Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)
genehmigte Dissertation von
Dipl.-Inf. Martin Junghans.

Methods for Efficient and Accurate Discovery of Services

Martin Junghans

Tag der mündlichen Prüfung: 27.11.2013
Referent: Prof. Dr. Rudi Studer
Korreferent: Prof. Dr. Hansjörg Fromm

Karlsruhe 2014

Abstract

The discovery of services is of one of the most integral parts of a service-oriented system. Discovery is the problem of identifying services from a pool of service descriptions that fulfill the requirements of a discovery request. With an increasing number of services developed and offered in an enterprise setting or the Web, users can hardly verify their requirements manually in order to find the appropriate services. Automated discovery methods can support users in discovering appropriate services that they have not been aware of before.

The ability to discover services effectively depends on how services are advertised, how requirements can be expressed, and how the requirements are verified. It is challenging to develop service discovery methods that can be applied in a wide variety of use cases, while providing a good trade-off between expressivity and efficiency.

In this thesis, we develop a method to discover semantically described services. The discovery method exploits comprehensive service and request descriptions that capture functional and non-functional properties. In our discovery method, we compute the matchmaking decision by employing an efficient model checking technique. Our logic-based discovery method automatically identifies accurate matches for a given request. The proposed method can be applied to services that describe their complete behavior in the form of executable process expressions. In addition, we introduce an alteration of the method tailored to discover services that cannot disclose their complete behavior and provide an interface description instead.

In order to facilitate service discovery in large bodies of offered services, we propose approaches for more efficient matchmaking. Formal service classes enable an automated and consistent service classification and induce a class hierarchy, which can be utilized as an offline index structure. While each class in the index is described by a given request, we also propose indexing structures that can be automatically populated either offline or online, i.e., during the processing of incoming requests. The offline indexes accelerate reasoning tasks by materializing possible propositions in advance. The online index aims at caching frequent requests such that repetitive requests can be processed faster. Our contributions are based on scenarios from current fields of research and have been implemented and evaluated in the context of large-scale research projects.

Acknowledgements

This work would not have been possible without the support and guidance of many persons. First and foremost I want to thank my advisor Prof. Dr. Rudi Studer for giving me the opportunity and granting the freedom to do this research. It has been an honor to be a member of his group. He has taught me a lot about research and management, shared his inspirations and has always been an excellent example of a superior man. Further, without the support, guidance and patience of Dr. Sudhir Agarwal, this study would not have been completed. The joy and enthusiasm they both showed for their research were contagious and motivational for me.

I also want to thank Prof. Dr. Dirk Riehle, who has been a great promoter of my endeavor to pursue a degree in the area of Semantic Web technologies. Moreover, I would like to thank Prof. Dr. Hansjörg Fromm and Prof. Dr. Stefan Tai for their interest, many fruitful discussions, and their consent to serve on the board of examiners.

All the members of the research group Knowledge Management at the AIFB as well as the colleagues from the KSRI continuously provided inspiration and constructive criticism that helped to improve my work. I am truly grateful for these valuable discussions and debates in a relaxed and fun environment.

I would also like to extend my deepest gratitude to my wife Conny for her unconditional patience and support throughout this time as well as Christel and Werner Junghans, my parents, who have always supported, encouraged and believed in me, in all my endeavors.

Contents

Chapter 1

Introduction

Service-oriented computing is an interdisciplinary paradigm that revolutionizes the very fabric of distributed software development, including not only complex enterprise applications, but also scientific, mobile, telecommunication, and embedded system-based applications amongst others.

Applications that build on services as building blocks of their architecture can adapt to a changing or unpredictable environment more easily and evolve during their life span. When implemented properly, services can be discovered and invoked dynamically using non-proprietary mechanisms, while each service can still be implemented in a black-box manner. This is important from a business perspective, since each service can be implemented using any technology, independently of the others. What matters is that everybody agrees on a common integration technology, and there is a consensus about this in today's middleware market: customers want to use Web technologies.

The success encountered by the Web has shown that tightly coupled software systems require prior agreements and a shared context between communicating systems [Pap08], whereas loosely coupled software systems can be more flexible, more adaptive and often more appropriate in practice. Loose coupling makes it easier for a given system to interact with other systems, possibly legacy systems that do not have a lot in common with it. Web services lie at the crossing of distributed computing and loosely coupled systems.

Service orientation enables an effective means for offering and consuming functionalities within and across organizations. Despite the promises of the adoption of services, service integrators, developers, and providers need to create techniques and tools to support cost-effective development, as well as the use of dependable services and service-oriented applications. The discovery of services is just one of the techniques that are required for the realization of service-oriented applications. Without automated, sometimes called dynamic, service discovery the loose coupling is restricted to a large extent. The main objectives of service orientation cannot be achieved without automated service discovery techniques.

From a technical viewpoint, the coupling between different components of a service based software system is loosened by the potential of Web services to replace them easily. Discovering other appropriate Web services dynamically is a key challenge and promises to reduce the tightness of the coupling between the individual components. In the beginning, service directories such as the CORBA Naming Service [Sie00] allow for lookups based on the service name, which requires the client to know the service name in advance. More dynamic service discovery services like UDDI [UDD01] provide more flexible but still keyword-based lookups.

As more and more Web services and Web-mediated services have been made public, more sophisticated discovery mechanisms are required. It is not possible anymore to assume that client users have prior knowledge about the names, terminologies, or providers of desired services. Automated service discovery methods aim at reducing the manual effort required to match requests with available services. Ideally, no manual effort is then required to evaluate the applicability of discovery results for the intended use. Especially when a large number of services are offered, any manual effort required for finding and binding services becomes infeasible.

To foster the automation of any service-related task in service-oriented systems, Semantic Web Services [SGA07, FFST11] emerged as a combination of Web services [ACKM04, Pap08] and Semantic Web technologies [HKR09]. A semantic, i.e., machine-interpretable, description of services allows for automated reasoning about services while resolving data and semantic heterogeneity using ontologies without dictating a global terminology. The additional semantic description of the functionality and the behavior of services furthermore allows reasoning over and finding services based on their properties, which further reduces the human effort of evaluating the discovery results.

In recent years, Web services gained a lot of attention within the Semantic Web community. Many formalisms such as DAML-S [BHL$^+$02], OWL-S [MBH$^+$04], and WSMO [dBBD$^+$05] with varying expressivity and complexity for describing Web services semantically have been proposed. Many service discovery approaches and systems that built on these description frameworks have been proposed in the literature. Within the Semantic Web community, the interest in Semantic Web Services seems to stagnate. This may be due to a focus on traditional procedure-oriented Web services in service-oriented architectures and the lack of the promised or expected millions of available Web services on the Web [Abe09, SLB09]. We believe that certain problems need to be solved before service orientation is not just used in enterprise systems but also finds a broader adoption on the Web.

In this thesis, we advance the extensive amount of existing works in the area of semantic service discovery with several novel contributions to the field. We further aim at solving existing problems such as formally modeling and reasoning over the behavior of atomic service profiles and services with complex interaction patterns in a unified way. We also aim at an efficient service discovery framework with the ability to compute logically correct discovery results in order to enable a high degree of automation for the use of the discovered services in tasks like service invocation and composition.

In the remainder of the chapter, we introduce in Section 1.1 the main hypothesis and research questions of our work. We also briefly introduce the approaches taken to address the individual research questions. In Section 1.2, we outline the structure of this thesis and give a comprehensive summary of contributions presented in this work.

1.1 Main Hypothesis and Research Questions

The goal of this thesis is to develop automated, effective, and efficient service discovery methods that can be applied in many scenarios using the functionalities offered on the Web. The development of distributed or component-based systems and Web applications from various domains benefits from such service discovery methods as the adoption of services, among other benefits, increases the flexibility of the developed systems and makes economic sense. We express our overall goal in the main hypothesis as follows.

Main Hypothesis: Services can be automatically, effectively, and efficiently discovered based on a verification of the constraints of expressive service requests over comprehensive service descriptions.

Approach: Our service discovery method builds on semantic, i.e., machine-interpretable, descriptions of services and requests in order to foster automation. The targeted automated discovery method shall not require human efforts during the matchmaking phase in which a given request is matched against the available descriptions of services.

The effectiveness of a service discovery method depends on the use case in which discovery is applied and on how the discovered services are going to be used. In our work, we derive requirements from scenarios that aim at an automated use of services (e.g., invocation and composition of services). Therefore, we measured the effectiveness of a discovery method by the accuracy of the discovery results. The applicability of discovered services is guaranteed if all constraints of a request are fulfilled, i.e., the results are logically correct. Therefore, we develop logic-based matchmaking methods that produce correct discovery results.

An efficient service discovery method is able to deal with large repositories of service descriptions and delivers results in a reasonable time with the use of commodity hardware.

We substantiate the main hypothesis by the following research questions and give an outline of how we will approach them in this thesis. Each research question highlights a different facet of the main hypothesis while being more specific and constraining with respect to the requirements of service discovery.

Research Question 1: Can we develop a service discovery method that considers functional and non-functional properties in a unified way as well as detailed and abstract service behavior descriptions in service descriptions and requests?

Approach: We establish a property-based service model, which captures functional and non-functional properties, and apply this upper service model to service descriptions and service requests. Different degrees of detail of a service behavior description are modeled as characteristics of the functional properties of service descriptions and requests. Then, the service discovery method treats services of different complexity (with respect to the degree of detail of behavior descriptions) conceptually equal.

Research Question 2: How can we support the description of the functionality of state-changing services, which cause effects during their execution, such that reasoning and discovery based on state changes in descriptions and requests is enabled?

Approach: We clarify the semantics of service descriptions and requests in order to capture the dynamics of services. While a state-based formal model of the service functionality is a proper means to model state changes caused by services, many existing description formalisms provide limited support for describing the service effects. We therefore extend the formalisms such that state changes can be logically modeled in descriptions and, thus, discovery over state changes is enabled.

Research Question 3: Is it possible to apply an efficient logic-based matchmaking technique (model checking) to the matchmaking of service interface (profile) descriptions without losing the ability to model and reason about state changes?

Approach: We introduce a model checking based matchmaking approach to the discovery of Web service profiles, as the model checking techniques excel in efficiently evaluating the constraints on the described models. We present a request formalism that allows us to base our matchmaker on a model checking technique.

Research Question 4: Is it possible to automatically classify services correctly? And can such a classification be used to increase the efficiency of the discovery approach while the discovery result accuracy is not compromised?

Approach: We develop a service classification approach that automatically and consistently classifies services. It derives a classification hierarchy based on formal class definitions. As a classification of services into formal classes can be used to precompute and materialize discovery results, the use of classes increases the discovery efficiency while preserving automation and result accuracy.

Research Question 5: Can index structures be used to reduce the computational complexity of service discovery during the matchmaking of service requests with available service descriptions?

Approach: We develop index structures that materialize precomputed intermediate results of the service discovery. The indexes can be built prior to the matchmaking phase, i.e., before the discovery request is processed (i.e., offline), or while requests are processed (i.e., online). As the reuse of intermediate results from the indexes allows reducing the computational effort during the query answering time, the results can be computed faster and the efficiency is increased.

1.2 Contributions and Outline

In this work, we develop a method to discover semantically described services with a model checking based matchmaking approach, which we apply to atomic services and services with a complex observable behavior in the same way. The discovery method exploits comprehensive service and request descriptions that capture functional and non-functional aspects. We introduce in Chapter 2 three concrete scenarios that require a service discovery method. Our scenarios comprise (i) the development of service-based Web applications, (ii) the provision of services in the logistics domain, and (iii) smart Web browsing based on a process centric view of the Web. Based on the individual characteristics of each scenario, we derive design choices and requirements for the development of our discovery method.

In Chapter 3, we introduce preliminaries from the domains of knowledge representation and explain the notion of services as we consider them in the scope of this work. We recapitulate the basic concepts of existing service description frameworks that model the service behavior and more abstract profile of the behavior. A summary of a process calculi for the formal description of observable behavior and of a temporal logic for the formal description of behavior properties are given, before a model checking based matchmaking technique is reviewed.

In Chapter 4, we first introduce our property-based service model to express comprehensive service descriptions. Then, we develop a request formalism with the appropriate expressivity to query the properties of a service description. We present a model checking based matchmaking for both description and request formalisms and report on its implementation, its application in the context of the WisNetGrid project, and the measured evaluation results.

The discovery approach from Chapter 4 can be applied to services in general. The matchmaking method considers complex behavior descriptions as functional properties. In order to extend the applicability of our discovery method to services that cannot disclose a description of their behavior, we focus in Chapter 5 on the modeling and matchmaking of the service functionality. We consider the functionality in particular as it is an abstract profile of the behavior that can be used to describe the functional capabilities of service interfaces. The functionality of services has gained a lot of attention in previous modeling and discovery approaches. We present an extension to model state changes logically correct in service descriptions and requests. An extensive overview of the state of the art in this area is given in order to highlight the differences of our approach. At the end of Chapter 5, we report on the implementation, its application in the SOA4All project, and the measured evaluation results.

Based on the experienced performance of the matchmaking, we develop in Chapter 6 an offline classification of services in order to gain efficiency in discovery. The developed service classification builds on formally defined classes, which enables the development of automated and logically consistent methods for the classification of services and the computation of a class hierarchy. We show how service classes can also be used to simplify the process of describing and requesting services as well as describe service properties implicitly. Implicit property descriptions are an extension to our service description formalism, which allows us to describe services even if precise property values are not known or shall not be disclosed. At the end of Chapter 6, we describe how our service classification impacts the discovery performance, and close with a discussion on the assumption and applicability of the classification-based discovery method.

In Chapter 7, we introduce two offline and one online index for use within the discovery method. The offline indexes are precomputed and, in contrast to our classification, do not require a given classification hierarchy. The online index is an online data structure that we incorporate to our discovery method in order to collect frequent discovery requests that can be cached, thus increasing the efficiency when requests are posed repeatedly. We evaluate the performance gain by the introduction of the index structures.

We conclude this thesis with a summary of the findings and contributions and a discussion of opportunities for prospective work in Chapter 8.

Chapter 2

Scenarios and Requirements

In this chapter, we present three different scenarios where the discovery of services is fundamental to realize them. The scenarios are carefully chosen to represent a broad range of use cases of a service discovery method. They also differ in their sets of requirements on a discovery method and in the emphasis they put on individual requirements. Incorporating all the requirements identified in the scenarios into our work allows us to develop a discovery method that can be applied in a multitude of future applications. The applicability of a discovery method is characterized by its usefulness and adequateness to fulfill the requirements of the applications.

Our scenarios are presented in Section 2.1. They are based on the research carried out in several research projects. The first scenario aims at the development of Web applications by enabling the utilization of existing Web resources and services as envisioned in the SOA4All project.[1] The development process of Web applications is based on software development principles from service-oriented architectures (SOA). Thus, a service discovery system is a basic component of the infrastructure for delivering services and Web applications and, thus, is an integral part of the Web application development process. The service discovery approach we developed in this project was introduced in [JAS10, JA10].

The second scenario similarly demands for a discovery method, but the requirements are substantially different in the domain of logistic services. Here, the discovery is an enabler for the provision of logistic services that combine real-world services with ICT-based services and have to adhere to the constraints of customers and providers. This scenario was one of the use cases in the InterLogGrid project.[2] Our service discovery method was developed in the WisNetGrid project,[3] presented in [JAS12, AJ11], and successfully applied and evaluated in

[1]Service Oriented Architectures for All (SOA4All) is a large-scale integrating project funded by the European Seventh Framework Programme, under the Service and Software Architectures, Infrastructures and Engineering research area; see http://www.soa4all.eu, retrieved 2013-08-15.

[2]InterLogGrid is a BMBF-funded German research project that developed grid technologies for logistics companies and supports planning and scheduling decisions in intermodal and multilateral logistics; see http://www.interloggrid.org, retrieved 2013-08-15.

[3]WisNetGrid is a BMBF-funded German research project that created a unified knowledge space in the grid. Therefore, methods and tools enabling collaborative work in a heterogeneous environment have been developed; see http://www.wisnetgrid.org, retrieved 2013-08-15.

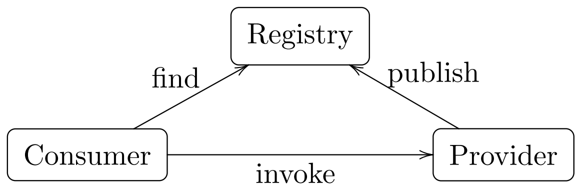


Figure 2.1: SOA triangle: Publish-find-bind pattern for dynamic service discovery and invocation

the context of both projects.

Our third scenario emerged from an internal research project focusing on the intelligent management and usage of processes and services (or “suprime” in short). The aim is to align and integrate several research results in the areas of services and processes from the group at AIFB in order to, for example, simplify and increase the potential of future Web browsing. Based on a more formal view of functionalities, services, and processes offered on the Web (e.g., in the form of programmatic interfaces or Web pages), the browsing experience of users can be extended to the point of an alternative approach to information search. We presented our developments within this scenario in [AJ10, JAS12, JA13, AJ13].

We studied the requirements and successfully applied our discovery method in further scenarios like cyber-physical systems and smart energy [WJSH11] that are part of a future Internet of Things/Services. The scenarios presented in the following are a representative selection that allows us to identify general discovery requirements. In Section 2.2, we present the requirements identified by analyzing each scenario. In Section 2.3, we describe our approach to address each requirement and discuss our design choices for the discovery approach.

2.1 Scenarios

2.1.1 Web Application Development

In our first scenario, we investigate the development of Web applications based on the reuse of existing Web-mediated services. We studied this scenario in the context of the SOA4All research project. One of the goals was to develop a comprehensive framework, infrastructure, and platform for service delivery using semantic Web technologies and principles from Web 2.0 and service-oriented architectures [KNSP09]. The following project use cases utilized the core technologies like the service search system to develop new Web applications: an e-government Web portal, a social Web portal of a telecommunication service provider, and a shopping application based on the Facebook application platform.

With the trend of shifting focus from static Web pages to more dynamic and interactive pages that can offer the functionality of applications, software engineering techniques and especially service orientation became increasingly adopted in the area of Web application development [Jaz07]. Web applications bridge the back-end information systems to the front-end hypermedia representation and provide functionality to end users who consume the offered functionality of the application in commodity Web browsers [Fra99].

In Web application development, the service-oriented architecture [NL04, PvdH07] is a software development pattern, which enables the development of networked systems based on (Web) services as its building blocks. The services provide certain defined functionality, e.g., access to and retrieval of information, simple or complex computations. The SOA triangle in Figure 2.1 shows the setup implemented in SOAs for dynamically publishing, finding, binding, invoking, and executing services. Service providers develop and offer services and publish the descriptions of their services at a registry (service description repository). Consumers in the client role can discover services at the registry and invoke services of the providers.

Service-based systems that have a loose coupling in comparison to more tight coupled monolithic systems are easier to enhance and adapt to changing requirements during their lifetime. Therefore, services abstract from their internal behavior and technical implementation details, and (semantic) Web technologies are used as a common middleware enabling the integration of services. The development of Web applications (or apps) can be cost-effective and quick when existing data sources and functionalities, for instance offered via Web APIs, REST- or SOAP-based Web services, are reused and integrated into the new application.

Consider, for example, the first use case of the SOA4All project: a platform for the development of e-government Web portals, which can be easily adapted to the needs of different public authorities. While the application of a particular city offering such a portal can be considered to be fairly static over time, a singular design still involves locating appropriate services from a limited set of available services. The set is limited due to strict regulations in the public domain. In general, the platform has to be flexible and dynamic enough to support the adaptation of e-government Web portals to the specific needs of different instantiations. E.g., in order to let portal designers dynamically choose which functionalities to include, a discovery method that reasons about functional properties is required. For example, it has to allow finding business services from the SAP service catalog and complementary third party services. In order to simplify the integration process by reducing the manual effort, the discovery method has to return only accurate results that provide the desired functionality. This feature excludes keyword-based service discovery methods, because these approaches need to be very restrictive on service descriptions and requests in order to guarantee such high result accuracy. In the context of the project, the discovery results, i.e., services providing basic functionality of the portal, were then combined by a design time composition component. A process editor also allowed for the manual specification or refinement of control and data flow among the services. Accurate discovery results were required for the design time composition as well as for the manual modeling as the editor did not provide facilities to inspect the discovery results to estimate their applicability for a given purpose.

The other two use cases of the project included more dynamic Web applications that can be often adapted, refined, or extended. These use cases deal with users being involved in developing shops upon the Facebook platform and with a telecommunication provider often extending its offers by adding further functionalities. When existing systems and Web applications are extended, the additional services should be easily integrated into the existing service system. Automatic discovery methods have to be used, as an unmanageable number of services are offered on the Web. Furthermore, multifaceted constraints over the functionality, the behavior, and the exchanged information need to be considered by a discovery method.

Besides a data integration challenge, the success of the new Web applications also depends on the ability to find appropriate APIs for accessing external data, services for the transformation

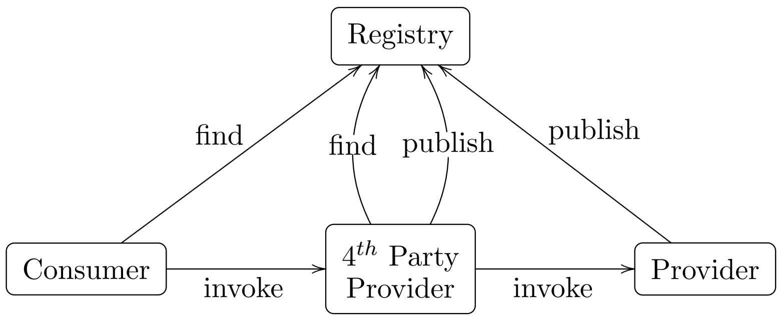


Figure 2.2: The introduction of a fourth-party provider role in logistics scenario

of data, and complementary functionalities like payment services. SOA4All showed that the integration of third-party services can be simplified and become more flexible by the use of semantic Web technologies as an integrating middleware.

We can summarize that the service discovery method has to automatically identify matches from a set of given service descriptions and a given request. Due to the openness of the Web and the distribution of the development and provision of services, the discovery method shall deal with heterogeneous descriptions and requests. Additionally, the results have to be accurate, i.e., they have to provide at least the requested functionality in order to be directly applicable.

2.1.2 Provision of Logistic Services

Logistic services can quickly become very complex. They involve multiple means and modalities of transportation, many service providers, and intermediaries. In the logistics industry, a fourth-party provider is a provider that creates new services by composing existing services, which are provided by actual logistic service providers (third-party providers). A fourth-party provider only consumes the services by designing solutions on behalf of the customer. The customer's requirements are expressed in terms of a contract between customer and fourth-party provider. In Figure 2.2, we extended the SOA triangle from Figure 2.1 by an additional role of the fourth-party provider that consumes services by creating service compositions and also provides these compositions as services to the consumers.

Therefore, it is not sufficient that a fourth-party provider offers uniform services to all customers. Adaptation of offered services to specific requirements implied by the transported goods, spatial conditions, legal restrictions and many more are necessary. The possibility to cost-efficiently and quickly integrate existing services into the business processes and service value networks [BKCvD09] of a fourth-party logistic provider largely depends on finding electronic as well as real-world services from this domain [LKS11, KKLF11]. For example, in order to integrate given services into a service network quickly, i.e., with low manual effort for adaptation, services that provide a behavior that is compatible to the behavior of the service network are preferable.

A discovery method has to locate third-party services at the behest of the fourth-party service provider based on functional and non-functional constraints. For instance, services that plan and optimize transport routes, track expenses and file invoices, and transport the goods of certain characteristics frequently occur in processes that fourth-party providers offer to customers. Also, there are many requirements besides the provided functionality that need to be considered in order to compare similar services with each other [KLS13]. The price, dispatch time, trust and user rating of a third-party logistic service are just a few examples of relevant non-functional properties that have to be considered by the discovery method.

