123] Nash, J.: The (Dantzig) simplex method for linear programming. Computing in Science & Engineering 2(1), 29–31 (2000)

[124] Noy, N.F., Musen, M.A.: Ontology Versioning in an Ontology Management Framework. IEEE Intelligent Systems 19, 6–13 (2004)

[125] Papazoglou, M., Georgakopoulos, D.: Service-Oriented Computing. Communications of the ACM 46(10), 25–28 (2003)

[126] Parkes, D.: Iterative combinatorial auctions: achieving economic and computational efficiency. Ph.D. thesis, University of Pennsylvania (2001)

[127] Parkes, D., Kalagnanam, J.: Models for Iterative Multiattribute Procurement Auctions. Management Science 51(3), 435–451 (2005)

[128] Parkes, D., Kalagnanam, J., Eso, M.: Achieving Budget-Balance with Vickrey-Based Payment Schemes in Combinatorial Exchanges (2001), iBM Research Report

[129] Parsia, B., Clark, K.G.: UMD Mindlab Rules Workshop Position Paper. `http://www.w3.org/2004/12/rules-ws/paper/81/` (2004), [Last visited: 08.11.12]

[130] Parsia, B., Sirin, E., Grau, B.C., Ruckhaus, E., Hewlet, D.: Cautiously approaching SWRL. http://www.mindswap.org/papers/CautiousSWRL.pdf (2005), [Last visited: 08.11.12]

[131] Pine, B., Davis, S.: Mass customization: the new frontier in business competition. Harvard Business School Press (1999)

[132] Pueschel, T., Neumann, D.: Management of Cloud Infastructures: Policy-Based Revenue Optimization. ICIS 2009 Proceedings p. 178 (2009)

[133] Raiffa, H.: The Art and Science of Negotiation. Harvard University Press, Cambridge (1982)

[134] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Applied Ontology 1(1), 77–106 (2005)

[135] Rubinstein, A.: Perfect equilibrium in a bargaining model. Econometrica: Journal of the Econometric Society pp. 97–109 (1982)

[136] Saaty, T.: How to make a decision: the analytic hierarchy process. European journal of operational research 48(1), 9–26 (1990)

[137] Sabin, D., Freuder, E.: Configuration as composite constraint satisfaction. In: Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop. pp. 153–161 (1996)

[138] Seaborne, A., Manjunath, G.: SPARQL/Update - A language for updating RDF graphs. http://www.w3.org/Submission/SPARQL-Update/ (2008), [Last visited: 08.11.12]

[139] Shearer, R., Motik, B., Horrocks, I.: HermiT: A highly-efficient OWL reasoner. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008). pp. 26–27 (2008)

[140] Sikka, V., Färber, F., Lehner, W., Cha, S., Peh, T., Bornhövd, C.: Efficient transaction processing in SAP HANA database: the end of a column store myth. In: Proceedings of the 2012 international conference on Management of Data. pp. 731–742. ACM (2012)

[141] Silva, G.: APT howto (2003), http://www.debian.org/doc/manuals/apt-howto/index.en.html, [Last visited: 08.11.12]

[142] Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Web Semantics: science, services and agents on the World Wide Web 5(2), 51–53 (2007)

[143] Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web 1(4), 377–396 (2004)

[144] Stojanovic, L.: Methods and Tools for Ontology Evolution. Ph.D. thesis, University of Karlsruhe, Germany (2004)

[145] Studer, R., Benjamins, V., Fensel, D.: Knowledge Engineering: Principles and methods. Data & Knowledge Engineering 25, 161–197 (1998)

[146] Tsarkov, D., Gardiner, T., Horrocks, I.: Framework For an Automated Comparison of Description Logic Reasoners. In: Proceedings of the 5th International Semantic Web Conference (ISWC) (2006)

[147] Unger, T., Leymann, F., Mauchart, S., Scheibler, T.: Aggregation of Service Level Agreements in the Context of Business Processes. In: Proceedings of the 12th Enterprise Distributed Object Computing Conference (EDOC). pp. 43–52. Munich (2008)

[148] Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. The Journal of Finance 16(1), 8–37 (1961)

[149] Walton, R.E., McKersie, R.B.: A Behavioral Theory of Labor Negotiations. McGraw-Hill, New York (1965)

[150] Wellman, P.: Online Marketplaces. In: Singh, M. (ed.) The Practical Handbook of Internet Computing, pp. 1–17. CRC Press, Boca Raton (2005)

[151] Wirth, S.: Individuelle Massenware kommt aus dem Internet. `http://www.welt.de/wirtschaft/article4625748/Individuelle-Massenware-kommt-aus-dem-Internet.html` (September 2009), [Last visited: 08.11.12]

[152] World Wide Web Consortium (W3C): Web Services Architecture Requirements. `http://www.w3.org/TR/wsa-reqs/` (Feb 2004), [Last visited: 08.11.12]

[153] Yang, D., Dong, M., Miao, R.: Development of a product configuration system with an ontology-based approach. Computer-Aided Design 40(8), 863–878 (2008)

[154] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for Web services composition. Software Engineering, IEEE Transactions on 30(5), 311–327 (2004)