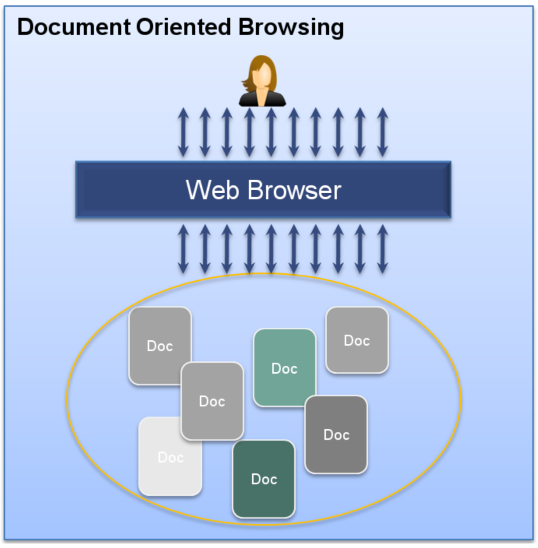
In comparison to the previous scenario, it is hard for humans to keep track of providers and offered services from multiple domains like the actual transportation, accounting, planning and optimization of routes, et cetera. Automated support for the discovery of services is mandatory. Furthermore, a distinction of services that provide equivalent functionalities becomes important in order to be able to select appropriate services which may suit or allow optimizing the overall business process. Non-functional properties support the selection process and shall be part of descriptions and considered for discovery. Another observation was that simple services as well as entire business processes can be integrated by fourth-party providers into new services. In order apply a discovery method to find any of these services, the varying complexity has to be supported in a unified way.

2.1.3 Smart Web Browsing

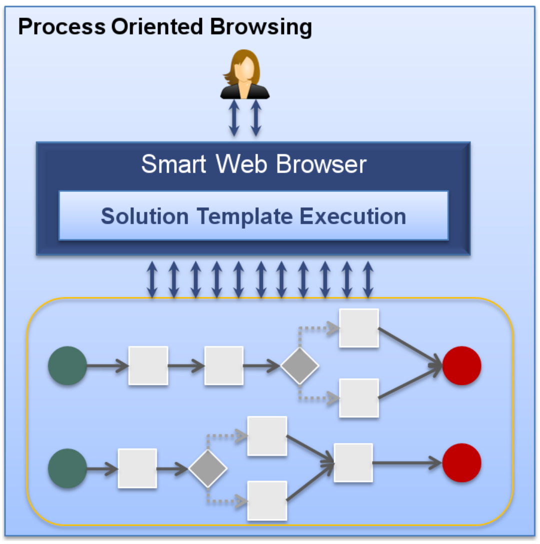
Complex tasks like making travel arrangements or shopping for birthday gifts are conducted in the Web and often involve several websites. Dealing with such tasks quickly becomes cumbersome as logical dependencies between the information entered to and received from different Web pages have to be managed manually. E.g., end users have to enter the dates and locations of a travel into multiple portals offering flights, hotels, rental cars, etc. It is even more complex if multiple websites with similar functionality are involved in order to compare prices or to receive more offers. Logical dependencies, like the pickup time and location of a rental car that depend on the chosen flight, introduce constraints on the order in which different Web pages can be queried. As depicted in Figure 2.3a, many interactions between users and the Web pages are required. Also, the coordination of the pages remains manual effort.

By a *Web browsing process* we mean the steps users perform in order to interact with the tremendous amount of functionality hidden in the Deep Web [Ber01]. This functionality is typically offered to users via dynamically generated Web pages. In order to obtain the desired functionality in the Deep Web, a user needs to perform certain steps, e.g., submitting Web forms filled with appropriate information. Thus, Web browsing processes are contrary to the static and data centric view on Web pages, which has been the main focus of the Semantic Web community until now.

In order to support users to handle the amount of information and business provided by websites and Web applications, search engines like Google, Yahoo! and Bing have been developed. The formal models underlying such search engines mainly consider the content of the static Web pages for building their respective search indexes. However, they hardly consider the dynamic aspects of the Web pages (such as the user interactions, data and control flow in browsing processes) in general. As a result, currently, it is hard to find websites that offer certain functionality or to find the desired information, which is returned on a page generated dynamically at some later stage in the process and not on the very first (mostly static) page. We are aware that search engines aim at integrating the data from Deep Web data sources and learning to access the data even through Web page interactions [MKK$^+$08]. However, they capture dynamic information to a very limited extent, e.g., the weather of a given location



(a) Document-oriented Web browsing



(b) Smart Web browsing

Figure 2.3: Manual interaction effort with present-day and smart Web browser

or information about an actors fetched from a movie database. The limited capability to integrate dynamic data indicates the lack of a generic model capturing the dynamics of the functionalities provided by Web pages.

Currently, users have to aggregate results of various Web browsing processes or take care of entering the same data in multiple forms as well as controlling the data flow between different Web browsing processes manually, even though most of such mediation work can be automated. Considering that many tasks that the users accomplish with the help of multiple Web browsing processes need to be performed again and again, supporting a user with automated techniques in coordinating the Web browsing processes can save a lot of human effort as well as eliminate errors.

Modeling the dynamics of browsing processes offers many benefits regarding user support and automation for further tools. Building on a process-oriented view on the Web functionalities, a smart Web browser can take advantage of the process descriptions and support end users in coping with logical dependencies, mediation, control and data flow issues. A smart Web browser, as depicted in Figure 2.3b, reduces manual efforts by automating the coordination of Web browsing processes. We gained this automation by computing solution templates, i.e., controlling processes that control and interact with the individual Web browsing processes. Further, we applied automated techniques for composing multiple browsing processes into a solution template that can handle complex tasks like the ones described above. Our service composition method is heavily based on an effective discovery method [AJ10], which treats atomic Web services and services with complex behavior that describe browsing processes, Web applications, and interactive Web pages alike.

Typically, service composition suffers from state-space explosion, a problem that is caused by the combinatorial explosion of possibilities of combining services for a given goal. In our approach in [AJ10], we restricted the number of combinations by dropping completeness of the set of computed solution templates and employing a discovery method that guarantees that the results provide a minimal functionality. That is, by adding a service returned by the discovery method to the solution template, we gradually reduce the complexity of the goal and the composition terminates after only few iterations (depending on the complexity of the goal).

An Alternative Approach to Information Search

Our smart Web browsing approach based on a process-oriented view on the functionalities offered on the Web not only simplifies complex end user tasks, it can also be exploited to develop a novel approach to information search.

Information search on the Web can become tedious if the desired information is scattered across multiple websites. Static websites can be reached by search engine crawlers and their content can be indexed to provide end users with efficient search methods. However, in many cases, end users are still required to do a lot of manual work to compile the required information. Consider for example an end user who is interested in knowing the names of the chairs of a particular track at the previous WWW conferences. As of today, Web search engines do not deliver satisfactory results for queries similar to "track chairs of all WWW conferences". In order to obtain the required information, the end user has to pose multiple queries to a search engine, browse through the hits, and aggregate the required information fragments outside of the found Web pages. The case of dynamic websites is even more complex. Accessing the information hidden in the Deep Web is in itself an open challenge for search engines.

Current search engines focus on finding the most relevant Web pages for a given information need rather than providing the information itself. The ranking of the Web pages is usually based on the link structure provided by their providers. As a result, a user receives a list of Web pages with similar content even though the information need of the user might require pages with complementary information.

End users need help in selecting the pages that are relevant for obtaining the information scattered across multiple Web pages. Such help must contain at least the set of pages that the end user should visit, as well as support for easily invoking all the pages of the set. More advanced help could support the complete end user browsing process including support for data flow between the user and the pages as well as among the pages, and control flow if there are data dependencies among inputs and outputs of Web pages in the set. In order to achieve this, we compute a list of hits for a given information need, where each hit consists of a set of pages. For each page, we need to know which information need it satisfies and a path that needs to be executed in order to reach the page.

We apply our service discovery method to search for complex behavior descriptions of browsing processes. Discovery request criteria describe the desired information to which the browsing processes lead and the behavior of the browsing process. The latter criterion is relevant as it allows including or excluding browsing processes that, for example, require multiple interactions or expect users to provide certain information. In Chapter 7, we will further examine this scenario as motivation for the development of an efficient discovery technique for finding Web browsing recipes from large repositories.

In both smart Web browsing and information search scenarios, a discovery method needs to deal with services with heterogeneous and complex behavior descriptions, non-functional properties to model rating and trust of shared browsing processes. Due to the number of expected browsing processes, the discovery efficiency becomes crucial.

2.2 Requirements Analysis

From the three scenarios above, we can identify requirements that discovery methods must fulfill. The main requirements are listed below.

Requirement R1: Automation
Discovering services that provide a desired functionality and may also fulfill further constraints is a time-consuming and error-prone task if performed manually. Large repositories with an unmanageable number of service descriptions need to be considered by service discovery in the above scenarios. Human efforts in service discovery have to be avoided if we want to support scenarios with potentially many services.

An automated method for service discovery determines and returns services without any human intervention when a valid request is given. Our work aims at automation of (i) methods for matching a given request with a given service description and (ii) methods to coordinate the matchmaking of a set of given service descriptions. Obviously, in order to develop automated matchmaking methods, machine-readable and machine-interpretable descriptions of service offers and service requests are necessary.

Our understanding of automated service discovery does not comprise further steps related to the discovery problem, such as automated requirement and preference elicitation [SS97, SPM06, MA10]. A discovery method receives and processes given requests.

Requirement R2: Accuracy
Automated discovery (as claimed in R1) is one of the requirements of automated utilization of the results. The (automated) computation of logically correct results with respect to the requirements and constraints specified in a request is the prerequisite for enabling automated techniques, which take the result as an input for further use. That is, automated binding, invocation, and composition of services in dynamic service systems are possible if the results provide the expected functionality and behavior, which is the prerequisite for successful interaction with a dynamically bound and integrated service.

It is important that the discovery result (at least) fulfills any requirements of a request. That is, results may additionally fulfill further requirements, provide properties, or comply with constraints that were not specified in a request. Services that do not fully comply with the constraints specified in a request are not considered to be an accurate match for the request. Consequently, it needs to be provable that a discovery method produces logically correct (i.e., sound) results.

We justify our rather strict choice by the objective of an automated use of results. The results that do not completely satisfy a request can be helpful to solve a given problem in some cases. However, then the discovery method would further require the capability of automated service composition, which is still a challenging task [GNT04, GNT14]. In Section 4.2.1, we elaborate on our motivation for a strict definition of discovery result accuracy.

Requirement R3: Expressivity
The formalism used to describe services has to be expressive enough such that it facilitates a precise description of various service properties. A high expressivity allows for comprehensive

descriptions of services and their properties, which can be utilized in manifold scenarios and use cases. Our scenarios from Section 2.1 revealed that various aspects of a service are used differently or are useful to a different degree in different scenarios. More expressive formalisms provide a higher flexibility with the potential to model more service properties and to support more use cases. Hence, there is a higher chance that expressive service description formalisms are appropriate in many use cases.

Service descriptions have to model the functionality as well as non-functional properties (like the quality of service, service level agreement expressions, pricing, usage constraints, etc.) in general. It should also allow for the description of the service behavior. A precise specification of constraints expressed in a request formalism is subject to the analogous requirements regarding expressivity. Discovery based on property constraints demands an expressivity of the formalism used to describe constraints in service requests that is aligned to the expressivity of the service description formalism. A request comprises combinations of inclusion and exclusion of properties of a desired service. In addition to that, it is convenient to provide the ability to express alternative configurations of desired services.

Requirement R4: Heterogeneity

Distributed systems and service-oriented computing inherently have to resolve heterogeneity issues, often on several levels. Within the context of service discovery, data and semantic heterogeneity [Hal05] arise when service requests are matched against service descriptions.

Services are developed and provided by different service providers, who can be independent of each other (especially in open settings like the Web). Therefore, the service providers need to be able to describe their services with domain-specific vocabulary that can be independent from the ones used in other descriptions, by other providers or requesters.

Hence, no institutional (global) scheme, vocabulary, or taxonomy to describe services and service requests can be dictated. It cannot be assumed that a service requester has any knowledge of existing services in advance. Consequently, there cannot be a prior agreement on the terminology to be used within requests.

Requirement R5: Efficiency

Last in the sequence but not least in importance is the requirement for efficient discovery methods. Efficiency means that the discovery method should deliver results quickly (i.e., have a short query answering time).

Due to the (envisioned) broad applicability of our discovery method to manifold scenarios with varying characteristics and requirements, it is impossible to specify universal metrics or constraints on the targeted discovery performance. In discovery use cases like the development of service-based systems and the composition of services during design phases in the service life-cycle, a response within a few minutes is justifiable. However, for discovery systems that target end users, as in our smart Web browsing scenario for example, a response should not take more than a minute such that the discovery system remains interactive. In other scenarios that rely on service composition at run-time, e.g., in order to adapt to a changing environment or to replace a broken service, the discovery results sometimes have to be returned in only a few seconds or less.

Although the efficiency of a discovery method is determined by the query answering time, there are further constraints, e.g., on computing resources like memory requirements, that must be taken into account. Often, the assignment of more computing resources leads to a decreased query answering time and a perceived higher efficiency. However, an efficiency increase by assigning more computing resources is always limited, as computing resources are always limited. Therefore, efficiency gains of discovery methods shall rather be achieved by improvements of used algorithms or the chosen system design than of technical details.

2.3 Design Choices and Approaches

In this chapter, we presented three scenarios that highlight the need for a discovery method. The scenarios demonstrated that a generic discovery method should address various requirements in order to be useful and applicable in different settings.

The discovery method that is developed in this work should adhere to all requirements we gathered from the various use cases, in order to achieve broad applicability. In the remainder of this chapter, we explain how we approach the requirements. Specifically, we present our design choices and contributions. Then, we discuss how they fulfill each of the outlined requirements.

Automated service discovery methods. Our work is based on machine-interpretable and formal descriptions of (i) offered services and (ii) constraints in service requests. Machine interpretability assumes that the descriptions of offers and requests are serialized in a machine-readable representation and that there is a relation between this representation and a corresponding formalism, which assigns a meaning to the described matter.

We apply formalisms to model relevant service properties that are part of service descriptions. Formalisms are necessary to facilitate automation. Properties that describe static service properties are commonly modeled with description logics [BCM$^+$03], i.e., decidable fragments of first-order logics. The semantics of service functionality and behavior cannot be modeled by knowledge representation formalisms like description logics. We therefore apply a process calculus, which allows for a description of concurrent systems with interactions and communications between agents or processes with a few primitives and operators [Hen88]. In addition, we use description logics again as the formalism for the description of resources within a process [Aga07a].

Analogously, we apply description logics in service requests in order to formally model static desired service properties. The desired functionality and behavior is formally described by a temporal logic, also combined with description logics for a meaningful description of resources in desired behavior descriptions.

Based on the formal semantics defined for description logics, process calculus, and temporal logics, we are able to develop automated methods for reasoning over services. Within the formal interpretation of service descriptions (which is based on labeled transition systems and model-theoretic semantics of description logic [Rud11]), we evaluate service requests automatically. Consequently, our discovery method detects matching services for given constraints autonomously and fulfills requirement R1.

Accuracy of discovery results. Given the formal models that underpin service descriptions, service properties, and discovery requests, it is possible to develop algorithms that can also guarantee the accuracy of results. As we identified in requirement R2, only logically correct matches allow immediate, i.e., without the need of further adaptations, and automated use of the discovery results in different use cases.

Discovery methods and systems that find matching services based on similarities or overlaps between a request and described services, e.g., using heuristics or structural similarities, aim at finding potentially useful services while they provide a high flexibility. However, they cannot guarantee immediate applicability of discovery results in general.

The precision[4] of the discovery method should be at 100% in order to be able to fulfill requirement R2. It means that only logically correct discovery results are returned by the method. In order to achieve the precision, we use a logic-based matchmaking approach based on model checking that produces accurate results. The accuracy of model checking was proven in [Ran01] for the temporal logic μ-calculus in general. Furthermore, we inherit the results on the logical correctness of behavior model checking results from [Aga07a].

To summarize, in order to achieve the required accuracy, we employ formal models for the description of properties in service offers and desired properties in requests. Based on these formalisms, we develop a model checking based service matchmaking method that computes logically correct results.

Expressive formalisms for comprehensive descriptions of services and discovery requests. High expressivity, as identified in requirement R3, is on the one hand gained by selecting an expressive description logic for modeling static service properties and process resources, and, on the other hand, by the formalism chosen to model the dynamic behavior of services.

The first aspect has been extensively studied by the Semantic Web community. Different description logics have been developed and studied [BCM+03]. They provide varying expressivity that is sufficient for the purpose of ontological modeling of services (except for expressing their dynamics) and their domains with respect to the use cases of discovery. In this work, we mainly focus on the second issue, which deals with functionality and behavior modeling. We apply a state-based model to capture the dynamic functional service properties and develop a discovery method that logically reasons over them. *Service dynamics* refers to the ability to change the state of the information space and introduce real-world effects during the service execution. It does not refer to dynamically changing description of services.

We also develop a property-based service model that unifies the view on service properties of any type. It allows us to treat functional and non-functional properties equally in service descriptions and requests. Discovery of services based on comprehensive descriptions and constraints is a prerequisite for the applicability of our discovery method to many different scenarios.

Semantic service modeling. We base our work on semantic descriptions of services and requests as this approach inherently enables machine readability, machine interpretability, and ability to translate them into formal model that allow for logic-based matchmaking and retrieval of accurate results.

Beyond these benefits, semantic modeling by means of ontologies copes with the heterogeneity that arises in decoupled and service-based systems. We use ontologies to describe service offers, requests, and domain knowledge. We allow resolving the data and semantic

[4]The precision of a discovery method is the fraction of discovery results that are relevant.

heterogeneity, as described by requirement R4, applying existing mediation and ontology mapping techniques [NVSM07, BHSS09]. As the mediation problem is outside of our scope of developing a discovery method, we do not further highlight this topic. However, the use of description logics as the formalisms to model domain knowledge bases, service descriptions, and requests in the form of ontologies permits to apply existing techniques to mediate between different vocabularies of various providers and consumers.

Service classification and indexing structures. Requirement R5 demands for efficient discovery methods that return accurate results within a feasible time frame. It is evident that requirements such as high expressivity, reasoning within expressive formalisms, and the demand for accurate results add computational complexity to any methods developed for such scenarios. Besides the exponential description logic reasoning complexity,[5] the verification of behavioral constraints over service functionalities adds additional complexity to the discovery problem.

We develop several extensions to the discovery method in order to support all requirements stated above and still retain a feasible and usable discovery system, which can deal with large repositories containing comprehensive service offer descriptions. The first contribution in this context is a classification of services that is used to materialize frequent discovery results by means of formally defined classes (see Chapter 6). A hierarchy over classes serves as an indexing structure during discovery.

Second, we develop in Chapter 7 multiple indexing structures that materialize and, thus, speed up frequent and expensive computations of intermediary results. We propose index structures that are created offline and reduce the query answering time because an index lookup is less time consuming than the computation of the result at query time. Furthermore, we introduce an online index that is continually populated as the discovery requests are processed. This index also materializes intermediate results in order to reduce the computational effort of repeated queries. Due to the reduction of the query answering time that we achieve by the classification as well as the offline and online indexes, we are able to significantly increase the discovery efficiency. This is achieved without compromising the expressivity or accuracy.

[5]The problem of reasoning in description logics, depending on the expressivity of the chosen description logic and the used reasoning tasks, is often in the class of Exptime or even NExptime problems. The complexity class denotes the theoretical upper bound of computational complexity. The experienced complexity in real scenarios can be considerably less, which makes the use of ontologies feasible in practice [MW11].

Chapter 3

Preliminaries

In this chapter, we summarize the main principles of the technologies and techniques that are used for the realization of the service discovery method in this work. The machine-interpretable representation of static information (knowledge) is part of the research in the area of knowledge representation. We introduce the concepts of first-order logics, description logics, and ontologies in Section 3.1. One less expressive and one rather expressive description logic are introduced in more detail as they are applied in the context of service descriptions in our work.

In Section 3.2, we clarify our notion of services as used in this work in the context of service discovery. We recapitulate in Section 3.3 existing service description frameworks that formally, we say semantically, model services on the basis of description logics. The service behavior is a significant part of service descriptions, which cannot be expressed in description logics. Hence, we introduce appropriate formalisms to describe the behavior with the π-calculus and behavior properties with the μ-calculus, before we summarize the findings of the PhD thesis by Agarwal [Aga07a], in which these calculi have been combined with description logics and applied for the service discovery problem. The mentioned thesis serves as a starting point of the present thesis. In Section 3.4, we show how the service discovery problem is solved by matchmaking and model checking techniques.

3.1 Knowledge Representation

Knowledge representation is a principal to represent knowledge by symbols and allow inferring new knowledge. The representation has to enable machines to behave as if they would understand the represented knowledge. Knowledge representation has been broadly studied in the area of Artificial Intelligence research and is used in all kinds of problem solving methods [New82]. The underlying formalism consists of knowledge representation languages in the form of a theoretical model. Formalisms like logics, semantic nets, and rules have been studied and applied to represent knowledge.

A language is an unambiguous and logically adequate representation of natural language statements. Each knowledge representation language has to provide a reasoning capability for inferring knowledge [Woo75]. *Reasoning* is a capability that infers new statements from given statements that represent existing knowledge. However, tractability problems quickly occur even for reasoning over simple (i.e., less expressive) languages [LB87].

Knowledge representation techniques can be used to describe domain models. The *domain knowledge* is a formal representation of the knowledge of a domain expert. Here, it is challenging to delimit the knowledge of a domain. Representing and allowing others to access this knowledge can be similarly difficult [IWA91].

While logic means the study of reasoning in philosophy, the area of ontologies focuses on the study of the state of being. An *ontology* describes the states of being of a particular set of entities and defines a part of the domain model on a conceptual level, i.e., describing the being of instances without including them. Ontologies contain the definition of concepts, roles, and correct inferences [SS09]. We introduce ontologies in Section 3.1.3. The knowledge representations of ontologies depend on the concrete language used to express the model. In Sections 3.1.1 and 3.1.2, we present the fundamentals of first-order and description logics.

3.1.1 First-Order Logic

Philosophically, logic is the study of correct reasoning and can be applied to represent knowledge [LMP07]. The first-order predicate calculus, or first-order logic (FOL), has been shown to be undecidable in general by Gödel's Incompleteness Theorem. In fact, the logical consequence of FOL is semi-decidable. That means there exist first-order logic statements that cannot be proven either true or false.

We introduce the first-order logic by an overview of the basic and compound language elements, and present their interpretation that assigns a truth value to each expression. We refer to the literature, e.g. [HA28, Fit96], for a list of the inference rules like modus ponens, resolution, double negation, and so on. The logic inference problem is the problem of deciding whether a set of formulae (knowledge base KB) semantically entails a formula ϕ, written as $KB \models \phi$. This problem is semi-decidable for the first-order logic.

Syntax

In general, a formal language is a recursively defined set of strings on a fixed alphabet. In first-order logic, the *signature* $\Sigma = P \cup F$ is the union of a set of *predicates* P and a set of *functions* F, where $P \cap F = \emptyset$.

The arity refers to the number of argument places that a predicate or function has. For every integer $n \geq 0$, we have a set P^n of n-place predicates and a set F^n of n-place functions. We call predicates with an arity of $n > 1$ *relations*. *Constants* are functions with arity 0. In contrast to predicates that are interpreted by a Boolean-valued function, functions are assigned to individuals of the domain of discourse.

The set V of individual *variables* is disjoint to the signature. Variables are singular terms that can either describe unspecified objects or express generality. Constants and variables are FOL *terms* that serve as a building block of FOL formulae.

Atomic formulae have the form $Vt_1, ..., t_n$, where $V \in P^n$ is an n-place predicate, and t_1 to t_n are terms. Often, an atomic formula is written as $V(t_1, ..., t_n)$. Let $\top$ and $\bot$ be additional atomic formulae that represent Boolean values True and False, respectively.

Compound formulae are all the formulae that can be constructed by the following rules. The formulae can be structured by parenthesis, logical connectives (symbolized $\wedge, \vee, \neg, \Rightarrow$), the universal quantifier ($\forall$), or the existential quantifier ($\exists$). To disambiguate formulae, the order of decreasing precedence is $(\neg, \wedge, \vee, \Rightarrow, \{\forall, \exists\})$.

- All atomic formulae are formulae.
- If ϕ is a formula, then so is $\neg\phi$.
- If ϕ and ψ are formulae, then so is $\phi \vee \psi$.
- If v is a variable and ϕ is a formula, then $\exists v.\phi$ is a formula.

If ϕ and ψ are formulae, then the conjunction, implication, and universal quantification can be derived from the above operators.

$$\begin{aligned} \phi \wedge \psi &\equiv \neg(\neg\phi \vee \neg\psi) \\ \phi \Rightarrow \psi &\equiv \phi \vee \neg\psi \\ \forall v.\phi &\equiv \neg\exists v.\neg\phi \end{aligned}$$

Free and bound variables. In a formula, the occurrence of a variable can be free or bound. Variables of atomic formulae are free. The connectives do not change the variable status. That is, if a variable v is free (or bound) in a formula ψ then v is free (or bound) in $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$.

Any occurrence of a variable v in a formula ϕ is bound in the scope of the quantifiers in $\forall v.\phi$ and $\exists v.\phi$. The status of the remaining variables of ϕ is unchanged. *Sentences* are formulae that contain no free variables. Bound variables are used to express generality. Free variables can be considered as placeholders.

Example 1. *Assume an entity represented by a variable* v. *Let* b *be a monadic predicate symbol for "is a book",* w *for "is a written work", and* a *for "is available". We can define that every book is also a written work and that there should be some book available by the following sentences.*

$$\begin{aligned} \forall \mathsf{v}.\mathsf{b}(\mathsf{v}) &\Rightarrow \mathsf{w}(\mathsf{v}) \\ \exists \mathsf{v}.\mathsf{b}(\mathsf{v}) &\Rightarrow \mathsf{a}(\mathsf{v}) \end{aligned}$$

Interpretation

The first-order semantic is defined by the interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Let the domain $\Delta^{\mathcal{I}}$ be an abstract and non-empty set of objects of the universe (the domain of discourse), which represent the interpretation of the symbols of the signature Σ.

Terms. The interpretation function $\cdot^{\mathcal{I}}$ assigns to each symbol $s \in \Sigma$ of the signature Σ an element $s^{\mathcal{I}}$ from the domain $\Delta^{\mathcal{I}}$.

$$\cdot^{\mathcal{I}} : \Sigma \times \Delta^{\mathcal{I}}$$

The interpretation functions assign an object $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to every constant $c \in C$. For every n-place predicate symbol $p \in P^n$, an n-ary relation $p^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ is assigned.

Formulae. The meaning $[\![\phi]\!]^{\mathcal{I}}$ of a FOL formula ϕ is a truth value, either $True$ or $False$. It can be inductively defined as follows.

$$\begin{aligned}
[\![\top]\!]^{\mathcal{I}} &= True \\
[\![\bot]\!]^{\mathcal{I}} &= False \\
[\![p(t_1,\dots,t_n)]\!]^{\mathcal{I}} &= True, && \text{iff} && \left([\![t_1]\!]^{\mathcal{I}},\dots,[\![t_n]\!]^{\mathcal{I}}\right) \in p^{\mathcal{I}} \\
[\![\phi \land \psi]\!]^{\mathcal{I}} &= True, && \text{iff} && [\![\phi]\!]^{\mathcal{I}} = True \text{ and } [\![\psi]\!]^{\mathcal{I}} = True \\
[\![\phi \lor \psi]\!]^{\mathcal{I}} &= True, && \text{iff} && [\![\phi]\!]^{\mathcal{I}} = True \text{ or } [\![\psi]\!]^{\mathcal{I}} = True \\
[\![\phi \Rightarrow \psi]\!]^{\mathcal{I}} &= True, && \text{iff} && [\![\phi]\!]^{\mathcal{I}} = True \text{ or } [\![\psi]\!]^{\mathcal{I}} = False \\
[\![\neg\phi]\!]^{\mathcal{I}} &= True, && \text{iff} && [\![\phi]\!]^{\mathcal{I}} = False \\
[\![\forall v.\phi]\!]^{\mathcal{I}} &= True, && \text{iff} && [\![\phi_{v\leftarrow c}]\!]^{\mathcal{I}} = True \text{ for all } c \in C \\
[\![\exists v.\phi]\!]^{\mathcal{I}} &= True, && \text{iff} && \text{there exists } c \in C \text{ such that } [\![\phi_{v\leftarrow c}]\!]^{\mathcal{I}} = True
\end{aligned}$$

The interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies a formula ϕ if and only if (iff) $[\![\phi]\!]^{\mathcal{I}} = True$, in any other case $[\![\phi]\!]^{\mathcal{I}} = False$. We also say that the interpretation is a *model* of ϕ. For a formula with a quantified variable v, a constant $c \in C$ ranging over the domain of discourse substitutes the variable v, which is denoted by $\phi_{v\leftarrow c}$.

A *theory* is a set of formulae. An interpretation satisfies a theory Φ, iff the interpretation satisfies every $\phi \in \Phi$. Then, the interpretation is a model of Φ. A theory Φ' is a logical consequence of a theory Φ iff every model of Φ is also a model of Φ'. We say that Φ' is *entailed* by Φ, denoted by $\Phi \models \Phi'$.

3.1.2 Description Logics

Description Logics (DLs) are a family of knowledge representation languages. Their members vary in their expressivity to represent concepts of a domain of discourse. The performance of their reasoning capabilities depends on the language complexity and choosing an appropriate DL for a given task affects the performance of the developed system that uses the DL and the respective reasoning facilities.

Most description logics are decidable fragments of first-order logic. They provide a formal semantics that allows specifying a precise meaning of the modeled concepts. Ontological modeling in the Semantic Web has been based on the earlier results on DLs. KL-One is considered to be the first system that overcame the problem of semantic ambiguities in knowledge representation languages like semantic networks [BS85]. Extensive and self-contained overviews can be found in the literature, among others in [BCM+03, LMP07, Rud11].

We give an overview of the building blocks of DLs and present two concrete description logics and their semantics in the following. The first description logic that we introduce is called $\mathcal{ALC}$, which is a very basic DL that has been widely studied and applied. The second DL is called $\mathcal{SHOIN}(\mathbf{D})$, which is underpinning the OWL Web Ontology Language and was used for the description of services in previous work.

The capability to infer *implicit knowledge* from the existing knowledge that is explicitly modeled, e.g., in an ontology, distinguishes DLs from non-logic modeling languages like UML. A set of inference rules allows us to infer implicit knowledge about the entities (concepts and individuals) automatically. Standard DL reasoning tasks include instance checking, subsumption, and satisfiability [BN03]. Instance checking denotes the task of retrieving instances of a specified concept. The subsumption task determines whether one description is more general than another one, which is a subconcept of the former. Satisfiability checks whether a description is free of contradictions. It is non-contradictory iff there exists a model of the description.

The analysis of worst-case complexity revealed that the subsumption problem of languages with a very low expressivity is already *intractable*, which means that it cannot be computed in polynomial time [Neb88].

Building Blocks

In analogy to the constants, monadic and 2-place predicate symbols of first-order logic, a description logic considers three basic syntactic building blocks: Individuals, concepts, and roles. *Individual* names are symbols that represent individuals of the domain. A DL *concept* represents a set of individuals and the *roles* represent binary relationships between individuals.

A DL ontology represents a state of the domain that is described by a set of ontology *axioms* (i.e., DL statements) that are true in this state. In description logics, we distinguish different types of axioms: terminological axioms (*TBox*) and assertional axioms (*ABox*). A *knowledge base* (*KB*) contains the terminology of the TBox and the assertions about individuals of the ABox.

Interpretation. In addition to the syntactical specification, a well-defined semantics is part of any formal logics. Based on a model-theoretic semantics of DLs, which assigns the models of a domain to a syntactical expression, sound and complete reasoning algorithms have been developed. The key feature of description logics is that they aim at retaining the decidability of reasoning tasks.

The *interpretation* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of DLs comprises a domain $\Delta^{\mathcal{I}}$ and the interpretation function $\cdot^{\mathcal{I}}$. Similar to FOL, the interpretation function maps individuals, concepts, and roles to the elements of the domain, subsets of the domain individuals, and binary relationships between the individuals, respectively.

Hence, the interpretation of a knowledge base assigns truth values to the axioms. $\mathcal{I}$ satisfies a set *KB* of axioms iff $\mathcal{I}$ satisfies each axiom of *KB*. If $\mathcal{I}$ satisfies an axiom (respectively a *KB*), then $\mathcal{I}$ is a *model* of this axiom (resp. *KB*). Axioms can be considered as constraints on the interpretations. A knowledge base *KB* *entails* an axiom α, iff the axiom α is true in any model of the knowledge base.

Basic Description Logic $\mathcal{ALC}$

The Attributive Language with Complements $\mathcal{ALC}$ is a description logic with a rather low expressivity and a basic set of language constructs. We summarize the specification of the $\mathcal{ALC}$ description logic as it was introduced in [SSS91]. Before we present the terminological and assertional axioms that can be expressed in $\mathcal{ALC}$, we list the constructors of concept descriptions in Table 3.1.

The basic language constructs are atomic concepts and atomic roles. Concept constructors allow for the definition of complex concept and role descriptions inductively.

Table 3.1: $\mathcal{ALC}$ language constructs of concept descriptions

$\mathcal{ALC}$ Concept Constructor	Syntax	Semantics
Universal Concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom Concept	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Complement	$\neg C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
Universal Restriction	$\forall R.C$	$\{c \mid \forall(c,d) \in R^{\mathcal{I}} \Rightarrow d \in C^{\mathcal{I}}\}$
Existential Restriction	$\exists R.C$	$\{c \mid \exists(c,d) \in R^{\mathcal{I}} \wedge d \in C^{\mathcal{I}}\}$

We adhere to the common notation as used in [BCM$^+$03], where capital letters A, B denote atomic concepts, C, D denote complex descriptions of concepts, and R, S denote atomic roles. $\mathcal{ALC}$ comprises the universal concept ($\top$), bottom concept ($\bot$), atomic negation ($\neg$), intersection ($\sqcap$), value restrictions ($\forall R.C$), and existential quantification ($\exists R.C$). In more expressive DLs, like $\mathcal{SHOIN}(\mathbf{D})$ that we introduce below, further constructors can be used to create complex descriptions.

The given semantics is based on the interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The domain $\Delta^{\mathcal{I}}$ is not empty and the interpretation function assigns to every concept symbol C a subset $C^{\mathcal{I}}$ of the domain and assigns every role symbol R to a subset $R^{\mathcal{I}}$ of pairs of individuals of the domain ($R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). Universal and bottom concepts are interpreted as $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$, respectively. The language constructors of $\mathcal{ALC}$ are summarized with their semantics in Table 3.1.

Terminological axioms constitute the TBox. In $\mathcal{ALC}$, concept and role equivalences as well as concept and role subsumption can be expressed in the TBox. We explain their informal and formal semantics by means of an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ in the following.

A concept equivalence axiom $C \equiv D$ defines the concept C. This axiom means that each individual of the domain either belongs to both sets $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ or to none of them. A role equivalence axiom $R \equiv S$ defines the role R. This axiom means that every pair of individuals of the domain either belong to both sets $R^{\mathcal{I}}$ and $S^{\mathcal{I}}$ or to none of them. A concept subsumption axiom $C \sqsubseteq D$ describes the inclusion. If both C and D are atomic concepts, then this axiom defines a concept hierarchy. This axiom means that if an individual $c^{\mathcal{I}}$ of the domain belongs the set $C^{\mathcal{I}}$, then it belongs to $D^{\mathcal{I}}$. A role subsumption axiom $R \sqsubseteq S$ states that every pair of domain individuals that belongs to $R^{\mathcal{I}}$ also belongs to the set $S^{\mathcal{I}}$ of individual pairs. Role subsumption axioms can define a role hierarchy.

Individuals are instances of the concepts and can be defined by assertions in the ABox. $\mathcal{ALC}$ ABox axioms comprise concept assertion, role filter, individual equivalence, and individual inequivalence. We summarize their syntax and semantics in Table 3.2.

Concept assertions express that an individual is member of a concept. Role assertions specify a role relationship between two individuals. Individual equivalence asserts that two individuals are actually the same entity, while inequivalence expresses that two individuals are not equal, i.e., they do not represent the same entity.

Concrete domains. So far, we have not covered the representation of data objects in description logics. Instead of a logic axiomatization of data types, *concrete domains* can

Table 3.2: TBox and ABox axioms of the $\mathcal{ALC}$ description logic

$\mathcal{ALC}$ Axiom	Syntax	Semantics
Concept Equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Role Equivalence	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
Concept Subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Concept Assertion	$C(c)$	$c^{\mathcal{I}} \in C^{\mathcal{I}}$
Role Assertion	$R(c, d)$	$(c^{\mathcal{I}}, d^{\mathcal{I}}) \in R^{\mathcal{I}}$
Individual Equivalence	$c \equiv d$	$c^{\mathcal{I}} = d^{\mathcal{I}}$
Individual Inequivalence	$c \neq d$	$c^{\mathcal{I}} \neq d^{\mathcal{I}}$

be integrated in description logics in order to constrain data objects of data types that represent numbers or strings. Properties, e.g., operators for counting or expressing equality and inequality, can be defined and included in concrete domains. This feature of concrete domains is useful for the development of many applications.

$\mathcal{ALC}(\mathbf{D})$ symbolizes the logic that integrates concrete domains $\mathbf{D}$ into $\mathcal{ALC}$. For a general overview on the integration of concrete domains to description logics we refer to [BH91, HLM99].

Concrete domains and its application in the logic $\mathcal{ALC}(\mathbf{D})$ are presented in [BH91] in detail. The basic idea of the authors is to introduce n-ary predicates to the language. The predicate arguments can be data objects of a concrete domain. Syntax and semantics of expressions that relate to and constrain data objects were defined formally for $\mathcal{ALC}(\mathbf{D})$. It allows to, e.g., express age restrictions. The example of an $\mathcal{ALC}(\mathbf{D})$ concept description shown below, where $\mathbf{D}$ represents integers, was presented in [BH91]. It describes that women are individuals, which are member of concepts Human and Female and also either an individual of the concept Mother, or their feature age has a value of at least 21.

$$\mathsf{Woman} \equiv \mathsf{Human} \sqcap \mathsf{Female} \sqcap \big(\mathsf{Mother} \sqcup \geq_{21} (\mathsf{age})\big)$$

Expressive Description Logic $\mathcal{SHOIN}(\mathbf{D})$

We now turn our attention to a more expressive description logic $\mathcal{SHOIN}(\mathbf{D})$ that is the formal underpinning of the Web Ontology Language OWL-DL. Furthermore, it has been used in previous works to describes services, e.g. in [Aga07a], which serves as the base of our work. In Table 3.3, the (additional) constructors for concept and role descriptions, which were not already presented in Table 3.1, are given. $\mathcal{SHOIN}(\mathbf{D})$ extends the $\mathcal{ALC}$ DL. That is, the constructors of $\mathcal{ALC}$ are also valid constructors of $\mathcal{SHOIN}(\mathbf{D})$ descriptions.

The name $\mathcal{SHOIN}(\mathbf{D})$ of the logic already suggests its expressivity: It inherits the expressivity of $\mathcal{ALC}$ and adds the support for role transitivity, role hierarchies, nominals (individuals in enumerations), inverse roles, cardinality restrictions, and data types [BCM+03]. Based on these language constructors, the set of axioms in $\mathcal{SHOIN}(\mathbf{D})$ contains: (i) $\mathcal{ALC}$ axioms (cf. Table 3.2), and (ii) role transitivity axioms $R_1 \circ \ldots \circ R_n \sqsubseteq S$ that hold if $R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$.

Table 3.3: $\mathcal{SHOIN}(\mathbf{D})$ concept and role constructs in addition to the ones in $\mathcal{ALC}$

Constructor	Syntax	Semantics
Qualified Number Restriction	$\geq nR$	$\{c \mid \#\{d \mid (c,d) \in R^{\mathcal{I}}\} \geq n\}$
	$\leq nR$	$\{c \mid \#\{d \mid (c,d) \in R^{\mathcal{I}}\} \leq n\}$
Enumeration	$\{c_1, \ldots, c_n\}$	$\{c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}}\}$
Role	R	$\{(c,d) \mid (c,d) \in R^{\mathcal{I}}\}$
Inverse Role	R^-	$\{(d,c) \mid (c,d) \in R^{\mathcal{I}}\}$
Role Transitivity	$Trans(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Example 2. *In Example 1, we described in first-order logic that every book is also a written work. For this purpose, we assume that the concepts* Book *and* WrittenWork *are specified. The TBox axiom* Book $\sqsubseteq$ WrittenWork *states that every individual of the concept* Book *is also an individual of the concept* WrittenWork.

The availability of books can be defined as the concept AvailBook*. We specify that individuals of the concept* AvailBook *are members of the concept* Book *and have to be stored at least at one location. Let* storedAt *be a role. Then we can define* AvailBook *as shown in the following.*

$$\mathsf{AvailBook} \sqsubseteq \mathsf{Book} \sqcap \geq 1 \mathsf{storedAt}.\top$$

3.1.3 Ontologies

Knowledge representation languages can be used to represent ontologies. Description logics are the latest evolution of formalisms that underpin ontologies. The well-defined semantics of description logics allow to unambiguously and logically represent domain models in form of ontologies. Ontology languages are typically based upon description logics and employ the open world assumption.

In our work, we will use the term *ontology* in its interpretation from the information science perspective. An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest [SBF98]. This definition merges the definitions from [Gru95] and [BAT97]. It means that ontologies are machine-interpretable representations of the objects, concepts, and relations of a specific domain. Ontologies feature clear (formal) semantics and provide automatic reasoning procedures. The aspect of sharing is an important aspect of ontologies, because it implies that different stakeholder agree on the conceptualization and understand it in the same manner [GOS09].

Ontology Languages

Ontologies constitute an important part of the Semantic Web. They are the basis for representing knowledge that is formalized and organized by description logics such that an intelligent and automated retrieval and combination of the information and knowledge in the Semantic Web is enabled.

The Web Ontology Language (OWL) is a W3C standard for the representation of ontologies [BvHH+04]. OWL is based on a family of description logics. Different OWL language profiles like OWL-Full, OWL-DL, and OWL-Lite exist. Each profile provides a different degree of expressivity that is provided by the appropriate description logic as the underlying formalism. An appropriate profile has to be selected for each system that needs to reason on ontologies. The less expressivity is required and offered by a profile, the less computational complexity is required for the reasoning.

OWL-DL is an ontology language that is based on the $\mathcal{SHOIN}(\mathbf{D})$ description logic. It features decidability and NExptime worst-case complexity [Don03, MW11]. If nominals are not included, Exptime worst-case complexity can be achieved. Several reasoning algorithms have been developed and implemented for this profile.

OWL-Full is even more expressive than OWL-DL. In fact, OWL-Full is the most expressive ontology language of the OWL family. Although OWL-Full is undecidable and thus not practical in many applications, it achieves interoperability with RDF [MM04] and RDF/S [BG04], which does not enforce a separation of classes, properties, individuals and data values.

OWL 2 [OWL09] is the current evolution of the OWL language definition with an overall structure that is similar to the first version (OWL or OWL 1 to delimit it from OWL 2). One of the differences is an increased expressivity. The expressivity of OWL 2 corresponds to the $\mathcal{SHROIQ}(\mathbf{D})$ description logic. In comparison to $\mathcal{SHOIN}(\mathbf{D})$, the $\mathcal{SHROIQ}(\mathbf{D})$ DL additionally allows expressing the negation of ABox roles, qualified cardinality restrictions, further role axioms and adds the self concept as well as a universal role. For the details on the individual language features of $\mathcal{SHROIQ}(\mathbf{D})$, we refer to [GHM+08].

Similarly, the Web Service Modeling Language WSML [dBFK+08] is a family of ontology languages with varying expressivity and is based on appropriate description logics. As the name suggests, the WSML ontology languages were originally developed for the description of Web services. Nevertheless, the WSML language family can be used independently of Web services, too.

Serialization Formats

The Semantic Web demands that the information and especially the formalized knowledge can be easily shared and exchanged between different stakeholders. The Resource Description Format (RDF) is a lightweight representation for data and knowledge on the Web [Pan09]. RDF is a W3C recommendation [MM04] and recent ontology languages like OWL and WSML provide an RDF serialization.

RDF builds on a graph model to decode entities and their relationships. *Resources* can be or refer to anything that has a URI (Uniform Resource Identifier) as a unique identifier and has to be dereferenceable. Resources can be annotated by properties, which are associated to a value (e.g., a resource, data object, or literal). RDF represents the properties and relationships among resources in the form of directed graphs.

An RDF statement presents a fact in the form of `subject property object .`, see Listing 3.1. All statements follow the same logical schema. An RDF graph is formed by a set of statements. Different syntaxes of the RDF serialization exist, most notable RDF/XML, Turtle, and N3. In this work, we will use the Turtle syntax [BBL11], which is compact and provides good readability. In Turtle, statements are represented by ⟨http://example.org/subject⟩ ⟨http://example.org/predicate⟩ ⟨http://example.org/object⟩ ., where URIs representing the resources are written in brackets. The prefixes can be abbreviated as shown in the following Listing 3.1. Quoted literals can be followed by a data type URI that indicates how the lexical form can be mapped into the literal [MM04].

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema> .

ex:subject ex:predicate ex:object .
ex:subject ex:anotherPredicate "a string literal"^^xsd:string .
```

Listing 3.1: Basic structure of RDF statements

In order to annotate the same resource in the subject position with several properties, the following abbreviation can be used (see Listing 3.2).

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema> .

ex:subject ex:predicate ex:object ;
    ex:anotherPredicate "a string literal"^^xsd:string .
```

Listing 3.2: Example of RDF statements

Reification can be problematic in RDF when statements about statements (propositions) need to be included. One option is to encode the proposition as a string literal. However, the semantics of the string literal will be lost. The correct way to include the proposition in the RDF model is to add a resource representing the proposition, which is then referring to subject, predicate, and object of the proposition by the properties rdf:subject, rdf:property, and rdf:object.

For a more comprehensive introduction of the RDF language elements like lists, containers, blank nodes, and typed literals, we refer to the W3C specification [MM04]. For an overview on the RDF model theory, which provides a semantics to RDF statements as well as RDF/S ontologies, we refer to [Hay04, Pan09].

RDF/S. RDF allows expressing arbitrary statements about individuals and their properties. In order to specify the terminological knowledge about resources and their domain, the schema language for RDF was defined. The RDF Vocabulary Description Language, also called RDF Schema (RDF/S), is part of the W3C RDF recommendation and defines a data model and a vocabulary for expressing RDF statements. RDF/S itself is a specific RDF (meta) vocabulary that can be used to define a semantics of any other RDF vocabulary.

RDF/S allows for the definition of very lightweight ontologies. It allows for the definition of classes, properties, property restrictions, and hierarchies of classes and properties. The following example in Listing 3.3 summarizes the definition of the classes ex:Book, ex:Author, a property ex:hasAuthor and the domain and range restriction of the property, which restricts the classes of the resources in the property's subject and object positions.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .

ex:Book rdf:type rdfs:Class .
ex:Author rdf:type rdfs:Class ;
    rdfs:subClassOf ex:Person .
ex:hasAuthor rdf:type rdf:Property ;
    rdfs:domain ex:Book ;
    rdfs:range ex:Author .
```

Listing 3.3: Example of RDF/S statements

3.1.4 Terms and Notations

In this thesis we will use knowledge representation techniques and ontologies to represent resources of services within their descriptions. After providing an overview on a broad range of logics, languages, and their syntax in this section, we now summarize the terms and notations that we will use throughout the thesis.

Within the literature, the terms used to describe the constructs of description logics and ontologies have sometimes been different from the ones that are typically used for classic logics. For instance, concepts and classes represent the same (a set of individuals). As our work uses ontologies and reasoning algorithms for the underlying description logics, we will in general stick to the terms ontology classes, ontology properties, and ontology individuals. If the context allows us to, we omit the term ontology to refer to these concepts, too. We use sans serif fonts to distinguish ontology concepts from the remaining text.

Classes are denoted with an upper-case first letter. Properties and individuals start with a lower-case letter. In contrast to individuals, property names typically start with is or has.

We use the predicate notation to describe logical formulae and axioms when they are not part of an ontological description. As used in Section 3.1.1 for first-order logic expressions, we write for example ex:Book(ex:b) to express that the individual ex:b is member of the class ex:Book.

3.2 Services

The development and management of large software systems is challenging from many perspectives. A few decades ago, the development process was centered on an incremental (multi-phased) approach [Roy87], often known as the waterfall model. This model best suits centralized processes of single complex software systems in which the costs for integrating different components are not included [Lef07, Chapter 2].

More recently, agile software development frameworks and methods for managing the software projects and the development itself emerged [BBvB$^+$01]. Organization and management structures that aim at agility also enable the support of less tightly coupled and service-based software systems, which are more decoupled than traditional monolithic systems. In fact, the use of agile methods in combination with service orientation has been investigated in the context of software development [Deb04, KLS05, SAM12] and also in the context of organizations [OBS06].

Services are software components that provide a defined feature. They can be developed independently from the remaining components. With interface definitions that are rarely altered, services can be integrated and used without knowledge about the underlying implementation or technology. In an agile development process, the implementation and the technologies used for the implementation can rapidly change without affecting the remaining system or other development teams if the interface remains unchanged.

An overall software system that integrates services is called a *service-based system.* It can be developed in an agile and flexible manner, as it evolves while the features can be easily customized by adding, replacing, and removing services to/from the system. The advantages of service-based software development methods also apply to the development of Web information systems. In fact, they can be considered as requirements for a cost-effective development of Web-based applications and systems. The flexibility provided by a loose coupling allows adapting systems to the ever changing environment of the Web. The distribution of services that can be developed, provided, and executed by different stakeholders is consistent with the distributed setting of the Web of pages (HTML documents).

In order to cope with the challenges that arise with the distribution on Web scale, automated methods are required to help developers, systems, and end users in using the potential that is offered by the services on the Web. The description of the features that services offer and an effective service search are fundamental to successfully employ service orientation on a large scale.

In the following, we introduce and define several terms that are used in this thesis. In Section 3.3, we introduce the foundations of semantic service modeling and the description of the service behavior. For this purpose, we give an overview of the π-calculus and μ-calculus.

Terms and Definitions

Various definitions of a service exist in literature. We restrict ourselves to an explanation of our view on services within the scope of the thesis. Araujo and Spring discussed a more general view on services including economic aspects in [AS06]. It is based on Hill's service definition [Hil77], which is not contradictory to our view on concrete service instances. The following view on services is based on a collective effort of the Karlsruhe Service Research Institute to define services from a multitude of perspectives and disciplines. We present the excerpt of the outcomes that is relevant for the current work.

Definition 1 (Service)**.** *A service is a software program that provides a defined and pre-implemented functionality through a specified interface. It serves as the basis for machine-to-machine interoperability and communication.*

A service is offered by means of an unambiguous, formal, and explicit description of the provided functionality including the information about the use of the interface and the state changes caused by the execution.

We call an actor who instantiates the server role, a *service provider.* A *service consumer* is another actor who invokes a service by sending a request signal. With the above interpretation of a service we intend to state that a service performs or triggers an action upon *service invocation*, i.e., the receipt of a request. The action changes the *state*, i.e., the condition, of the service provider, which can include some objects that were provided by the consumer with the invocation. State changes involve changes in a database, knowledge base, or in knowledge that is represented in some other way.

The service description allows advertising services. Agents are able to interpret the offer and, thus, there is an implicit agreement between providing and consuming agents prior to performing the service.

Our notion of a service includes (i) ICT-based services (e.g., Web and Cloud Services), (ii) ICT-mediated services, (e.g., Amazon's Mechanical Turk), and (iii) regular business services (e.g., product support) . People-to-people services (e.g., pickup and logistics services) can fall in the second category of ICT-mediated services if the people-to-people services are described and triggered by the help of ICT. As the term Web services has been widely used in the literature, we clarify our interpretation of Web services as follows:

Definition 2 (Web Service, based on [LP06])**.** *A Web service is an ICT-based service that can be requested via standard Internet protocols. It provides (remote) procedures that can be invoked with a machine-readable and standardized request syntax. The description of Web services includes at least input and output messages.*

Orthogonal to the previous discussion, layers of abstractions need to be considered, too. The *general service* is a service for which the provider has not been instantiated. This service may be defined at various levels of abstraction, depending upon purpose. The *particular service* is a service that the provider instantiated. The *service instance* is a completely specified service with both the provider and consumer specified. The *service execution* of a service instance should cause some change of state to occur according to the service agreement.

A service can offer several operations, where each operation provides a functionality that can be independent of the other operations. The operations are often related in a sense that they provide complementary functionalities, but they can be used independent of each other. In this thesis, we do not distinguish between services and operations. Without loss of generality, we assume each service provides one operation and refer to it as a service.

3.3 Semantic Service Modeling

After we have defined our view on services, we will now focus on the description of service offers. We only consider recent approaches for machine-readable and machine-interpretable, i.e., formal, description approaches in our overview. As we will develop a service discovery approach that operates on a logic and abstract level based on descriptions, we do not introduce protocols (like HTTP), standard representations of exchanged messages (like SOAP [GHM$^+$07]), or syntactical interface description languages (like WSDL [CCMW01]) that merely focus on implementation issues.

3.3.1 Service Description Frameworks

With the advent of Semantic Web Services, the Semantic Web community with their focus on languages developed service description approaches that use DL, as for example in [GCTB01, SPAS03, LH03, BHL$^+$05]. While static objects like input and output parameters (we say *resources*) of a service can be properly described in DL, the dynamic aspects of the service functionality cannot be expressed solely using DLs with static signatures. Hence, state-based service modeling approaches emerged more recently. Assuming the discrete time paradigm, each state represents a snapshot of the world and the differences between two consecutive states allow drawing conclusions about the service functionality that caused this state change.

Semantic Markup for Web Services (OWL-S) [MBH+04] and the Web Service Modeling Ontology (WSMO) [RKL+05] are two recent and most prominent frameworks for semantic description of services. The commonality of both specifications is that they provide an upper ontology that provides the basic vocabulary necessary to declare and describe concrete services. Although there are some conceptual differences between OWL-S and WSMO [LRPF04], both of them provide means to describe service functionalities by OWL-S Service Profile and WSMO Capability respectively. The *functionality* of a service describes what the service does or provides while abstracting from the internal service implementation and external communications. Both specifications provide further means to describe the *behavior* that reveals internal details of the service implementation and external communications. The respective elements of the specifications are the OWL-S Service Model and the WSMO Interface (which further distinguishes Orchestration and Choreography elements). The service behavior is often described by a process expression to specify how a service is provided and how a consumer can interact with the service.

OWL-S is an upper ontology to describe Web services semantically and replaces DAML-S. The OWL-S ontology is expressed in the Web Ontology Language (OWL) and consists of the following three parts [SPAS03]. The *Service Profile* describes what a service provides to prospective clients. The *Process Model* (also known as Service Model) describes how a service works and can be used. Possible interactions with a service are described by the *Service Grounding.* In order to describe each part of an OWL-S service description, OWL-S provides further ontology classes and properties. For instance, classes like Input, Output, Condition, Result and properties like hasInput, hasOutput, hasCondition, hasResult are used for the description of the functionality as part of the Service Profile. The OWL-S specification in [MBH+04] contains a book selling service example that shows how OWL-S can be used to create actual service descriptions. We do not want to repeat the extensive descriptions of such an example, but recommend looking at the original description in [MBH+04].

WSMO is an alternative approach to OWL-S. Driven by the same goals, both approaches share commonalities but also differ in some aspects like the support for mediation and language layering, which are addressed by WSMO only [LRPF04]. The capability element of WSMO enables the description of the Web service functionality by assumptions, preconditions, postconditions, and effects. It provides a clear separation of the information space including inputs and outputs (described by preconditions and postconditions) and the state space (described by assumptions and effects). Analogously to OWL-S, the upper model is specified in a WSML ontology that contains respective classes and properties to create structured and semantic Web service descriptions. We also recommend the example WSMO description of a train ticket booking service, which has been presented in [FFST11, Chapters 7.4 and 8.4].

In contrast to OWL-S, WSMO specifies Goals that capture the objectives of a user, who for instance wants to find Web services. Goals are result-oriented as they consist of postconditions and effects, only. The assumption for omitting preconditions and assumptions is that users are only interested in the outcomes of a service. However, as the semantic description of Web services should enable automation in manifold use cases, one has to consider scenarios like service composition in which objectives with constraints over preconditions and assumptions play a similarly important role. Especially when automated methods verify requirements over service offers in order to compose them, many criteria of desired services must be specified in a request in order to guarantee their applicability as well as the usability, liveness, and safety of the composition. Otherwise, human effort will still be required.

Many different service description approaches besides OWL-S and WSMO have been proposed. WSDL-S provides semantic annotation of the elements of technical service interface description in WSDL documents [AFM$^+$05]. Additional elements and attributes (such as wssem:modelReference) extend the Web Service Description Language and allow to link to a formal description of inputs, outputs, preconditions, effects, and a concept from a service classification schema. Semantic Annotations for WSDL and XML Schema (SAWSDL) [KVBF07] extends WSDL-S and provides annotations for the WSDL language in version 2.0. The language used to represent semantic models of the annotated service elements is not dictated and can be chosen independently. For example, the WSMO-Lite ontology that mainly provides the concepts for expressing the service functionality can be used in combination with SAWSDL [VKVF08].

3.3.2 Service Behavior Modeling

The behavior of a service is a functional property that describes how a service provides the offered functionality (*orchestration*) and how to interact with a service in order to access it successfully (*choreography*). In the study of computation, state-changing machines are used to model the behavior of systems formally. Formalisms like abstract state machines [Bör98, BGG$^+$06], Petri nets [MPPP02, DLC$^+$07], and π-calculus [BWR09, Aga07a] have been applied for this purpose.

An own process language for OWL-S has been proposed based on the findings and best practices from planning languages like PDDL (Planning Domain Definition Language [MGH$^+$98]), process and workflow modeling with PSL (Process Specification Language [SGT$^+$00]), and the description of distributed systems (π-calculus [Mil99]) among others. However, besides a specification of the language constructs, a formal semantics of the OWL-S process language is not part of the W3C Specification [MBH$^+$04]. Abstract state machines [Gur94, Bör98] "inspired" a framework for WSMO Choreography and Orchestration descriptions [RKL$^+$05]. However, such a framework including a formal semantics never became part of the WSMO specifications.

In the following, we define a state-based model of service computations, before we introduce the π-calculus and μ-calculus. Both logics can describe state-based systems that model the service behavior in our work.

State Transition Systems

A state transition system is a theoretical model of a software system that abstracts from the concrete processing performed on a computer. Labeled transition systems can be used to model the behavior of a particular service. A service invocation can trigger a set of actions that change the state of the service provider with respect to the information within the premises of the provider.

Definition 3 (Labeled Transition System)**.** *For a set of atomic propositions P and a set of actions A, a labeled transition system (LTS) is a tuple $(S, \rightarrow, A, \lambda)$, where S is a finite set of states, $\rightarrow \subseteq S \times A \times S$ a set of labeled transitions between the states, and $\lambda : S \rightarrow 2^P$ a labeling function that maps each state $s \in S$ to the set of atomic propositions that are true in s.*

Atomic propositions are logical formulae, which describe the provider knowledge in a state. Every state $s \in S$ of the LTS is described by a set of propositions that are true in this state. $KB(s)$ denotes the state knowledge $\lambda(s)$, such that $KB(s) \models \lambda(s)$.

Transitions are defined by the ternary relation $\rightarrow$. A transition $(s_i, a, s_j) \in \rightarrow$ between the two states $s_i \in S$ and $s_j \in S$ is labeled with a description of the action $a \in A$. s_i is called the source and s_j the target of a transition. We write $s_i \rightarrow^a s_j$ if $(s_i, a, s_j) \in \rightarrow$. The structure of the labels in A depends on the action type. For example, a channel, the involved actors, and the exchanged parameters are the main characteristics of a communication action, while a local (i.e., computational) action is modeled differently.

A transition system that models the behavior of a service will always have a unique state that not the target state of any transition. We refer to this state as the *start state* of the system, even though the definition of an LTS does not make this distinction.

The π-calculus Process Algebra

Process algebras can be used to describe the observable behavior of services by providing constructs for modeling data and control flow. Here, concrete actions with concrete parameter values occurring in service instances are aggregated into variables.

The π-calculus is a process calculus and was developed in 1992 by Robin Milner, Joachim Parrow, and David Walker [MPW92a, MPW92b, Mil99]. As Milner stated in [Mil99], "communication is a fundamental and integral part of computing, whether between different computers on a network, or between components within a single computer."

The π-calculus evolved from the calculus of communicating systems (CCS) [Mil80] and can be distinguished by its ability to handle mobility. In π-calculus, the names of communication channels can be communicated along the channels themselves. Moving information inside of a computer program is treated equally to the exchange of messages and computer programs across a network. Concurrent computations within networks that can reconfigure themselves can be described as well.

This calculus provides a simple but powerful language to describe labeled transition systems that represent the observable behavior of communicating systems. The calculus is Turing complete; the lambda calculus can be encoded in the π-calculus [Mil92]. Expressions of the language are named *process expressions*, which are used to describe the service behavior. We use the polyadic π-calculus [Mil99, Chapter 9], which is an extension that allows communicating more than one parameter in a single action.

π-calculus syntax. The set of valid process expressions of the polyadic version can be defined as follows.

Definition 4. *Structure of Process Expressions*

$$\begin{aligned}\pi \quad ::= \quad & \mathbf{0} \mid c[x_1, \ldots, x_n].\tau \mid c\langle y_1, \ldots, y_n\rangle.\tau \mid \\ & \gamma.\tau \mid \tau_1 \parallel \tau_2 \mid [\omega_1]\tau_1 + [\omega_2]\tau_2 \mid @A\{y_1, \ldots, y_n\}\end{aligned}$$

The *null process* **0** denotes a process that does nothing. It is used as termination symbol in a process expression.

The *input process* $c[x_1, \ldots, x_n].\tau$ is a process that takes inputs at a port with the name c, which is a communication channel, and binds them to the variables $x_1, \ldots, x_n$. The subsequent behavior of this process is defined in the process expression τ.

Analogously, the *output process* $c\langle y_1, \ldots, y_n\rangle.\tau$ denotes a process that outputs the values $y_1, \ldots, y_n$ at port c and then behaves as defined in τ.

The *local process* $\gamma.\tau$, as used in [Aga07a], performs the action γ and then behaves as defined in τ. In π-calculus, observable, non-observable, internal, and complementary actions can be modeled. Local actions perform changes in the knowledge base of the system, e.g., state resources can be changed, added, or deleted.

The *composition* $\tau_1 \parallel \tau_2$ consists of processes τ_1 and τ_2 acting in parallel.

The *summation* $[\omega_1]\tau_1 + [\omega_2]\tau_2$ denotes a choice of one of the alternatives τ_1 or τ_2 guarded by conditions ω_1 and ω_2 respectively. Conditions are Boolean queries that can be evaluated based on the knowledge $\lambda(s)$ of the current state s.

We write $\omega?\tau_1\colon\tau_2$ for a process $[\omega]\tau_1 + [\neg\omega]\tau_2$ that models an if-then-else construct. It is a *deterministic choice* process, whose behavior is determined by the condition ω. If ω is true, then its subsequent behavior is defined by the expression τ_1, otherwise, it will behave as defined in τ_2.

$@A\{y_1, \ldots, y_n\}$ denotes the invocation of an agent identifier A. With the invocation of an external process that is identified by the agent, the resources $y_1, \ldots, y_n$ of the current process are passed to the agent.

A named process expression, as syntactically defined above, is denoted by an *Agent Identifier*. $A(x_1, \ldots, x_n) \overset{\text{def}}{=} \pi$ is the definition of the agent identifier A with *invocation parameters* $x_1, \ldots, x_n$ that are passed to the agent along with its invocation.

The operational semantics of π-calculus is defined inductively [MPW92b]. It maps a process expression to a labeled transition system. Input, output, and local actions are translated into state transitions of the LTS.

In the following, we summarize the results of the Ph.D. thesis from Agarwal, who combined the π-calculus with DL in order to describe resources semantically and applied this formalism for the service behavior description [Aga07a].

Semantic process expressions. In pure π-calculus, the process resources and variables are represented by names. Names are strings without any structure. As a result, it is hard for end users to understand which values they should provide for the variables in order to get the desired result. Also, it is not possible to automatically let machines interpret and reason over such process expressions.

In order to overcome this limitation and enable automated reasoning techniques, the process resources and variables are annotated with domain ontology concepts. We assume that a domain ontology $O_{\mathcal{D}_\omega}$ is attached or referenced by each service description $\mathcal{D}_\omega$ and that $O_{\mathcal{D}_\omega}$ comprises the ontological concepts that are used to describe the process resources. That is, $O_{\mathcal{D}_\omega}$ supplies the static TBox knowledge that can be applied throughout the entire process execution.

By combining the π-calculus process description language with description logics, one cannot only describe the types of process resources and variables but also how they relate with each other. For example, if a process input action receives two parameters of type Person, one can also describe that both persons are authors and both have co-authored the same book. Furthermore, it is useful to have information on the type of communication protocol and messages transmitted over a channel. E.g., a book selling process sends the book via "HTTP" as PDF file or via "surface mail" as hard copy.

Example 3. *Let* $\pi \overset{\mathsf{def}}{=} \mathsf{httpCh}[\mathsf{a}_1, \mathsf{a}_2].\tau$ *be the input process that accepts two input parameters* a_1 *and* a_2 *over the communication channel* httpCh*.*

The ontology $O_{\mathcal{D}_\omega}$ *defines classes* ex:Person*,* ex:Author*,* ex:Book *and a property* ex:hasAuthor*. Then, the following logical formulae are attached as a label to the input action* $\mathsf{httpCh}[\mathsf{a}_1, \mathsf{a}_2]$*.*

$$\exists b.\mathsf{ex{:}Book}(\mathsf{b}) \land \mathsf{ex{:}hasAuthor}(\mathsf{b}, \mathsf{a}_1) \land \mathsf{ex{:}hasAuthor}(\mathsf{b}, \mathsf{a}_2)$$
$$\land\, \mathsf{ex{:}Person}(\mathsf{a}_1) \land \mathsf{ex{:}Author}(\mathsf{a}_1) \land \mathsf{ex{:}Person}(\mathsf{a}_2) \land \mathsf{ex{:}Author}(\mathsf{a}_2)$$

The service itself is the recipient of the service invocation, whereas the invoking client of the communication is not further specified.

$$\mathsf{ex{:}HTTP}(\mathsf{httpCh}) \land \mathsf{ex{:}hasRecipient}(\mathsf{ex{:}httpCh}, \mathsf{ex{:}service})$$

In the following, we will directly add types of input and output parameter to the description of the actions. E.g., we write $\mathsf{httpCh}[\mathsf{a}_1 : \mathsf{ex{:}Person}, \mathsf{a}_2 : \mathsf{ex{:}Person}]$ to indicate the input parameter types in the above example.

In general, the invocation, input, and output parameters used in the process expressions are semantically described analogously. These descriptions are attached to the individual process language constructs. Note that all the descriptions of the resources cannot be modeled in a single ontology, as their description may change during the execution, which could lead to logic inconsistencies.

The local actions are semantically annotated with the knowledge that is affected by these changes. The changes Δ of a local action are a set of ontology axioms (class membership, object property, data property, individual equivalence, and individual inequivalence axioms are allowed). Every axiom is associated with a '+' or '$-$' in order to express whether this change has added or removed an axiom from the provider's knowledge base.

The unique name assumption for an entire process expression is required in both formalisms. More details about this formalism and its semantics may be found in [Aga07a, ARA08, ALS09].

Temporal and Modal Properties of Processes

Modal and temporal logics describe processes declaratively. A declarative process description specifies modal and temporal properties of a process.[1] These process property descriptions can be considered as constraints over processes. Hence, temporal logics can and have been used to express discovery requests.

[1]The μ-calculus allows specifying modal and temporal properties of processes. In our work, we use the μ-calculus-based language to specify properties of the service behavior. Though we will use the term *behavior constraint* in order to distinguish it from the properties in service descriptions.

The modal μ-calculus is used to express modal and temporal properties of labeled transition systems. It extends the propositional modal logic with least fixpoint μ and greatest fixpoint ν operators. Like π-calculus, it features a simple language structure, provides formal semantics, and provides immense power due to the fixpoint operators [BS06]. In the following, we summarize the syntax and the semantics of the μ-calculus. Extensive introductions, examples, and further research can be found in the literature [Koz83, Sti01, BS01, BS06].

Syntax. For the purpose of defining the syntax of μ-calculus expressions, we let Ψ denote temporal and modal properties of an LTS. In the context of service discovery, we often refer to the properties as constraints.

Definition 5. *Basic μ-calculus Syntax*

$$\Psi ::= \Psi \wedge_\mu \Psi \mid \neg_\mu \Psi \mid \mu X.\Psi(X) \mid \langle a \rangle \Psi \mid P \mid \mathbf{true} \mid \mathbf{false}$$

Properties can be combined by the conjunction operator (symbolized $\wedge_\mu$ in contrast to the Boolean operators). The negation ($\neg_\mu$) allows excluding properties. Terminals of μ-calculus expressions are propositions P as well as **true** and **false**, which match all or no processes, respectively. The existence of an action a with the constraint that in the subsequent process after the action a the property Ψ holds is expressed by $\langle a \rangle \Psi$.

Modal equations of the form $X \overset{\text{def}}{=} \Psi$ state that X expresses the same property as the formula Ψ and allow for the description of perpetual processes. For example, $X \overset{\text{def}}{=} \langle \mathit{tick} \rangle \mathbf{true}$ means that only processes that immediately execute the action *tick* have the property X. The recursive modal equations $X \overset{\text{def}}{=} \langle \mathit{tick} \rangle X$ describes a clock process that perpetually executes the *tick* action. This recursive modal equation has extremal solutions that are least and greatest fixed points of the monotonic function $f[\langle \mathit{tick} \rangle X, X]$, which represents the evaluation of the property over an LTS [Sti01]. We will give further insides on the evaluation of μ-calculus formulae in the end of this section. For an extensive overview on modal equations and fixed points of the monotonic functions, we refer to [Sti01, Chapters 4.5 and 5]. Syntactically, the least fixpoint is denoted by $\mu X.\Psi(X)$ and represents the minimal solution for a constraint Ψ.

Given the basic syntax, further constructs can be added to increase readability without changing the language expressivity. We summarize them in the extended syntax definition.

Definition 6. *Extended μ-calculus Syntax*

$$\begin{aligned} \Psi \quad ::= \quad & \Psi \wedge_\mu \Psi \mid \neg_\mu \Psi \mid \mu X.\Psi(X) \mid \langle a \rangle \Psi \mid P \mid \mathbf{true} \mid \mathbf{false} \mid \\ & \Psi \vee_\mu \Psi \mid [a]\Psi \mid \nu X.\Psi(X) \mid \Psi \ \mathbf{until}\ \Psi \mid \mathbf{eventually}\ \Psi \mid \mathbf{always}\ \Psi \end{aligned}$$

As we can see from the extended μ-calculus syntax, alternatives of properties can be expressed by $\vee_\mu$, too. The universal quantification of an action a is expressed by $[a]\Psi$. The greatest fixpoint $\nu X.\Psi(X)$ represents the maximal solution for a temporal constraint Ψ. E.g., $\nu X.\langle \mathit{tick} \rangle X$ expresses with the help of the greatest fixpoint operator that a clock is always ticking.

Both fixpoint operators allow us to define properties based on recursion. While the specification and readability of formulae containing fixpoint operators can be difficult for many users, yet they allow us to define the usual temporal operators. They are introduced as macros to the extended syntax as they are often used and can be helpful to conceal the syntactical complexity.

always Ψ means that Ψ holds on every path. With the help of the greatest fixpoint operator, it is defined by $\nu X.\Psi \wedge_\mu [-]X$. The property X means that Ψ holds in the current state and X holds recursively in any state succeeding the current state. We consider any state transition as no constraints ($[-]$) over the actions are specified.

For example, if a user always has to be authorized in order to interact with a system, then a proposition P_{auth} that asserts the user authorization holds in every state of the LTS describing the system. This example system satisfies $\nu X.P_{auth} \wedge_\mu [-]X$ since P_{auth} is immediately satisfied and in every successor state the property X holds again. The recursive formula asserts that the proposition P_{auth} holds in every state of a perpetual process. The state transitions can be arbitrary (denoted by $[-]$).

Ψ_1 **until** Ψ_2 means that the process will reach some state in which Ψ_2 holds, and Ψ_1 holds until this state is reached. A process that must reach a state in which Ψ holds is expressed by **eventually** Ψ.

Denotational semantics. The semantics of μ-calculus expressions is defined on a labeled transition system. Given an LTS $L = (S, \rightarrow, A, \lambda)$ as defined in Definition 3, the semantics maps μ-calculus formulae to sets of states (i.e., elements of 2^S). In the following, we summarize the semantics of the modal μ-calculus as specified in [BS01].

$$\begin{aligned}
[\![\mathbf{true}]\!]_\mathcal{V} &= S \\
[\![\mathbf{false}]\!]_\mathcal{V} &= \emptyset \\
[\![P]\!]_\mathcal{V} &= \mathcal{V}_{Prop}(P) \\
[\![X]\!]_\mathcal{V} &= \mathcal{V}(X) \\
[\![\Psi_1 \wedge_\mu \Psi_2]\!]_\mathcal{V} &= [\![\Psi_1]\!]_\mathcal{V} \cap [\![\Psi_2]\!]_\mathcal{V} \\
[\![\Psi_1 \vee_\mu \Psi_2]\!]_\mathcal{V} &= [\![\Psi_1]\!]_\mathcal{V} \cup [\![\Psi_2]\!]_\mathcal{V} \\
[\![\neg_\mu \Psi]\!]_\mathcal{V} &= S - [\![\Psi]\!]_\mathcal{V} \\
[\![\langle a \rangle \Psi]\!]_\mathcal{V} &= \{s \in S \mid \exists s.s \rightarrow^a t \wedge t \in [\![\Psi]\!]_\mathcal{V}\} \\
[\![[a]\Psi]\!]_\mathcal{V} &= \{s \in S \mid \forall t.s \rightarrow^a t \Rightarrow t \in [\![\Psi]\!]_\mathcal{V}\} \\
[\![\mu X.\Psi(X)]\!]_\mathcal{V} &= \bigcap \{\hat{S} \subseteq S \mid \hat{S} \supseteq [\![\Psi]\!]_{\mathcal{V}[X:=\hat{S}]}\} \\
[\![\nu X.\Psi(X)]\!]_\mathcal{V} &= \bigcup \{\hat{S} \subseteq S \mid \hat{S} \subseteq [\![\Psi]\!]_{\mathcal{V}[X:=\hat{S}]}\}
\end{aligned}$$

$\mathcal{V}_{Prop} : Prop \rightarrow 2^S$ is the valuation function of atomic propositions that assigns a set of states (in which the proposition is true) to each proposition. $\mathcal{V}_{Prop}$ is called the interpretation of the propositions.

The valuation function $\mathcal{V}[X := \hat{S}]$ maps the property X to the set $\hat{S}$ of states that have the property X and allows for the evaluation of recursive modal equations, i.e., to determine the respective fixed points. Otherwise, this valuation function agrees with the interpretation $\mathcal{V} : Var \rightarrow 2^S$ of variables [BS01].

true holds in every state of an LTS and thus $[\![\mathbf{true}]\!]_\mathcal{V} = S$. **false** does not hold in any state. The semantics $[\![P]\!]_\mathcal{V}$ of a proposition P is defined by the set of states $\mathcal{V}_{Prop}(P) \subseteq S$ in which the proposition P holds. The conjunction (disjunction) of individual constraints Ψ_1 and Ψ_2 is the set of states in which both (either of) the constraint is satisfied. The meaning of the negation $\neg_\mu\Psi$ of a constraint Ψ is the complement of the set of states in which the constraint Ψ holds.

Variables X express properties and occurs free or bound. We say that variable X is bound if it is in the scope of a fixpoint operator. Otherwise, say that the variable is free and the valuation function $\mathcal{V}$ determines its meaning. The formula $\langle a\rangle\Psi$ expresses that there has to exist an a-transition (from a state s to another state t) and the constraint Ψ is satisfied in the target state t. The meaning of the formula $\langle a\rangle\Psi$ is the set of states, which are the origin of such an a-transition with a following state satisfying Ψ. The formula $[a]\Psi$ requires that Ψ holds in the following states of any a-transition.

The interpretation of variables ranges over 2^S. The semantics of a formula $\Psi(X)$ with a free variable X is a monotonic function $f : 2^S \to 2^S$. According to the Knaster-Tarski theorem, f has a unique maximal and a unique minimal fixed point. The semantics of the formula $\mu X.\Psi(X)$ is the least fixed point of f. Analogously, the semantics of the maximal fixpoint operator ν is the greatest fixed point of f. The purpose of the inclusion of recursive formulae in μ-calculus is the ability to express usual operators of temporal logics such as **until**, **eventually**, and **always**.

3.4 Service Discovery

The term service discovery has been used with various meanings in the recent service research efforts including those of the Semantic Web Services community. We therefore will shed light on different methods that sometimes have been part of the service discovery problem and provide an interpretation of the terms as they are going to be used in this work.

Definition 7 (Service Discovery)**.** *Discovery is a method in service-oriented systems that automatically locates services for a given request based on their description. In order to allow users and programs (software agents) to make use of the offered services without continuous user intervention, common machine-interpretable languages have to be consequently applied for service offer descriptions and service requests [BLHL01]. The result of service discovery is the set of services fulfilling the requirements of a request.*

Related tasks that are not part of service discovery comprise: crawling, service description mining and learning, service ranking, service selection, and service composition. Crawling of services, as well as the mining and learning of service descriptions are tasks prior to discovery. Service discovery assumes that the service descriptions are already available, accessible, e.g., in the form of a repository of service descriptions, and can be read and interpreted.

Ranking of services computes an ordering of a given set of services based on a given definition of preferences on the ordering. A ranking can be computed after discovery, but it could also be applied prior to discovery, e.g., to influence the order in which services are considered for discovery or to prune the set of services to be considered for discovery at all. The former approach can be helpful for interactive discovery approaches that quickly deliver discovery results successively and interactively. The latter constellation reduces the search space, i.e., the set of services on which the requested requirements have to be evaluated upon. It can lead to increased discovery efficiency but it also removes the completeness of the discovery result set.

Service selection deals with the identification of the best match from a set or an ordered list of services. Composition creates compound services from a set of individual services. Compound services then provide a more complex functionality than the individual services. Service search is a combination of discovery, ranking, and composition techniques. A service search system therefore can deliver discovery results directly, but also produces ranked results by combining ranking and discovery. Furthermore, an on-the-fly composition of services can be computed in order to deliver even more results that match a request.

Matchmaking is a task that compares a request with a service description. If the described service fulfills the requirements of the request, we say that it *matches* the request, and so the service is part of the discovery result set. A *service request* is an abstract description of the (functional and non-functional) requirements of the desired service. We often refer to them as constraints.

In the following, we outline the two main matchmaking techniques that are necessary to understand the contributions of our work. The first technique is the matchmaking of service functionalities, e.g., described by the OWL-S Service Profile or WSMO Capability. Here, we will discuss the conceptual idea of functionality matchmaking. Later in this thesis, we discuss existing implementations of the matchmaking approach in order to distinguish our discovery approach from existing works. The second matchmaking approach that we describe in this section is instantiated by the μ-calculus model checking. It allows evaluating μ-calculus process properties over existing labeled transition systems, which can be described by π-calculus process expressions.

3.4.1 Matchmaking of Functionalities

Similar to the behavior descriptions, there exist various approaches to describe the service functionality. Without being too specific, we want to introduce the basic idea of functionality matchmaking as it has been widely applied in many service discovery approaches.

The functionality of an offered service ω is interpreted as a set of service instances. The requested functionality is commonly described by the same structure. Hence, a requested functionality is also interpreted as a set of service instances (possible solutions of the request).

Within this interpretation of a service and a request, a match is given if there is an overlap between both sets of service instances. Different *matching degrees* are commonly considered: Exact, Plug-in, Subsume, Intersect, and Disjoint. Each degree corresponds to a subset relationship among them. We diagrammed the matching degree in Figure 3.1. An exact match is the strongest relationship. It means that the set of instances described by the offered service is equivalent to the set of instances described by the request. Plug-in and Subsume are less strict matching degrees, as they require that one set is a proper subset of the other. The Plug-in match can be particularly useful as the service provides at least the required functionality. A service that matches a request to the degree of a Subsume match cannot directly be applied, as the remaining functionality of the request, which is not provided by the matching service, has to be provided by other means. The Intersect match is even less strict as it requires a non-empty overlap between the two sets of service instances described

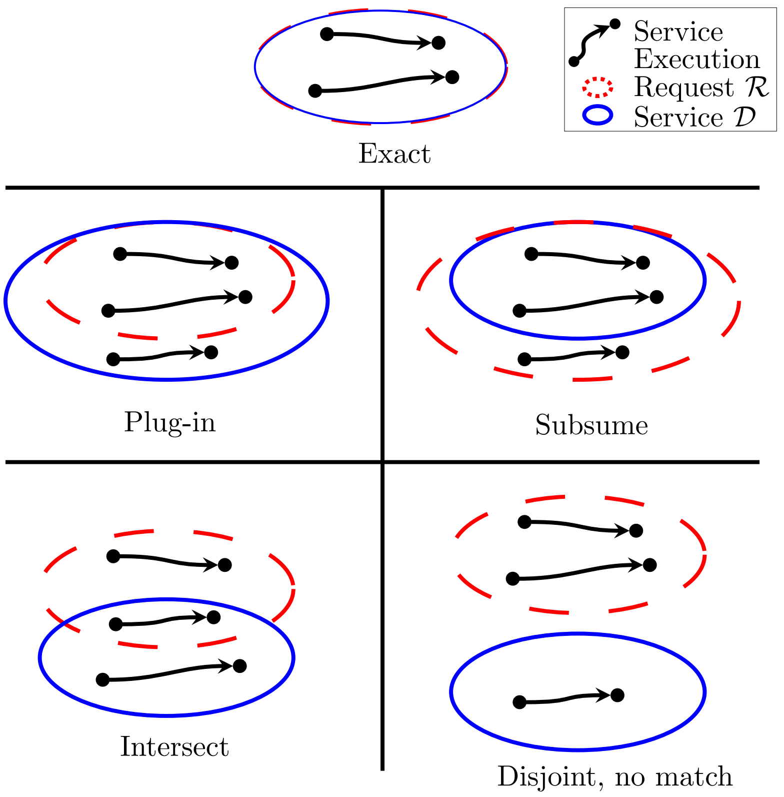


Figure 3.1: Matching degrees between the interpretation of request $\mathcal{R}$ and service $\mathcal{D}$

by the service and the request. If both sets do not intersect, i.e., there is no common service instance, then there is no match.

In our work, we say *intersection-based matchmaking* to refer to the discovery approaches that identify matching services based on the Intersect, Plug-in or Subsume relationship between the sets of service instances that represent the interpretations of service description and request. The intersection-based matchmaking is realized by checking whether the service description subsumes the request or vice versa. However, subsumption checks that can be reduced to the satisfiability problem have a high worst-case complexity of Exptime, even for less expressive description logics like $\mathcal{ALC}$ [DM00, Don03]. Note that our view on matching degrees differs from original definitions, as in [PKPS02], where these degrees have been defined for individual input and output parameters. The presented perspective yet can be abstracted from such definitions.

3.4.2 μ-calculus Model Checking

Model Checking is a technique that automatically and exhaustively checks whether a given model of a system meets a given specification. Formally, the model checking problem is described as follows. Let M_L be the model of a (labeled transition) system and let $\mathcal{R}$ be the expression of a desired modal or temporal property. Then, model checking decides if the system satisfies $\mathcal{R}$ iff $M_L \models \mathcal{R}$ under conditions Ω.

Model checking has been widely applied to the verification of hardware and software systems. For the purpose of service discovery, model checking is applied to properties expressed in the formalism obtained by the combination of μ-calculus and description logics as introduced above. This model checking technique for this particular setting was presented in [Aga07a, ALS09, Aga07b]. We refer to existing literature for a general overview on model checking [EC80, BCM$^+$92, CGP01]. In particular, the model checking of μ-calculus properties is described, for instance, in [EL86, Sti01, BS01].

We now describe how service behavior constraints (i.e., modal and temporal properties) are automatically verified on a given service behavior description. In order to do so, first an LTS is created from a given behavior description (i.e., a π-calculus process expression with DL expressions describing the process resources).

Verification. For a given LTS representation of a behavior, denoted by L, and a given desired property ϕ of a request $\mathcal{R}$, the atomic formula ϕ is verified as follows.

- if $\phi = \mathbf{true}$, then all the states of the LTS L are returned.
- if $\phi = \mathbf{false}$, then an empty set of states is returned.
- if $\phi = P$ then according to the semantics of the request formalism, one needs to find those states of the LTS L in which the proposition P holds. So, all states of the LTS L are iteratively checked, and a state $s \in S$ is added to the result set if the proposition P is entailed by the state knowledge $KB(s)$. As the state knowledge is modeled in form of an ontology and a proposition is an entailment query, the query is executed on the ontology. If the result set of the query is non-empty, then the proposition holds in the state, otherwise it does not.
- if $\phi = \langle a \rangle \phi'$, different types of actions a are distinguished (i.e., input, output, or local). In the model, there are three types of actions, namely, input actions, output actions, and local actions. If there is a transition of the requested type in the source state s that also fulfills the desired properties of the action specified in required action a, then the formula ϕ' in the target state t reached from s through this transition is checked. If ϕ' holds in t, then state s is added to the result set.

Example 4. *Let ϕ denote a desired input action $c[\mathsf{p_1} : \mathsf{C_1}, \ldots, \mathsf{p_n} : \mathsf{C_n}]$. A behavior τ of a system L also contains an input action, say $\tau \overset{\mathsf{def}}{=} c[\mathsf{q_1} : \mathsf{D_1}, \ldots, \mathsf{q_n} : \mathsf{D_n}].\pi$.*

If for each desired input parameter $\mathsf{p_i}$ there exists a parameter $\mathsf{q_j}$ such that $\mathsf{C_i} \sqsubseteq \mathsf{D_j}$, then $c[\mathsf{q_1} : \mathsf{D_1}, \ldots, \mathsf{q_n} : \mathsf{D_n}]$ models ϕ. It also holds, if for each $\mathsf{p_i}$ there exists a $\mathsf{q_j}$ such that $\mathsf{C_i} \sqcap \mathsf{D_j} \neq \emptyset$ under the condition that $\big(typeOf(p_i) \sqsubseteq \mathsf{C_i} \sqcap \mathsf{D_j}\big) \in \Omega$ [Aga07a]. Then, τ matches ϕ and the state prior to the first action in τ is added to the result set.

Composite formulae are broken down to atomic formulae and their result is aggregated from the results of the atomic formulae recursively according to the semantics of the formalism. When the processing of the complete request $\mathcal{R}$ is terminated, it is checked whether the start state of the process definition is in the set of states representing the results of the request $\mathcal{R}$. If this is the case, then the process definition is considered a match for the request $\mathcal{R}$, otherwise it is not.

3.5 Summary

With a brief overview of first-order logic and description logics we have given a summary on knowledge representation formalisms and languages. They are relevant with respect to services and our work as we use these formalisms and languages to model and reason on static aspects of services (such as non-functional service properties and process resources). We also use logic formulae to constrain individual states of the behavior model.

We defined our notion of services as they are seen in the remainder of this work. Existing service description frameworks like OWL-S and WSMO were introduced. Based on them, extensive research under the umbrella of Semantic Web Services is and has been conducted. As such, semantic service discovery approaches have been proposed. We introduced intersection-based matchmaking, which is a technique that is the foundation of many existing semantic service discovery approaches.

Furthermore, we introduced formalisms to describe the behavior of services and to describe temporal properties over behavior descriptions. We described extensions for both formalisms that allow for an additional semantic (DL-based) description of process resources. A summary of the model checking of semantic behavior descriptions has been given. This model checking based matchmaking approach is the starting point for the development of our discovery approach in the following chapters. We will further develop a comprehensive service description model and discuss the applicability of model checking based matchmaking in the context of functionality matchmaking. Then, the efficiency of matchmaking and hence of the service discovery method will be increased by including a service classification and index structures into the matchmaking technique.

The reader should now be familiar with the concepts of semantic service descriptions including process descriptions and their counterpart in requests. Furthermore, the given background on matchmaking of Web services and process expressions should foster the understanding in the following chapters.

Chapter 4

Discovery of Services

In the previous chapters, we detailed the need for service discovery, and outlined fundamentals, requirements as well as approaches to attain such a mechanism. In this chapter, we introduce our contributions for the discovery of services. Specifically, we describe an upper model that we developed for the semantic and comprehensive description of services and their properties. This model enables a unified view on functional and non-functional service properties and allows for a practical service discovery utilizing their description. Furthermore, we develop a declarative query language, which is similarly comprehensive as the description formalism, for expressing service requests. Based on the semantics of offer and request formalisms, we develop an automated and model checking based method for the matchmaking of service descriptions.

We start this chapter with an introduction of the formalism that we use to describe service offers in Section 4.1. An abstract upper model that captures relevant service properties in service descriptions is presented. Then, we show how individual properties are semantically described and hooked to the upper model. We focus on the description of the behavior property in particular, as the functionality and behavior-based service discovery is the most challenging part of this thesis.

A corresponding formalism for expressing service requests is introduced in Section 4.2. In Section 4.3, we describe our discovery method that is based on a model checking approach for behavior properties proposed in [Aga07a]. We extent the existing approach to services and also consider further service properties besides the behavior. Then, Section 4.4 presents the implementation and evaluation results of this discovery method.

The semantic modeling of comprehensive descriptions and requests was developed in the WisNetGrid project and documented in project reports [JA11, WJH11] and publications [AJ10, JAS12, HJA12, JA13, AJ13]. We briefly summarize their content.

- We synthesized behavior descriptions for the composition of Web-based browsing processes [AJ10] in order to automate and simplify tedious browsing tasks as we introduced in Section 2.1.3.
- We applied the formalisms and the discovery method to model and analyze end user browser behavior [HJA12]. It enables the provision of next step recommendations and identification of obstacles in the end user browsing process, among others.

- With the aim of an efficient service discovery method, we developed classification-based [JAS12] and index structure based [JA13] discovery approaches for services with complex behavior.
- In [AJ13], we presented how the semantics of the complex behavior descriptions of browsing processes is captured.
- In [JA11], we reported on the development of this discovery approach and its suitability for the service discovery in the Grid context.
- We could also apply this service discovery approach for locating service in the area of ICT-enabled electric mobility [WJH11].

4.1 Description of Services

We advance state of the art with a formal model of services that establishes a unified view on service properties, which characterize functional as well as non-functional aspects. Our service description formalism allows for the description of independent properties and their values. We consider the behavior and the functionality of a service as just another property with a complex value. The functionality is a profile of the behavior description and describes what the service does. It is typically used for describing atomic services or services whose complete behavior description cannot be disclosed.

In this chapter, we focus on behavior descriptions. We introduce the property-based service model to describe individual services and the set of properties that characterize them in Section 4.1.1. In Section 4.1.2, we show how the property-based service model and the values of individual properties can be semantically modeled. We give an example of a book selling service description in Section 4.1.3.

4.1.1 A Formal Model of Services

We consider a finite set P of service property types and a finite set $\mathcal{V}$ of value sets. Each property type $p \in P$ is associated with a value set $V_p \in \mathcal{V}$. We view a service as a finite set of property instances Q with each property instance $q \in Q$ being of a property type $t_q \in P$ that is associated with a value $v_q \in V_{t_q}$ (diagrammed in Figure 4.1).

We explain our terminology that is used throughout the paper in the following and provide examples.

Definition 8 (Property Type)**.** *A property type $p \in P$ represents a property that characterizes a service and can be part of service descriptions.*

Example 5. *The availability, price, user rating, functionality, and behavior of a service are examples of property types. The first three property types are instances of non-functional service properties. The other two property types are instance of functional service properties.*

Definition 9 (Property Value)**.** *We say property value to refer to the concrete value $v \in V_p$ of a service property type p. The value v is member of the value set $V_p \in \mathcal{V}$, i.e., the set of possible property values of the property type p.*

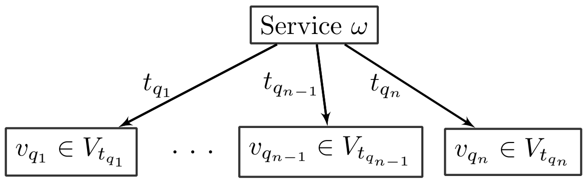

Service ω
t_{q_1}
$t_{q_{n-1}}$
t_{q_n}
$v_{q_1} \in V_{t_{q_1}}$
. . .
$v_{q_{n-1}} \in V_{t_{q_{n-1}}}$
$v_{q_n} \in V_{t_{q_n}}$

Figure 4.1: Formal property-based model of services

Example 6. *Let the availability of a service be 99%. The property value v is 0.99. The property type p is availability. The set $V_{availability}$ of possible property values for this property instance is specified by the set of all real numbers between 0 and 1, i.e., $V_{availability} = [0, 1]$ and $v \in V_{availability}$.*

Definition 10 (Property Instance)**.** *A property instance $q \in Q$ has a key-value structure and represents a property type $t_q \in P$ (the key) that is associated to a property value $v_q \in V_{t_q}$.*

Example 7. *Given the property type and property value from the examples above, the following property instance q represents the availability of a concrete service.*

$$q = (availability, 0.99)$$

Of particular interest is the service behavior describing how the service provides its functionality and how to interact with the service. We assume that every service description has one behavior property at most. Using the above notation, the behavior is a property type $behavior \in P$ with the value set $V_{behavior}$, which is the set of all process expressions that can describe valid service behaviors. The description $\mathcal{D}_\omega$ of a particular service can contain one property instance $q_{behavior} \in Q$. This property instance is associated with a value $\pi \in V_{behavior}$ that denotes the behavior.

Service Behavior Property

In order to support a broad range of services in our service discovery method, we need to model changes introduced by the service execution. Services that provide information typically do not cause changes. However, generic services often cause changes in the knowledge base, i.e., state, of the service provider. In order to support state changing services, we introduce a formal model that captures the service dynamics and use it as the formal underpinning of service functionality and behavior descriptions.

For all intents and purposes, we have to consider a set of actors α identified by a unique identifier. For each actor $a \in \alpha$, we consider a knowledge base KB_a. Furthermore, each actor a can provide a set of services. A service can use other services (of the same or different actors). That is, the execution of a service ω provided by an actor a can cause changes not only in the knowledge base KB_a of actor a, but also in the knowledge bases of actors whose services are used by the service ω. The execution of ω cannot cause any changes in the knowledge bases of the actors that are not involved in the provision of ω. To the best of our knowledge, no logic-based service modeling approach that explicitly models the knowledge of different agents has been presented so far. In our discovery method, we assume that any involvement of actors

$$\begin{tikzcd} s_w \arrow[r, "i"] & s_i \arrow[r, "{[\omega_1]f_1}"] & s_1 \arrow[r, "{[\omega_2]f_2}"] & \dots \arrow[r, "{[\omega_n]f_n}"] & s_n \arrow[r, "{[\omega_e]f_e}"] & s_e \end{tikzcd}$$

Figure 4.2: Formal behavior model of services

is explicitly modeled with the agent invocation process description element (cf. Section 3.3.2). For a single state transition, we assume that we need to consider only one actor or several actors whose knowledge bases can be integrated into a single knowledge base of the state. Our assumption means that we cannot model local actions, which are executed by multiple actors. Communication actions cannot be executed by multiple actors.

Figure 4.2 illustrates the behavior of a service as a labeled transition system $L = (S, \rightarrow, A, \lambda)$ as defined in Definition 3. Each state of the LTS represents the provider's knowledge that holds at the respective state during the service execution. As elaborated in the following, the execution path (*trace*) can be seen as a series of connected states:

(1) The start state $s_w \in S$ represents the knowledge available to the service before the service is invoked. It is described by the initial knowledge denoted as the propositions $\lambda(s_w)$. The parameter values passed with the service invocation are not available.

(2) A state $s_i \in S$ comprises the knowledge after the input action i. This state contains the input parameter values that are bound to the individuals that represent the input parameters in this state. In addition to that, the knowledge that holds in s_w still holds in s_i, since the input action does not alter the existing knowledge from s_w.

(3) A series of states $s_1, \dots, s_n$ of length $n \geq 0$ represents the knowledge that occurs during the execution of a service while computing the output values, interacting with external agents, and performing any changes with actions $[\omega_1]f_1, \dots, [\omega_n]f_n$. An action f_j (with $1 \leq j \leq n$) in a state $s_j \in S$ takes place only if condition ω_j is true in the previous state s_{j-1}.

(4) An end state $s_e \in S$ represents the knowledge available to the service after it has performed all the changes. The values of the output parameters are part of the end state description.

The transitions $i, f_1, \dots, f_n, f_e$ of $\rightarrow$ can take place only if their respective source state is consistent. In particular, after the input values are available in state s_i, the trace continues only if s_i is consistent. Any conditions on the input values are available as part of the knowledge in state s_i. The transitions are annotated with a description of the respective action of A that causes the transition. In our behavior model, we focus on the description of valid and successful traces. If the conditions are not true or a state is inconsistent, we assume that the service fails. However, additional branches can be added to behavior descriptions such that alternative continuations are available if a condition does not hold. Inconsistent state knowledge and the asynchronous appearance of failures and exceptions are not considered in our current model.

The input transition i is a communication action that receives the input parameter values passed along with the service invocation by the consumer. The transitions $f_1, \dots, f_n$ are either communication actions or local actions performing computations within the knowledge base as well as adding or deleting facts in the provider's knowledge base. The complete LTS representing the behavior of a service may also include branches. That is, there can be alternative paths to successive states. We have omitted them here to improve the readability of Figure 4.2 and the above description.

Non-Functional Properties

Non-functional properties (NFPs) are part of comprehensive service descriptions and supplement the functional aspects of services. In contrast to the functional properties like the service behavior, NFPs describe manifold quality attributes of services. We consider NFPs that can be expressed independent of each other in a key-value structure. In many scenarios, like the ones presented in Section 2.1 or in an Internet of Services, there can be multiple services offering the same or similar functionalities that are suitable for a given purpose. Non-functional quality attributes of the services enable to differentiate between them, e.g., in service discovery, ranking, and selection.

We observed that existing service discovery approaches are usually limited to the matchmaking of functional aspects of services and that non-functional requirements[1] are exclusively considered for computing a ranking over services [ALS09]. Specifically, an external ranking method is invoked after the functionality has been matched. The ranking methods evaluate the non-functional requirements and compute an ordering for a given set of services. These approaches, e.g., [VHA05, DFGC08, GJR$^+$13], limit the functional requirements to hard constraints and are used to filter the result set computed by the discovery method. However, the requirements over non-functional properties are often expressed as preferences and perceived as soft constraints that influence the ordering of the result services returned by the ranking method.

It is not determined per se whether requested service properties are interpreted as hard or soft requirements and it is equally valid to perceive non-functional requirements as hard requirements, too. For example, consider the NFP *availability* with a value of 0.99 in a service description. Exploiting this information to filter the result set in the discovery phase allows to accurately describe desired services and to obtain accurate results from the discovery method. For that purpose, a request may specify that desired services have to offer an availability of at least 90%, which is considered to be a hard requirement. That is, non-functional requirements are also valid filter criteria that can be used for matching services within the discovery phase. Likewise, a ranking component may also sort services based on a degree of adequateness of both functional and non-functional requirements.

In order to enable service discovery based on NFPs, the property-based model of service descriptions is flexible enough to include any kind of independent non-functional properties that can be expressed by a key-value structure. That is, we assume that the values of NFPs do not depend on the values of other NFPs. Also, complex property values (similar to the complex structure of the functional properties) can be included in service descriptions and considered during matchmaking. While we propose matchmaking semantics for the functionality and the behavior properties, we refrain from dictating the future approaches that can be used to match complex NFPs. For example, policies that restrict the usage of services as in [Spe12] are encapsulated into a complex NFP value with a deferred evaluation by a dedicated policy compliance checking framework.

[1]Non-functional requirements are part of service requests and express constraints or preferences on non-functional properties of a service description.

4.1.2 Service Description Language

We now introduce how we express the formal model of services as presented above in a language using an ontological representation of the model. Because it is not feasible to assume a global set of property names, we model service properties as properties in a known ontology language, e.g., using RDF/S [BG04], OWL 2 [OWL09], or WSML [dBFK+08]. Also, the use of standardized ontology languages allows us to apply existing ontology reasoners to reason about service properties while not enforcing a global set of property names. More precisely, we

- define for each property value set $V \in \mathcal{V}$ an ontology concept V. Furthermore, we assume a set of common data types either available directly or modeled as ontology concepts as well.
- define for each service property $p \in P$ with an associated value set V_p an ontology object property p with range $\mathsf{V_p}$ if V_p is a set of individuals. Otherwise, if V_p is a data type, we define an ontology data type property p with range $\mathsf{V_p}$.

Regarding the openness of the Web, semantic service descriptions are considered to be highly heterogeneous as different actors may use different vocabularies and ontologies. As ontology languages allow for an alignment of concepts and properties in subclass-of and subproperty-of relationships respectively, we fulfill requirement R4 and achieve interoperability among the service properties without demanding a global set of property names. For overviews on the ontology alignment and mapping problem we refer to existing literature, e.g., [Ehr06, ES07, RVNLE09].

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix supSvc: <http://suprime.aifb.kit.edu/service/>.

supSvc:Service a rdfs:Class .
supSvc:Property a rdfs:Class .
supSvc:FunctionalProperty rdfs:subClassOf supSvc:Property .
supSvc:BehavioralProperty rdfs:subClassOf supSvc:FunctionalProperty .
supSvc:NonFuntionalProperty rdfs:subClassOf supSvc:Property .

supSvc:hasProperty a rdf:Property ;
    rdfs:domain supSvc:Service ;
    rdfs:range supSvc:Property .
supSvc:hasFunctionalProperty rdfs:subPropertyOf supSvc:hasProperty ;
    rdfs:domain supSvc:Service ;
    rdfs:range supSvc:FunctionalProperty .
supSvc:hasBehavioralProperty rdfs:subPropertyOf supSvc:hasFunctionalProperty ;
    rdfs:domain supSvc:Service ;
    rdfs:range supSvc:BehavioralProperty .
supSvc:hasNonFuntionalProperty rdfs:subPropertyOf supSvc:hasProperty ;
    rdfs:domain supSvc:Service ;
    rdfs:range supSvc:NonFuntionalProperty .
```

Listing 4.1: RDF/S representation of the property-based service model

The property-based service model introduced in Section 4.1.1 above is easily mapped to an ontological representation. In Listing 4.1, we present the RDF/S representation of the upper model using the Turtle RDF serialization format.[2] OWL 2 and WSML representations

[2]Turtle, the Terse RDF Triple Language, is a concrete syntax for RDF. It is specified in http://www.w3.org/TeamSubmission/turtle, retrieved 2013-06-29.

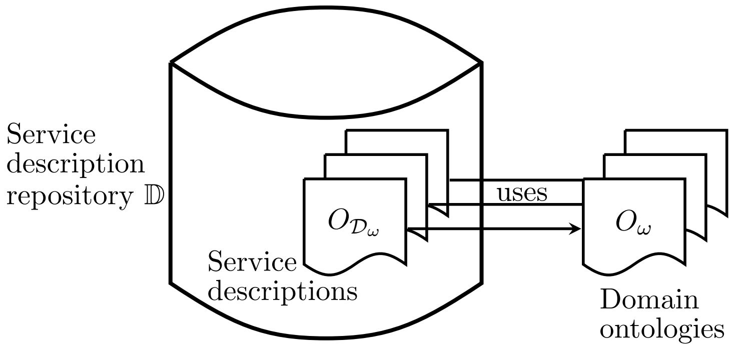


Figure 4.3: Service description ontologies $O_{\mathcal{D}_\omega}$ and domain ontologies O_ω

of the upper model can be expressed analogously as the basic concepts (classes, sub-class-of relationships, properties, and so on) are defined in any of these languages.

We now turn our attention to the ontological modeling of the individual service properties. We assume that a domain ontology O_ω is attached to or referenced by the description of a service ω and is available to the discovery engine. O_ω defines a TBox with static knowledge about the conceptualization of the domain. The TBox is assumed to be static in our work. It means that the conceptualization of the domain will remain unchanged during each service execution. The changes caused by the services are limited to the ABox. It means that individuals of the ABox and the relationships between them can be changed, created, or deleted during the execution.

If we would allow for changes in the TBox during the service execution, we could not 'inherit' the knowledge of a state from the previous state. For instance, consider that the specification of an ontology property is changed. The semantics of how it is used before this change is different from the one after the change. So, the axioms that hold in the states before the change have to be excluded in the following, which typically does not happen in the implementation of a service. Within the implementation it is still possible to access the previously computed results. Simply adding knowledge to the TBox during the service execution may not always introduce inconsistencies. Regardless of this, in our work we assume that the TBox is already complete in the beginning.

As depicted in Figure 4.3, the description $\mathcal{D}_\omega$ of service ω refers to a domain ontology O_ω that specifies the static TBox with concepts of the domain of discourse. The service description $\mathcal{D}_\omega$ utilizes the concepts from O_ω in order to describe the service ω. The ontological representation of the service description $\mathcal{D}_\omega$ is another ontology that is denoted by $O_{\mathcal{D}_\omega}$.

Behavior. The behavior property is described by a process expression π and uses concepts from the domain ontology O_ω to describe the process resources in π semantically. In the service description ontology $O_{\mathcal{D}_\omega}$, we model the service behavior as an ontology instance of the concept supSvc:BehavioralProperty. We provide a serialization of the behavior description π using an additional vocabulary to represent language concepts of the π-calculus (including input action, input parameter, process invocation, and so forth). In the following examples of behavior descriptions, this vocabulary is identified by the prefix http://suprime.aifb.kit.edu/process/ or its abbreviation supProc:.

Functionality. The service functionality as a profile of the behavior description is added as an instance of the concept supSvc:FunctionalProperty to $O_{\mathcal{D}_\omega}$. This instance links to input and output parameters, a description of the start state and the end state, as well as changes caused by the service execution. We further elaborate in Section 5.2 on the issue of modeling this service profile consistently.

Non-functional properties. In our model, NFPs are independent of each other and do not change during the service execution. Thus, they can be modeled in $O_{\mathcal{D}_\omega}$ independently from the state-based functional description. They can be directly expressed as instances of the class supSvc:NonFunctionalProperty and a domain specific concept defined in O_ω describing the type of the property value. The use of general purpose data types is already allowed in many ontology languages. For example, OWL 2 supports data types from XML Schema, the specification of data ranges and literals [HKP+09]. Ontology reasoners like HermiT[3] for OWL 2 also support the reasoning on OWL 2 data types, which is sufficient for simple NFPs with a value that can be expressed by a standard data type. Otherwise, specific reasoning techniques to evaluate complex NFPs can be added as an extension to ontology reasoners.

4.1.3 Modeling Example

We now present an example of a service description using our formalism. Our service ω offers a book ordering functionality to customers. In addition to selling books, the provider of the service ω offers further functionalities like querying the availability or the price of given books. Each functionality is provided by a multi-step interaction with the provider.

The interaction with the service can be accomplished programmatically by a machine in the customer role. Alternatively, it is also possible that the service provides a more end user ready user interface that allows humans to interact with the service, e.g., in a Web browser. Regarding the service description model, both possibilities can be modeled similarly.

After the invocation of the service, the consumer has to provide the search parameters. The desired book is identified by its author and title. Using the π-calculus syntax, the input action is expressed by the following process expression.

$$\mathsf{ex{:}seq0prefixConn}[\mathsf{ex{:}author}, \mathsf{ex{:}title}].\pi$$

This expression is part of the overall behavior description. Here, π denotes the remaining service behavior. The input parameters ex:author, ex:title, and the communication channel ex:seq0prefixConn are further specified by a set of ontology axioms attached to the input action description. These axioms are given in Listing 4.2. We introduce the resource _:book to the description in order to explicitly describe that the author and the title actually refer to the same book. We use an RDF blank node as the individual representing the concrete book that is not yet identified.

Our example service continues after the input action with a local action that checks the availability of the specified book. It returns the identified offer to the consumer in an output action. Based on this information returned to the customer, she has the option to order the book. In order to do so, the consumer has to continue by providing her log-in credentials in a

[3]HermiT OWL reasoner – http://hermit-reasoner.com, retrieved 2013-06-11

Table 4.1: Sequential actions of the example trace

Pos.	Activity (informal)	Process Expression
1	Input of author and title of a book	ex:seq0prefixConn[ex:author, ex:title]
2	Local action checking availability	$\texttt{checkAvailability}^{\Delta}$
3	Output of the identified book	ex:seq2prefixConn⟨_:book⟩
4	Input of log-in credentials	ex:seq3prefixConn[ex:user, ex:password]
5	If user details complete, output of order	ex:Ready(ex:user)?ex:seq4prefixConn⟨ex:order⟩
6	Input accept to confirm the purchase	ex:seq5prefixConn[ex:confirmation]

subsequent input action. If all user information, like preferred payment method, delivery type and address is already available, the service emits a summary of the order details. Finally, the costumer can confirm the purchase and payment. The order of actions for this particular trace is summarized in Table 4.1. As we consider one particular trace only, our description simply contains a sequence of actions.

```
ex:author a ex:Author .
ex: title  a ex:Title  .
_:book a ex:Book ;
    ex:hasAuthor ex:author ;
    ex:hasTitle ex: title  .

ex:seq0prefixConn a supProc:Connection ;
    a <http://dbpedia.org/resource/HTTP> ;
    supProc:hasReceiver ex:bStore .
```

Listing 4.2: Semantic annotation of the first input action.

The description of the whole service behavior will become more complex (e.g., including alternatives) if we consider any other traces. E.g., additional traces comprise the cases in which no corresponding book can be found or is in stock, the payment options have to be updated, the shipping address was not recorded, and so on. We limit this example service to the basic functionality. It is sufficient to highlight and illustrate the concepts we develop in this thesis.

Service description. In Listing 4.3, we show an excerpt of the RDF/S representation $O_{\mathcal{D}_\omega}$ of the service description $\mathcal{D}_\omega$. The presented listing comprises three non-functional properties and an excerpt of the behavior description. The service ω is represented by the individual ex:bStore. In addition to the behavior description represented by the individual ex:bStoreBehavior, the non-functional properties of the service refer to a rating of the service and specify the average as well as the maximum delivery time as two and five days, respectively.

The behavior description of the service begins with the sequence _:seq0 that has the input action _:seq0prefix as its first element. This input action accepts the two parameters ex:author and ex:title at port ex:seq0prefixConn. The port ex:seq0prefixConn denotes the communication channel of the input action. It connects the service provider with the invoking service consumer and is used to establish the communication of the two input parameters. The channel ex:seq0prefixConn may use any communication protocol. If the service provides a Web-based end user interface, the HTTP protocol is used to pass ex:author and ex:title to the service.

The input action further specifies ontology axioms that describe the input parameters. The property supProc:hasSemanticRelation relates a set of OWL 2 ontology axioms to the action. When the service description is translated into a labeled transition system, these axioms are added to the subsequent state knowledge.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix supSvc: <http://suprime.aifb.kit.edu/service/> .
@prefix supProc: <http://suprime.aifb.kit.edu/process/> .
@prefix rev: <http://purl.org/stuff/rev#rating> .
@prefix ex: <http://example.com/books#> .

ex:bStore a supSvc:Service ;
    supProc:hasOntologyPhysicalIRI "http://example.com/book-shop-onto.owl" ;
    rev:rating rev:Excellent ;
    ex:avgDeliveryTime "2D"^^xsd:period ;
    ex:maxDeliveryTime "5D"^^xsd:period ;
    supSvc:hasBehavioralProperty ex:bStoreBehavior .

rev:rating rdfs:subPropertyOf supSvc:hasNonFunctionalProperty .
ex:avgDeliveryTime rdfs:subPropertyOf supSvc:hasNonFunctionalProperty .
ex:maxDeliveryTime rdfs:subPropertyOf supSvc:hasNonFunctionalProperty .

ex:bStoreBehavior a supSvc:BehavioralProperty ;
    supProc:hasBehavior _:seq0 .

_:seq0 a supProc:Sequence ;
    supProc:hasPrefix _:seq0prefix ;
    supProc:hasNext _:seq0next .

_:seq0prefix a supProc:Input ;
    supProc:hasConnection _:seq0prefixConn ;
    supProc:hasVariable _:author , _:title ;
    supProc:hasSemanticRelation
        "ClassAssertion(<http://example.com/books#Author>␣_:author)" ,
        "ClassAssertion(<http://example.com/books#Title>␣_:title)" ,
        "ClassAssertion(<http://example.com/books#Book>␣_:book)" ,
        "ObjectPropertyAssertion(<http://example.com/books#hasAuthor>␣_:book␣_:author)" ,
        "ObjectPropertyAssertion(<http://example.com/books#hasTitle>␣_:book␣_:title)" .

_:author a supProc:Variable ;
    supProc:hasValue supSvc:Constant1906 .
_:title a supProc:Variable ;
    supProc:hasValue supSvc:Constant1907 .
```

Listing 4.3: Excerpt of ontology $O_{\mathcal{D}_\omega}$ describing the example book selling service ω.

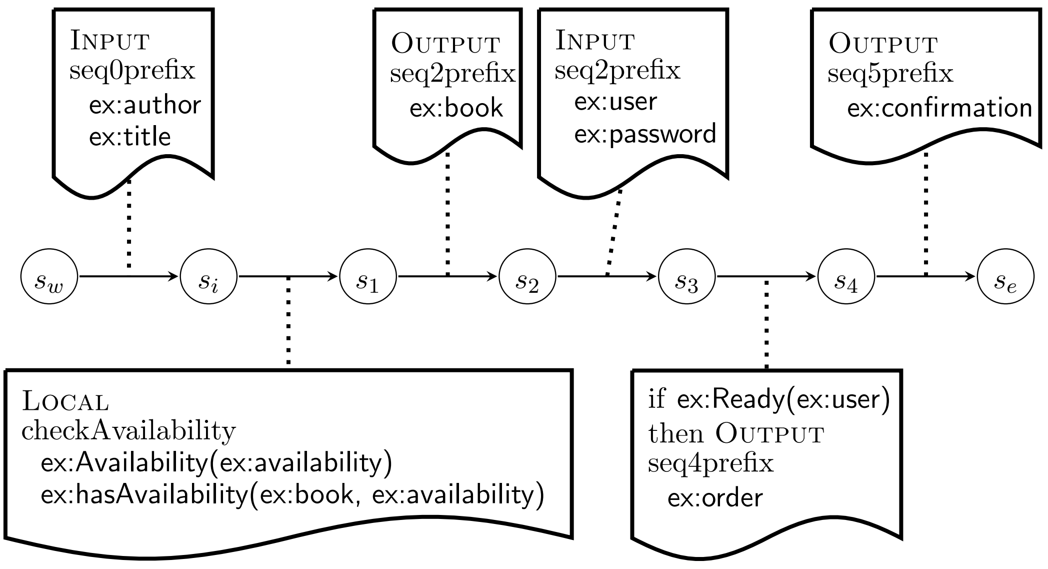


Figure 4.4: LTS representation of the example service

As a single ontology $O_{\mathcal{D}_\omega}$ contains multiple actions with multiple axioms that hold in various states after each action, the axioms can introduce contradictions to $O_{\mathcal{D}_\omega}$. In order to avoid these problems, we exclude the semantics of the axioms from $O_{\mathcal{D}_\omega}$ by adding the axioms as literals (as shown in Listing 4.3). We could also use stratification, however, the first option will be sufficient in our use case. The literals contain OWL axioms that are parsed and added to the respective state knowledge bases during the matchmaking of services.

Labeled transition system. After we explained the general structure of the example service, we now show how the LTS representation of the behavior is derived from the process expression and its semantic annotations. The LTS for the transitions shown in Table 4.1 comprises seven states $S = \{s_w, s_i, s_1, s_2, s_3, s_4, s_e\}$. Figure 4.4 depicts the example LTS with an informal representation of the semantic annotations of state transitions. Since we did not specify any conditions on the start state and there are no process arguments passed with the invocation in our example, the state knowledge $KB(s_w)$ is empty. With the transition $(s_w, \text{\_:seq0prefix}, s_i) \in \rightarrow$ modeling the first input action _:seq0prefix, we derive the state knowledge $KB(s_i)$ of the state s_i by computing the union of the description of the input parameters (see Listing 4.2) and the previous state knowledge, i.e., $KB(s_w)$.

```
ex:author a ex:Author .
ex: title  a ex:Title  .
_:book a ex:Book ;
     ex:hasAuthor ex:author ;
     ex:hasTitle ex: title  ;
     ex: hasAvailability  ex: availability   .
ex: availability  a ex: Availability  .
```

Listing 4.4: State knowledge $KB(s_1)$ of state s_1.

The next state knowledge of the sequential process is derived similarly. Local actions can add, update, and delete existing knowledge, i.e., of $KB(s_i)$ in this case. The operation `checkAvailability`$^\Delta$ adds availability information to the existing resource _:book. The state knowledge $KB(s_1)$ is shown in Listing 4.4.

We annotate local actions with a set of changes that are caused by the execution of that local action and introduced to the subsequent state. A change δ is described by its type $chType(\delta) \in \{+, \sim, -\}$ that states a fact was either added (symbolized $+$), updated ($\sim$), or deleted ($-$) in the ABox of the state knowledge. The fact itself is an ontology axiom. In our example, the local action `checkAvailability`$^\Delta$ comprises two changes that add the following two facts to the ABox of $KB(s_1)$.

$$\begin{gathered}\mathsf{ex{:}Availability(ex{:}availability)}\\ \mathsf{ex{:}hasAvailability(\_{:}book, ex{:}availability)}\end{gathered}$$

The availability of the specified book is positive in the trace that we consider in our example. Therefore, the value of the availability is assumed to be true. For a complete description of the behavior, a distinction of the cases would be required for the continuation of the process.

In Section 5.4, we discuss the application of our discovery method to atomic services and focus on the functionality profile exclusively. The functionality description of our example service will be introduced in this context, too.

4.2 Service Request Model

The formalisms to describe service offers and requests must provide sufficient expressivity to allow users to create comprehensive descriptions and to precisely formulate constraining requirements in requests, respectively. Users should rather be limited by their willingness to invest effort than by any technical limitation. Our service discovery method allows users and machines to pose requests. Therefore, a formal query language that facilitates automation is a prerequisite. Furthermore, in order to use services automatically, for instance in service compositions, the query language must allow for the specification of constraints in a comprehensive and logically correct manner such that the desired services can be precisely described. Otherwise, the accuracy of discovery results may be insufficient for many use cases that require an automated service discovery. We will show that an intersection-based matchmaking is insufficient to ensure applicability of services for a given request.

Definition 11 (Desired Service)**.** *A desired service is a service that is described in the service repository and provides the desired service properties in its entirety. The desired service properties are described in the service request by means of constraints (on the property values). If the request describes alternatives, a desired service fulfills at least one of these alternatives. In a request, an alternative is a description of a set of desired service properties.*

Although formal service requests are used, their use by humans can be simplified by the provision of an unambiguous and thus comprehensible interpretation of service descriptions and requests. For example, it needs to be clear to users whether the requested inputs are interpreted as inputs that must or can be accepted by desired services. In the scope of this work, we do not investigate usability aspects, which could be provided by means like a simple syntax, a query language for expressing formulae or a supportive user interface. To this extent, our approach differs from goal driven approaches as in [SHH07] by not focusing on an abstract and non-technical request (goal) description that a user creates. Such user goals have to be translated into machine understandable requests. The latter representation of a request is the starting point of our work.

After we described above how we model comprehensive service descriptions semantically, we now introduce our model for service requests. The expressivity of the request model coheres to the expressivity of the service description model. That is, we aim at expressing constraints over any information captured in service descriptions and the expressivity of requests should not allow more, because a high expressivity always involves a higher computational complexity and consequently decreases the efficiency.

We start in Section 4.2.1 with a motivation for the need of a model for service requests, which differs from the service description model. After that, consistent with the structure of the previous section, we introduce in Section 4.2.2 the request model on a higher level of abstraction. This upper level model for requests directly corresponds to the property-based service model for service descriptions. We focus on the modeling of constraints over individual properties and their ontological representation in the subsequent Section 4.2.3.

4.2.1 Motivation for a Discrete Request Model

Before introducing our model that is used to express service requests, we discuss the reasons for having a request model, which is different to the one used in service descriptions. Assume that we would not introduce a distinct request model, then both descriptions and requests are declarative. However, declarative service descriptions have to be hand crafted while the procedural ones can often be generated, at least to a great extent if not completely, from the code. Because of this, the hand crafted descriptions are also hard to manage since they are not connected with the code and every time the implementation of a service changes, the service description has to be manually adapted accordingly.

We discussed in [JAS10, JA10] that the development of a practical service discovery approach requires expressive service description languages that are non-ambiguous, easy to use as well as support heterogeneous descriptions. Furthermore, an effective service request formalism used to query service repositories needs to address these requirements analogously.

Conceptual Mismatch

Although most of prior service discovery approaches developed by the Semantic Web Services community provide a formal semantics, their pragmatics to describe service requests is improper since it differs from the user intention. The common problem of most Semantic Web Service discovery approaches is that they apply the same formalism for describing service offers and service requests. We investigate the problem of using a single formalism for service requests and service offer descriptions further in the following paragraphs. In contrast to existing works, we develop in Section 4.2.3 a request formalism that allows describing a set of potentially matching services in a service request. The definition of a discovery match and a matching algorithm for effective service discovery based on the given semantics of the request formalism are presented in Section 4.3.

It is an intuitive interpretation that a service description formalizes the actual values of the service properties and a service request specifies the acceptable set of values of the respective properties. Therefore, using the same formalism with same interpretation for both service description as well as request is counter-intuitive to users requesting services. Such mismatch between the semantics of formalisms and their intuitive interpretation of the requester makes these approaches hard to use in practice. Furthermore, the use of the same interpretation of service descriptions and requests limits the expressivity of service requests, because alternatives and negation are typically not supported in service description formalisms. It also decreases the result accuracy if similarity-based matchmaking is applied to increase the flexibility of matching the requested service description with offers.

A service description (middle layer in Figure 4.5) is an abstract representation of multiple service instances (bottom layer), i.e., a description aggregates all different service traces that are possible. For simplicity, only successful traces are often considered in service descriptions. Service discovery approaches like [PKPS02, LH03, CFB04, dBKZF09, SHF11] that use the same level of abstraction for the description of a service request therefore describe a (one) desired service. Such approaches rely on the query-by-example (QBE) paradigm [Ull88, Chapter 4.4] to retrieve matching services from a repository. Unlike QBE approaches from the database domain, where the specified sample is translated into a structured query, QBE service discovery is similar to QBE in the domain of information retrieval. Here, the similarity between the sample specified in a query and the existing entries is computed. Based on a similarity metric, matches can be determined and possibly a ranking can be derived from the degree of similarity [PKPS02, LH03].

The matchmaking based on the similarity between a desired service described in a service request and the service offers is not appropriate in our settings (considering the requirements presented in Section 2.2) for the following reasons:

- The correctness of logic-based matchmaking algorithms is favorable for automated tasks like service invocation or the integration of a replacement service into a given service system.
- An automated use of similar search results is very complicated. Consider for example that there are no services matching a given request. Then, available services that are similar to the request can be of interest and may indicate to the user that the original request can be reformulated, e.g., such that over-specifications of requested constraints are removed. Although users posing service requests may appreciate similarity-based matchmaking under some circumstances, further use cases, e.g., without user involvement, need to be considered by a service discovery approach as well.
- Existing discovery approaches do not automatically provide an explanation for a match. That is, the computed similarity and dissimilarities have to be examined manually.
- If a use case targets humans that create and pose the service requests, reasons like the simplicity of formulating a request by describing a desired service cannot justify the use of a single formalism. Even worse, the semantic description of a service is a time-consuming effort that requires at least a basic understanding of the underlying

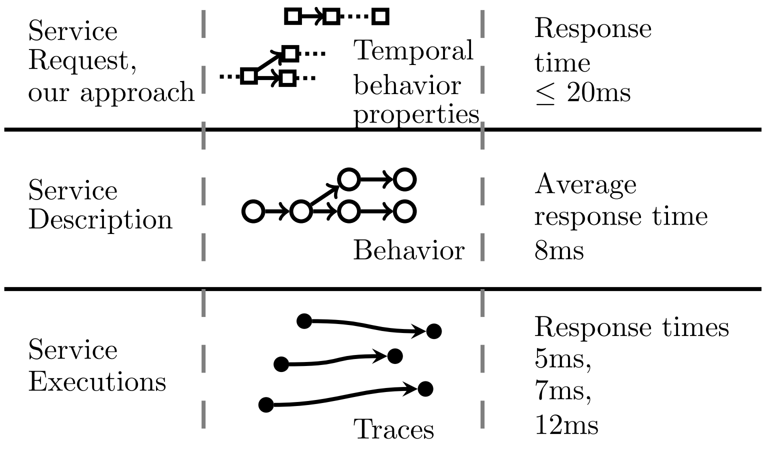


Figure 4.5: Different layers of abstraction in requests, descriptions, and executions, whereas requests are expressed in a different formalism in our approach.

formalisms. Consequently, users or agents posing a request will not benefit from using service descriptions in requests, as the semantic descriptions of service offers are created by the provider with expertise about the implementation and the domain.

The use of a dedicated request formalism with defined semantics is common in a plethora of information systems with a formal underpinning. For instance, SQL is used to query relational databases, Datalog for deductive databases, SPARQL for RDF data sets, and XQuery and XPath for XML documents. The wide acceptance of query languages for the retrieval tasks confirms that they are not counter-intuitive to (technically skilled) users. For inexperienced end users, simplified user interfaces that guide the specification of a request can be developed on top of any query language.

The clear definition of a match based on the semantics of query and description languages is another benefit as it allows implementing algorithms and developing tools in a logically correct manner. In a QBE approach, there are several ways to define a similarity function, which always has to accompany the definition of a match. Also, users posing requests and examining the discovery results constantly have to conceive the implemented similarity function in order to understand the results.

In summary, we constitute that the use of a dedicated service request formalism for the targeted scenarios and use cases of service discovery and for the fulfillment of the identified requirements on our discovery method is advantageous. Yet, we want to examine the differences and the impact on the matchmaking technique if a dedicated request formalism is used or not. In Figure 4.5, we depict the informal semantics of the state-based service descriptions (in the middle layer). Service descriptions abstract from the concrete executions depicted in the bottom layer. The formalism for the description of the functionality of service profiles has prevalently been applied for service requests [PKPS02, SPAS03, LH03, CBF05, SHH07, GLA^{+}04]. That is, a dedicated request model as depicted in the top layer Figure 4.5 can describe desired properties and specifies the set of desired services. Instead, the description formalism is used to express requests in existing works that did not introduce such an additional layer, i.e., a dedicated request model. Then, different degrees of a match (exact, plug-

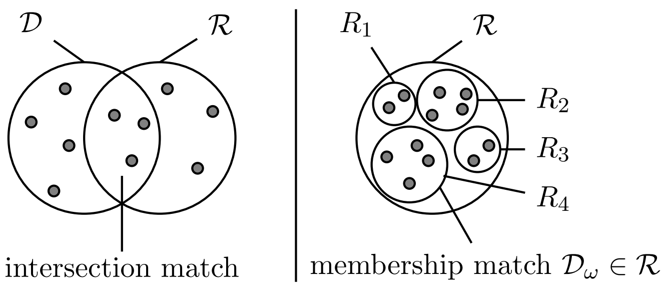


Figure 4.6: Discovery approaches that use the same formalism for an offer $\mathcal{D}_\omega$ and a request $\mathcal{R}$ are based on intersections (left). Using two different formalisms allows specifying the (exact) matches in the request (right).

in, subsume, intersect, none) between an offered service and a request are computed by a matchmaker. However, such an intersection-based matchmaking decision (using the mentioned different matching degrees) is an impractical and unintuitive approach, because the mismatch between the semantics of formalisms and the intuitive interpretation of the requester makes the description, requests, and discovery methods hard to use in practice.

At this stage, we want to highlight that the choice of using a dedicated request formalism or not is independent from the matchmaking semantics that used for service discovery. However, all the existing works mentioned above combine the use of a single formalism for descriptions and request with an intersection-based matchmaking semantics using different types of matching degrees.

Discovery Result Accuracy

While intersection-based matchmaking (as in [PKPS02, LH03, CBF05, SHH07]) offers more flexibility, it also requires further effort to evaluate matching services in order to ensure the applicability for the desired tasks at hand. That is, the search result accuracy of intersection-based discovery methods is insufficient and does not fulfill our requirement R2 as presented in Section 2.2. Consequently, the freedom provided by the different matching degrees is not feasible for the purpose of automated service invocation or composition.

As an example, consider a service that offers to ship goods from a city in the UK to a city in Germany. A user requests a shipping provider that operates between European cities. Intersection-based matchmaking methods identify the mentioned service as a match. However, if the requester uses this service to ship an item from Berlin to Hamburg, then the matched service offer fails. The reason is that intersection-based matchmaking cannot guarantee that a matching service can be successfully invoked. Figure 4.6 shows the principle of intersection-based matchmaking on the left. A service description describes a set of different service instances, depicted by the dots, where each instance represents a trace for a particular combination of input parameter values. If the same formalism is applied to service descriptions and requests, then the service request again describes a set of traces. A service matches a request if both have at least one instance in common.

In general, if a matching service ω provides some of the requested traces (i.e., intersect match), then it does not mean that ω is able to provide the requested functionality under any circumstances. It is possible that ω behaves as requested for some specific traces and ω shows an undesired behavior in other traces. In order to assess the applicability of ω, a user or a program needs to identify these configurations in which the service behaves as requested or not [GLA+04]. Such applicability estimation for intersection-based matchmaking was not considered in existing approaches so far.

It is also possible that, for instance, another service ω' is an intersection match with a request if the set of service output parameters intersects the set of desired outputs. Consequently, additional services to ω' are required that produce the complete required outputs. Sometimes these additional services need to be compatible to the behavior of ω' in order to interact with it. In order to invoke or integrate ω' as well as the additional services into a given service system, automatic planning techniques, which quickly become theoretically challenging [RS04], become necessary.

The right side of Figure 4.6 shows the matchmaking principle of our matchmaking approach in contrast to intersection-based techniques. A service description still represents a set of instances. A request represents a power set of service instances. Each set $R_i \in \mathcal{R}$ in the power set describes a potential service that fulfills the requirements of the request. We will apply a matchmaking technique that checks whether there is an interpretation (model) of the request that is equivalent to the offered service $\mathcal{D}_\omega$. Therefore, we apply a model checking based matchmaking that evaluates whether $\mathcal{D}_\omega$ is a model of $\mathcal{R}$ ($\mathcal{R} \models \mathcal{D}_\omega$).

Negations in Requests

If the requester wants to prohibit specific service properties, then it is not possible to ensure that intersecting services fulfill these constraints, because the excluded property of the request is not necessarily a part of the intersection. As we aim at the development of an automated service discovery method (see requirement R1) that (i) delivers accurate discovery results (requirement R2), and (ii) is expressive enough to cover practical use cases (requirement R3), the request formalism has to provide the expressivity to specify desired and rule out undesired service properties.

Efficiency

The efficiency of the service discovery method is also affected by the chosen formalisms used to describe services and requests. Certainly, the expressivity of the languages correlates to the computational complexity of reasoning about expressions of these languages. And as a service discovery for highly expressive descriptions is required in order to support various use cases, we do not trade off expressivity for efficiency.

Instead, the way that requests are matched against service descriptions impacts the performance of a service discovery method. That is, choosing a matchmaking technique that efficiently detects the services that fulfill the constraints of a request impacts the discovery efficiency.

The existing service discovery approaches referenced above, which use an intersection-based matchmaking technique, compute the subsumption relationship among service description and request. In order to identify a match, the matchmaker evaluates if the offered functionality subsumes the request or vice versa (depending on the matching degree to be computed).

$$\begin{tikzcd}
 & & \boxed{\text{Service Request } \mathcal{R}} \arrow[dll, "t_{q_1}"'] \arrow[d, "t_{q_{m-1}}"] \arrow[dr, "t_{q_m}"] & \\
\boxed{V_{R,t_{q_1}} \subseteq V_{t_{q_1}}} & \cdots & \boxed{V_{R,t_{q_{m-1}}} \subseteq V_{t_{q_{m-1}}}} & \boxed{V_{R,t_{q_m}} \subseteq V_{t_{q_m}}}
\end{tikzcd}$$

Figure 4.7: Formal property-based model of service requests

The satisfiability problem and, thus, the subsumption problem has been shown to be Exptime-complete [EJ99, Theorem 6.3], i.e., one on the hardest problems in Exptime, even for one of the least expressive description logics $\mathcal{ALC}$ [DM00, Don03]. Exptime complexity of subsumption can be a significant burden for the development of a practical discovery approach that can be applied in use cases with large service description repositories. The use of a model checking based service matchmaking approach is considerably less complex. In [EL86] it has been shown that the naive μ-calculus model checking is polynomial in the size of the formula (request specification) and the size of the LTS.

4.2.2 A Formal Model of Service Requests

A service request describes a set of services, which we call desired services. In order to characterize the desired services, a service request specifies constraints on any service property. For each property type, i.e., service behavior, functionality, non-functional properties, an appropriate constraint specification language is used. Regardless of the property type, each property constraint describes a set of desired property values. That is, in contrast to the property-based model of service descriptions (cf. Figure 4.1), the key-value-set structure of a service request features value sets instead of single values. The request model is diagrammed in Figure 4.7 for comparison. Note that the request model features value-sets $V_{R,p} \subseteq V_{t_q}$ for each requested property instance. The description of value-set $V_{R,p}$ is a constraint, which defines the desired property values $v \in V_{R,p}$ of desired services. Of course, the number m of constrained property instances in the request model is independent of the number n of property instances in the description model.

The interpretation $(\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ defines the request semantic. The function $\cdot^{\mathcal{J}}$ assigns to each expression of the request language a set of matching services from the domain $\Delta^{\mathcal{J}}$ of services within a repository. For this purpose, $\cdot^{\mathcal{J}}$ maps the key-value-set structure of a request into sets of desired property instances, i.e., property-value pairs. The latter representation describes a set of desired services, i.e., each desired service is described by a set of desired property-value pairs.

The request is interpreted as a power set $\mathcal{Q} = 2^Q$ of property instances. Each requested property instance $q \in Q'$, with $Q' \in \mathcal{Q}$, also specifies a property type t_q that is assigned to a value $v \in V_{R,t_q}$ of the desired value set V_{R,t_q}. Desired values V_{R,t_q} have to be a subset of the value range V_{t_q}.

Let $q = (p, V_R)$ denote a requested property instance of the user's concern. $V_{R,p} \subseteq V_{t_p}$ is the set of acceptable property values of the property $p = t_q \in P$. Then, the interpretation of a requested property is $(p, V_{R,p})^{\mathcal{J}} = \{(p, v) | v \in V_{R,p}\}$, which is the set of property-value pairs that is constructed by considering each value $v \in V_{R,p}$ that is a member of the set of acceptable values individually.

Our interpretation function $\cdot^{\mathcal{J}}$ of a service request allows to describe a set of desired services. It corresponds to a power set of service instances as we depicted in Figure 4.6, and is in contrast to existing approaches that waive a distinct request formalism.

Example 8. *As depicted in Figure 4.5 earlier, a request may specify that a service should have an average response time of less than 20ms. The property instance* $(\mathsf{responseTimeMS}, \leq_{20})$*, where* $\mathsf{responseTimeMS}$ *represents the property type in* P *and* $\leq_{20}$ *denotes the set* $V_{R,p}$ *of desired values, is part of a request. The interpretation* $(\mathsf{responseTimeMS}, \leq_{20})^{\mathcal{J}}$ *depends on the definition of* V_{t_q}*. Assuming that* V_{t_q} *is the set of non-negative integers, the interpretation of the requested property is as follows:*

$$(\mathsf{responseTimeMS}, \leq_{20})^{\mathcal{J}} = \{(\mathsf{responseTimeMS}, v) \mid 0 \leq v \leq 20 \wedge v \in \mathbb{N}\}.$$

4.2.3 Service Request Language

The ontological representation and its serialization of service requests, as introduced for service descriptions in Section 4.1.2, is not a necessary aspect of our approach. We assume that service requests are posed at run-time to a discovery engine. While service descriptions need to be stored in a repository, the requests are typically not stored and the request representation and serialization becomes less important.

A request is a combination of constraints over several service properties. Individual constraints can be combined by using Boolean operators: conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). The semantics of combined expressions is defined as follows. Assume we have two expressions φ_1, φ_2. Each of the expressions constrains the desired values of the respective service property.

The interpretations $\varphi_1^{\mathcal{J}}, \varphi_2^{\mathcal{J}}$ of the expressions represent sets of services that fulfill constraints φ_1, φ_2, respectively. We gain two sets of matching services $\mathcal{S}_1 := \varphi_1^{\mathcal{J}}$ and $\mathcal{S}_2 := \varphi_2^{\mathcal{J}}$. Both $\mathcal{S}_1 \subseteq \mathcal{S}$ and $\mathcal{S}_2 \subseteq \mathcal{S}$ are subsets of the available services $\mathcal{S}$, e.g., in the service repository.

Combinations of expressions are interpreted as follows.

$$\begin{aligned}(\varphi_1 \wedge \varphi_2)^{\mathcal{J}} &\equiv \varphi_1^{\mathcal{J}} \cap \varphi_2^{\mathcal{J}} = \mathcal{S}_1 \cap \mathcal{S}_2 \\ (\varphi_1 \vee \varphi_2)^{\mathcal{J}} &\equiv \varphi_1^{\mathcal{J}} \cup \varphi_2^{\mathcal{J}} = \mathcal{S}_1 \cup \mathcal{S}_2 \\ (\neg\varphi_1)^{\mathcal{J}} &\equiv \mathcal{S} - \varphi_1^{\mathcal{J}} \quad = \mathcal{S} - \mathcal{S}_1\end{aligned} \tag{4.1}$$

The laws of the Boolean algebra hold and allow evaluating more complex combinations of constraint expressions, too. In the following, we show how constraints over the functional and non-functional service properties can be specified.

Behavioral constraints. We use declarative process expressions to constrain the behavior of desired services. The underlying formalism that combines the temporal logic μ-calculus with a description logic to semantically specify the resources was introduced in the preliminaries in Section 3.3.2. For now, the formalism is applied without any modifications.

Functionality constraints. We introduce the requests for atomic services in Section 5.3. The service request model for atomic services instantiates our request model from Section 4.2.2 and we will introduce request modeling with focus on the functionality property.

Non-functional requirements. As non-functional requirements (NFRs) can be seamlessly combined with functional requirements in a request, it is also possible to demand alternative service configurations. For example, the combination allows specifying that a user is willing to accept a higher price if the convenient credit card payment method is available. Also, more complex alternatives can be described in a request using our formalism. For example, the desired inputs or outputs may depend on other functional or non-functional properties and vice versa.

Constraints over NFP values can be expressed in different ways. Depending on the property types and the semantics of the NFP values, different means to express desired value sets have to be used. For many simple data types, like integer or string, there are utility functions with a defined semantics that can be used to express ranges or sets of values. For example, the operators $\leq, <, \geq$, and $>$ can be used for the definition of ranges of integer values. For other and potentially more complex properties, the specification of value sets requires the use of constraint specification languages for this particular type of values. As these NFRs can be evaluated by external tools, like a policy compliance checker, the structure of NFR expressions is not dictated by us.

Summary. Now that we have introduced our service request formalism and have shown how requests can be expressed in the request languages, we want to recapitulate the benefits of this approach. Our request model that we applied is depicted in the upper layer of Figure 4.5. Requests describe a set of (desired) services, whereas each service description represents a set of service executions. The request model captures functional and non-functional properties uniformly. Each desired property instance is described by a set of desired values, which avoids the above mentioned conceptual mismatch.

The accuracy of search results is guaranteed as the description and request formalisms allow us to apply model checking to verify the requested constraints on service descriptions. The model checking that we introduce in the next section leads to logically correct results [Ran01], which fulfill the requirement R2. The existing discovery approaches, which use a single formalism for descriptions and requests, can also compute accurate results, e.g., if exact and plug-in matches are considered only. However, their formalisms are not as expressive as the one we use for requests, and the matchmaking has to be based on satisfiability checks, which is usually very complex. Model checking is faster than satisfiability-based approaches and thus has better chances of scaling to Web scale.

4.2.4 Modeling Example

A request for a book selling service can be specified as follows. We request that desired services have to deliver a book at some point and should have the option to pay with a credit card. In order to describe the desired behavior Φ, we specify that there has to be an initial action *search*, which is an input action as further specified in Listing 4.5, that receives parameters to search for books.

The proposition P poses constraints on the knowledge in the state after the input action. For instance, P expresses that both input parameters are added to the knowledge base if $P \equiv \mathsf{ex{:}Author(ex{:}author), ex{:}Title(ex{:}title)}$.

```
ex:search a supProc:InputAction ;
    supProc:hasParam ex:author , ex:title .
ex:author a ex:Author .
ex: title a ex: Title .
```

Listing 4.5: Description of requested action *search*

Then, eventually in the process there has to be the action *delivery* that delivers the book. Similar to Listing 4.5, the action *delivery* is described as an output action with one parameter that denotes the book. Furthermore, the proposition Q with constraints on the subsequent state should assert that the resource representing the book actually has author and title as specified in the beginning. We specify Q as follows:

$$\begin{aligned} Q \quad \equiv \quad & \exists \mathsf{ex{:}book} \,.\, \mathsf{ex{:}Book(ex{:}book)}, \\ & \mathsf{ex{:}hasAuthor(ex{:}book, ex{:}author), ex{:}hasTitle(ex{:}book, ex{:}title)} \end{aligned}$$

In a path with such an action *delivery*, there has to exist another action that describes the credit card payment. *payment* is an input action that receives the credit card details, and in the proposition R we can constrain the amount to which the credit card can be charged. Ideally, only the cost of the delivered book should be charged.

Besides the behavioral requirements, non-functional properties of the desired services can be constrained, too. E.g., in order to express that the delivery time of the service should be less than ten days, we can add $(\mathsf{ex{:}deliveryTimeDays}, \leq_{10})$ to the request $\mathcal{R}$. The complete request combines constraints on two property instances:

$$\mathcal{R} := (\mathsf{ex{:}deliveryTimeDays}, \leq_{10}) \wedge (\mathsf{ex{:}behavior}, \Phi), \text{ with}$$
$$\Phi \overset{\mathsf{def}}{=} [search]P \wedge_{\mu} \textbf{eventually } \langle delivery \rangle \Big(Q \wedge_{\mu} \textbf{eventually } \langle payment \rangle R \Big)$$

4.3 Model Checking Based Matchmaking

Matchmaking is the core technique of our service discovery method. It compares the service requests with the available service descriptions from the repository. We use a model checking based technique to verify the individual constraints over available services. The model checking approach evaluates if a given service is a correct model of the service request. Services that fulfill all requested requirements are returned as search results.

4.3.1 Matching Properties

We first investigate how to match properties in general. The subsequent paragraphs discuss the matchmaking of the behavior in detail.

A service description $\mathcal{D}_\omega$ is interpreted as a set Q_ω of property instances comprising functional and non-functional properties in a unifying way:

$$Q_\omega^{\mathcal{J}} \subseteq \bigcup_{P \in \mathcal{P}} P \times V_P \tag{4.2}$$

Within the formal model of service descriptions, the property instances $q \in Q$ model the assignment of a property type t_q to a value $v_{t_q} \in V_{t_q}$.

Constraints on property instances $Q_{\mathcal{R}}$ in a request $\mathcal{R}$ are interpreted as a set of values assigned to a property:

$$Q_{\mathcal{R}} \subseteq \bigcup_{P \in \mathcal{P}} P \times 2^{V_P} \tag{4.3}$$

$$Q_{\mathcal{R}}^{\mathcal{J}} \subseteq \left\{ Q : Q \subseteq \bigcup_{P \in \mathcal{P}} P \times V_P \right\} \tag{4.4}$$

As can be easily derived from Equations 4.2 and 4.4, a set of property instances $Q_\omega^{\mathcal{J}}$ of a service description $\mathcal{D}_\omega$ matches the desired properties $Q_{\mathcal{R}}^{\mathcal{J}}$ if and only if $Q_\omega^{\mathcal{J}} \in Q_{\mathcal{R}}^{\mathcal{J}}$.

A service is a match if there exists a set $Q'_{\mathcal{R}} \in Q_{\mathcal{R}}^{\mathcal{J}}$ of property instances that equals the set $Q_\omega^{\mathcal{J}}$ (see right part of Figure 4.6 for an illustration). Then, there also exists a $q_{\mathcal{R}} \in Q'_{\mathcal{R}}$ for each property instance $q_\omega \in Q_\omega^{\mathcal{J}}$ with $q_{\mathcal{R}} \equiv q_\omega$. This means that the advertised property type is always subsumed by the requested type ($t_{q_\omega} \sqsubseteq t_{q_{\mathcal{R}}}$) and values $v_{q_{\mathcal{R}}} \equiv v_{q_\omega}$ are equivalent. Of course, heterogeneity among property types can be resolved by the use of ontology mapping and alignment techniques.

Example 9. *Given a service ω with an advertised average response time of 8ms. This service property is expressed by $q_\omega = (\mathsf{responseTimeMS}, 8)$ in the service description. A request with a desired response time of less than 20ms is expressed by $q_{\mathcal{R}} = (\mathsf{responseTimeMS}, \leq_{20})$ (as shown in Example 8).*

With respect to this non-functional requirement, the service ω is a match because (i) the advertised property instance $(\mathsf{responseTimeMS}, 8)$ is an element of the interpretation of the requested property

$$\begin{aligned} (\mathsf{responseTimeMS}, 8) &\in (\mathsf{responseTimeMS}, \leq_{20})^{\mathcal{J}}, \textit{with} \\ (\mathsf{responseTimeMS}, \leq_{20})^{\mathcal{J}} &= \{(\mathsf{responseTimeMS}, v) \mid 0 \leq v \leq 20 \wedge v \in \mathbb{N}\}, \end{aligned}$$

and (ii) both property types are equivalent.

If the request $\mathcal{R}$ combines constraints over multiple properties, we apply the semantics of composed constraints as defined in Equation 4.1. That is, each constraint is evaluated individually and the individual result sets are combined according to Equation 4.1. This allows us to compute a final result set of services that fulfill $\mathcal{R}$.

4.3.2 Matching Functionalities

At large, the service functionality is treated like any other property. The description of a service offer specifies its functionality as a service property value. In a request, the set of desired functionalities is described as values of the functionality property. Although the functionality descriptions of offers and in requests are composed of several sub-properties, the matchmaking approach is analogous to the above principle for matching properties. The specifics of the matchmaking of functionalities are presented in Section 5.4 in the context of the discovery of atomic services.

4.3.3 Matching Behaviors

Creating LTS representation of process expressions. The behavior of a service is described by an agent identifier. A labeled transition system is generated from a given definition of an agent identifier $A(x_1{:}\,t_1, \ldots, x_n{:}\,t_n) \overset{\mathsf{def}}{=} \pi$. In such an LTS, a state corresponds to a knowledge base while transitions can be input, output or local actions. An agent identifier A contains a reference to an ontology O_ω that describes the domain terminology (TBox) used in the process expression.

The first state s_0 is generated as follows: Create a new ontology for $KB(s_0)$ and copy all the TBox axioms of O_ω to $KB(s_0)$. Then, add an individual $\mathsf{x_i}$ of the type $\mathsf{t_i}$ for every argument $x_i : t_i$ of A to the ABox of $KB(s_0)$. The object property values and data property values corresponding to the relations among the individuals $x_1, \ldots, x_n$ are added to the ABox of $KB(s_0)$ as well. The rest of the states and transitions are generated according to the execution semantics of the process expression π that defines the behavior of the agent identifier A as follows.

For a sequence $\pi \overset{\mathsf{def}}{=} \tau.Q$, a transition τ is added from the state s_π to a new state s_Q. State s_Q is created from the state s_π by inheriting the knowledge of s_π and applying the changes of τ to the new state. For a parallel composition $\pi \overset{\mathsf{def}}{=} \pi_1 \parallel \ldots \parallel \pi_n$, the LTS creation is performed recursively on the individual component processes $\pi_1, \ldots, \pi_n$ as described above. For a choice $\pi \overset{\mathsf{def}}{=} \pi_1 + \ldots + \pi_n$, the LTS creation for $\pi_1, \ldots, \pi_n$ is invoked recursively as well.

Choosing the right type of model checking approach. The LTS structures can be represented in different ways, which affect the matchmaking's efficiency. In the explicit state approach, the LTS structure is represented extensionally using conventional data structures such as adjacency matrices and linked lists so that each state and transition is enumerated explicitly.

In contrast, in the symbolic approach Boolean expressions denote large LTS implicitly [BCM+92]. Typically, the data structure involved is that of Binary Decision Diagrams (BDDs) [Bry86], which can, in many but not all applications, efficiently manipulate Boolean expressions denoting large sets of states. To a large extent, the distinction between explicit state and symbolic representations is an implementation issue rather than a conceptual one. The naive model checking method was based on an algorithm for fixpoint computation and was implemented using explicit state representation. The subsequent symbolic model checking method uses the same fixpoint computation algorithm, but now represents sets of states implicitly. However, the succinctness of BDD data structures underlying the implementation can make a significant practical difference. BDD-based model checkers have been remarkably effective and useful for debugging and verification of hardware circuits. For reasons not well understood, BDDs are often able to exploit the regularity that is readily apparent even to the human eye in many hardware designs. Because software typically lacks this regularity, BDD-based model checking seems much less helpful for software verification.

In the monolithic approach, the entire structure of an LTS is built and represented at any time in computer memory. While conceptually simple and consistent with standard conventions for judging the complexity of graph algorithms, in practice it may be highly undesirable to keep the entire structure in computer memory, as it might not fit. In contrast, the incremental approach (also referred to as the "on-the-fly" or "online" approach) entails building and storing only small portions of the whole structure at any one time [JJ89].

Complexity. The theoretical complexity of the naive μ-calculus model checking is exponential in the alternation depth d. The alternation depth of a formula is the maximum length of μ/ν alternations in a chain of nested fixpoints. For a constraint formula of size m with alternation depth d and an LTS of size $n = |S|$, model checking has the time complexity $O(m \cdot n^{d+1})$ assuming that the cost of evaluating a proposition (Algorithm 1, line 8) is negligible [EL86]. Formulae of alternation depth higher than 2 are notoriously hard to understand and rarely produced in practice [Bra98]. As argued in [Aga07a], formulae are typically small and alternation depths are typically 1 or 2. Also, a chain of polynomial problems of increasing degree is obtained by stratifying the formulae according to alternation depth.

Evaluation of behavior constraints. The naive model checking algorithm is presented in Algorithm 1. The procedure *evaluateFormula* computes a subset S' of the LTS states S such that all states in S' satisfy the behavior constraint Ψ of a request.

In order to verify the constraints of a discovery request, we employ the verification method introduced in the preliminaries (cf. Section 3.4.2 on page 46). As shown in Algorithm 1, the recursive structure of Ψ is broken down into its atomic constructs that can be evaluated on the LTS. For each atomic construct, a set of states in which the atomic constraint holds is returned. Then, the individual result sets are combined based on the set algebra. For instance, the μ-calculus conjunction ($\wedge_\mu$) is resolved into an intersection of the intermediate result sets (cf. line 14 in Algorithm 1).

Propositions are evaluated over individual states by checking whether the knowledge base implies the proposition. That is, we use an ontology reasoner the check whether the knowledge $KB(s)$ of a state s subsumes a proposition P (i.e., $KB(s) \models P$). We also check for subsumption when we verify the semantic descriptions of actions including, e.g., input parameters and their semantic annotations, the communication channels, and the changes of local actions. We derive a separate knowledge base $KB(a)$ from the description of an action a and the TBox of the domain knowledge. Note that $KB(a)$ does not correspond to the state knowledge bases. Then, we also evaluate constraints φ over an action a by checking the subsumption relationship $KB(a) \models \varphi$.

We obtain the final Boolean answer by checking whether the initial states of the LTS are in S' or not.[4] This means, a service ω matches a requested behavior constraint Ψ if and only if the start state of the LTS L^ω modeling the behavior of ω is in the set of states that satisfy the constraint Ψ.

[4] S' is the set of states of an LTS that match the request and is returned by the model checking algorithm in Algorithm 1.

Algorithm 1: evaluateFormula

```
Require: Formula $\Psi$ and LTS $(S, T, A, \mathcal{V})$
if $\Psi = \textbf{true}$ then
  return $S$
else if $\Psi = \textbf{false}$ then
  return $\emptyset$
else if $\Psi = P$ then
  $S' \leftarrow \emptyset$
  for all $s \in S$ do
    if $evaluateProposition(P, s) = true$ then
      add $s$ to $S'$
    end if
  end for
  return $S'$
else if $\Psi = \Psi_1 \wedge_\mu \Psi_2$ then
  return $evaluateFormula(\Psi_1) \cap evaluateFormula(\Psi_2)$
else if $\Psi = \neg_\mu \Psi_1$ then
  return $S \setminus evaluateFormula(\Psi_1)$
else if $\Psi = \langle a \rangle \Psi_1$ then
  $S' \leftarrow \emptyset$
  for all $(s_1, a', s_2) \in T$ do
    if $a' = a$ and $s_2 \in evaluateFormula(\Psi_1)$ then
      add $s_1$ to $S'$
    end if
  end for
  return $S'$
else if $\Psi = \mu Z.\Psi_1(Z)$ then
  $Z \leftarrow \emptyset$
  repeat
    $Z' \leftarrow Z$
    $Z \leftarrow evaluateFormula(\Psi_1(Z))$
  until $Z' = Z$
  return $Z$
else if $\Psi = Z$ then
  return $\mathcal{V}(Z)$
end if
```

4.4 Implementation and Evaluation

We implemented the naive model checking approach using conventional Java data structures and an OWL reasoner. We conducted our performance evaluation on a set of service description with complex behavior. The behavior expressions were derived from so-called Web scripts that record the end user browsing processes (at the consumer side with the help of a browser plug-in). Browsing processes describe how to interact with a Web-mediated service that provides its functionality to end users via interactive Web pages.

We developed Java APIs for modeling services with their properties and serialized theses service descriptions in form of ontologies (one ontology for each service). A separate API allows modeling the executable behavior of a service. For specifying discovery requests, we developed similar APIs for desired services and processes with desired behavior. We also developed a Web-based graphical interface to create semantic service descriptions and describe their behavior (see Figure 4.8).

For the DL part of service descriptions and requests, we use the OWL API[5] for semantically describing simple service properties and the processes resources. During matchmaking, the HermiT OWL reasoner was used for reasoning about this knowledge.

4.4.1 Service Discovery in WisNetGrid

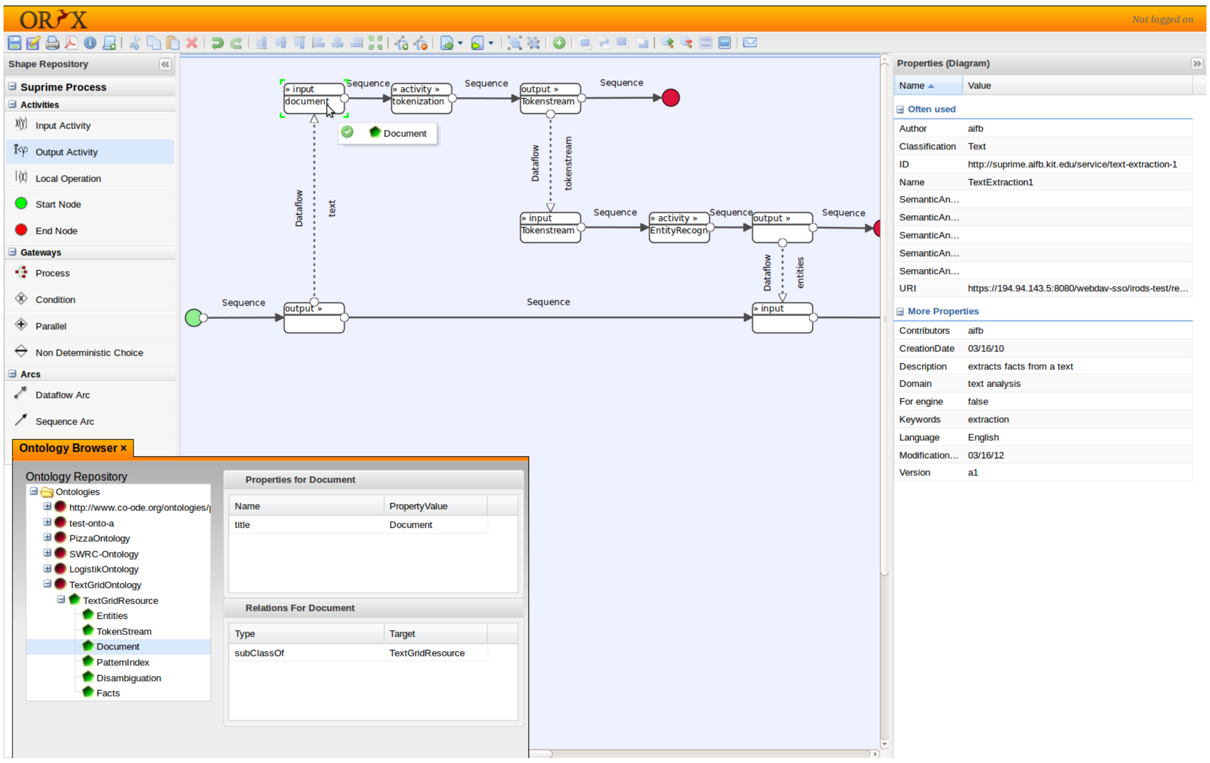
The goal of WisNetGrid was to develop a common knowledge layer on top of Grid resources from various communities. In order to establish the knowledge layer, common services with generic functionalities like information extraction, ontology and service repositories as well as technologies like unified and secured access to different data sources have been provided to the Grid infrastructure.

Our service discovery method presented in this chapter was integrated into the WisNetGrid service repository. The repository provides a REST-based service interface [ECPB13] with functionalities to retrieve and filter service descriptions that reside in the repository. Grid communities can either use the centrally provided repository or host their own instance. The service descriptions are serialized as RDF/S ontologies, as presented in Listing 4.3, and stored in the Grid.

The service descriptions contain the descriptions of the behavior and the non-functional properties. Workflows are treated equally. In addition, we defined a meta data schema to add information about the service description documents. For instance, the creator and the time of creation of the service description, related services, and descriptive keywords can be added [JA11]. These meta data are serialized to RDF and can be queried with SPARQL [W3C13], the RDF query language. The repository allows retrieving service descriptions based on simple meta data constraints. Complex search requests including non-functional and behavioral requirements are also possible.

We also provide a graphical user interface to browse existing services in the WisNetGrid service repository, create new service descriptions, and compose services manually. Elements of *suprimePDL*, our behavior description language, can be dragged to the central canvas to model the behavior. Ontologies can be loaded and used to annotate the process expressions and non-functional properties. Concepts, relationships, and individuals of loaded ontologies

[5]OWL API – http://owlapi.sourceforge.net, retrieved 2013-06-11



Properties for Document

Name	PropertyValue
title	Document

Relations For Document

Type	Target
subClassOf	TextGridResource

Properties (Diagram)

Name	Value
Often used	
Author	aifb
Classification	Text
ID	http://suprime.aifb.kit.edu/service/text-extraction-1
Name	TextExtraction1
SemanticAn...	
SemanticAn...	
SemanticAn...	
SemanticAn...	
SemanticAn...	
URI	https://194.94.143.5:8080/webdav-sso/irods-test/re...
More Properties	
Contributors	aifb
CreationDate	03/16/10
Description	extracts facts from a text
Domain	text analysis
For engine	false
Keywords	extraction
Language	English
Modification...	03/16/12
Version	a1

Figure 4.8: Service and workflow modeling in the WisNetGrid project

are displayed in the lower left widget. Dragging selected elements from the ontology widget and dropping them over the elements of the central modeling canvas or the properties on the right side of the application adds semantic annotations to the behavior description. The annotations can be further refined, e.g., by selecting a specific input parameter of an input action in a separate dialog. The semantic service descriptions can be stored in the repository for later reuse. The graphical user interface was implemented as a Web application. We used the open source version of the Signavio Process Editor.[6] We added the ability to model services in the *suprimePDL* language using graphical elements that can freely be rearranged and connected according to the language syntax. We added the ability to add semantic annotations by dragging existing ontology classes and individuals to elements, e.g., communication actions (to describe parameters) or non-functional properties. We developed an axiom editor within the process editor in order to model more complex ontology axioms.

In Figure 4.8, we depict the modeling area of the Web application. In the shown example, a workflow for the extraction of knowledge from a secondary source is modeled. On the left, the available process language elements are provided. In the lower left corner, ontology classes and individuals are displayed in a tree view. On the right, non-functional service properties and description meta data can be edited. In the center area, the behavior of the workflow is modeled. We used our service discovery method to identify appropriate services that can be integrated into a workflow or bound to the agent invocation elements. We also

[6]Signavio Process Editor – http://signavio.com/products/process-editor/process-modeling, retrieved 2013-06-29

Table 4.2: Behavior characteristics per browsing process

Process Behavior Characteristics	Range
Number of actions	3 ± 1
Number of parameters per I/O action	3 ± 2
DL Axioms per I/O parameter	3 ± 2
Behavior class annotations	3 ± 2

developed an execution engine that executes workflows composed of Web services and RESTful services [JA12]. Required input parameter values are entered by the consumer in a dynamic Web page. The computed outputs are displayed to the consumer.

4.4.2 Test Data

The evaluation of our service discovery approach is based on the above described discovery system. In this section, we describe how we obtained a large set of semantic service behavior descriptions that we could use in our evaluation. In the subsequent section, we report on the conducted experiments measuring the system's performance.

The test data is derived from given descriptions of processes that coordinate existing Web-based services. We use CoScripts from the IBM CoScripter repository.[7] CoScripts describe executable processes. As we show below, the script language is directly mapped to our behavior description language. Our analysis of CoScripts from different domains like travel and real estate[8] resulted in the observation that many end user browsing processes are rather short, comprising a small number of actions performed sequentially. The reason is that most of the existing CoScripts automate interactions with only one website. More precisely, scripted travel processes, e.g., for flight or hotel offers typically involve approximately 3 (mainly input) actions (with about 3 parameters per input action on average). The desired information within the returned Web page was often not explicitly extracted for further processing. So, there is only one parameter per output action on average.

Based on the analysis of the CoScripts, we generated browsing processes with the characteristics summarized in Table 4.2. The correctness with respect to the content of existing CoScripts cannot be guaranteed, as we do not focus on automatic learning of browsing processes. Still, we can argue that the behavior complexity of our test data corresponds to the complexity of the behavior described by the CoScripts. In our experiments we used 2000 generated descriptions. In average, every input/output parameter is specified by three DL Axioms (i.e., at least its data type plus its relationship to other process resources).

[7]The IBM CoScripter repository (http://coscripter.researchlabs.ibm.com/coscripter, retrieved 2012-06-15) provides approximately 5800 scripts for automating Web processes.

[8]We used the repository's keyword-based search facilities, which by no means guarantees the completeness of the search results.

Translation of CoScripts to LTS

In this section, we provide a translation of the CoScripter script language to *suprimePDL*. Other Web automation systems, such as iMacros[9] or Chickenfoot [BM05, BWR+05], can be similarly mapped. A CoScript solely consists of a sequence of instructions. Instructions may contain (i) defined keywords (such as `go to`, `enter`, etc.) for the type of action performed, (ii) a reference to the control of the Web page (e.g., text field, button, etc.), and (iii) values or variable names for input and output parameters. There are many different ways to reference the control in instructions. For instance, XPATH expressions, the HTML element's id or label are a few possibilities that are valid substitutions for `<control>` (angle brackets indicate placeholders).

Concrete values (`<value>`) and variable names (`your <variable>`) can be used interchangeably. A variable name is replaced during script execution with the concrete value from the user's knowledge base. In our translation of a CoScript we build an LTS with a single state s_0 containing the static domain knowledge, only. Then, the LTS is enhanced by each CoScript instruction as follows:

- **Input** From the current state s_i, a state transition (s_i, a, s_{i+1}) is added to the LTS. The transition label a stores the input transition type, semantic description of the input parameter, and the used HTML control. State s_{i+1} is described by the knowledge from s_i extended by the input parameter description. The following CoScript instructions are translated to input actions:
  `enter <value> into the <control> <controltype>`,
  `select <value> from the <control> <controltype>`,
  `you <instruction>`
  The latter command denotes an instruction, which is completely performed by the user.
- **Computational** Pressing a button, specified by `click the <control> button`, triggers a server-side action. As for input actions, a transition (s_i, a, s_{i+1}) is added. State s_{i+1} is described by the knowledge from s_i applying the changes that this action causes. So far, changes are not explicitly described in the scripts. However, it is sufficient for information gathering browsing processes to assume that computational actions can only add information to the subsequent state.
- **Output** Output parameters are extracted from the following two instructions: `clip the <control>` and `put the clipboard into your <variable>`. Analogue to input actions, a transition is added to the LTS and the new state contains the output parameter description.
- **Invocation** The instructions `go to <url>` and `click the <control> link` denote an invocation of another page, which is translated to an agent invocation. The description of the invoked process then produces a separate LTS.
- **Conditional** Control constructs like the if-then-else instruction `if there is a <CONTROL> <CONTROLTYPE> ... else ...` is translated to two additional transitions (s_i, a, s_{i+1}) and (s_i, a, s_{i+2}). The target states are equivalent to s_i except for the

[9]iMacros Web Automation – http://www.iopus.com/imacros/, retrieved 2012-06-29

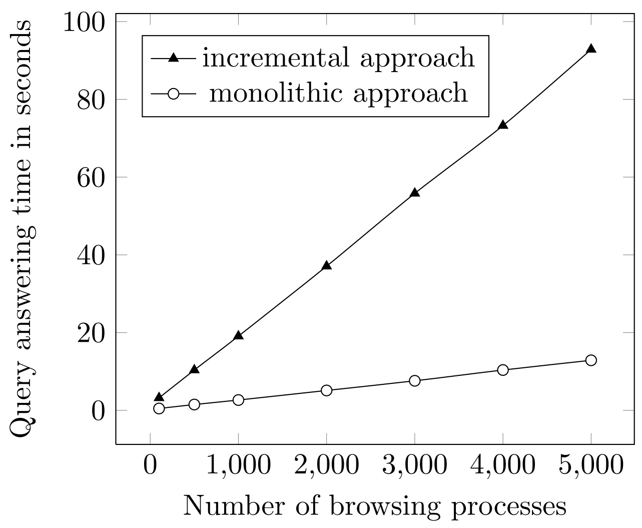


Figure 4.9: Performance of the naive model checking based discovery algorithm (incremental approach vs. monolithic approach)

condition of the if-statement that is added as an assertion to s_{i+1} and as a negated assertion to s_{i+2}.

4.4.3 Performance Results

We created several search queries from 9 simple constraints that were composed by the operators $\wedge_\mu$, $\vee_\mu$, and $\neg_\mu$. A *simple constraint* can be one of **eventually** ϕ, **always** ϕ, where ϕ is a simple constraint like a proposition P or the existence of an action a ($\langle a \rangle P$). Analogously to the descriptions, the complexity of each proposition and parameters of desired actions is set to an average of 3 ± 2 DL axioms (class and object property assertions, e.g., $P \equiv \mathsf{Flight(f)} \wedge \mathsf{Time(t)} \wedge \mathsf{departureTime(f,t)}$).[10] A desired action is expressed by its type (class assertion) and the description of types and relationships between messages.

Figure 4.9 shows our evaluation results for the performance of our naive model checking implementation. In our evaluation setup, the browsing processes contained 3 ± 1 actions each, where each action had 3 ± 2 parameters, and each parameter had 3 ± 2 ontology axioms as semantic annotations. In the first case (incremental approach), the states are loaded or modified on-the-fly. As a result, $\mathcal{ALC}$ reasoning is done during the model checking time and total query answering time increases quickly with increasing number of processes. The satisfiability problem and thus the subsumption problem in $\mathcal{ALC}$ have been shown to be Exptime-complete in [DM00]. In the second case (monolithic approach), we load all the state ontologies offline, which means the expensive $\mathcal{ALC}$ inferencing is done by the $\mathcal{ALC}$ reasoner at loading time before model checking. Thus, we achieve linear time complexity that verifies the theoretical time complexity of model checking (see Section 3.4.2). Even though browsing processes are independent of each other it is hard to parallelize the problem because there may be mappings between the domain ontologies of different browsing processes.

[10]Travel ontology – http://www.w3.org/2000/10/swap/pim/travelTerms.rdf, retrieved 2013-05-01

Even though we achieve linear time complexity, model checking requires 1ms per browsing process. Considering that end users nowadays are used to extremely fast search engines like Google, and that we might have millions of browsing processes in a production environment, the performance achieved in the second case is still not sufficient. Developing a new model checking algorithm that has theoretically better linear time complexity in the size of LTS is out of scope of this thesis. Rather, our aim is to develop indexing techniques that reduce the constant factor in the absolute time required to find matching browsing processes.

4.5 Summary and Conclusions

In this chapter, we introduced our service description and request formalisms, developed a discovery method suitable for these formalisms, and reported on the experimental performance.

The upper model of service descriptions allows for a comprehensive description of services including their functionality, behavior, and non-functional properties. By this, we gain the expressivity to fulfill requirement R3 of our envisioned scenarios.

We developed a corresponding request formalism that furthermore allows for a precise specification of requirements. We thoroughly discussed the reasons for using a dedicated request formalism in favor of a single formalism that is used in both service descriptions and requests. Based on two different formalisms for the description of services and requests, we developed a matchmaking technique that achieves the required accuracy of search results (requirement R2).

We also showed how we use ontologies for the domain and service descriptions in order to overcome the heterogeneity problem, as qualified in requirement R4. We achieved automation as required in R1 by utilizing formal models for descriptions and requests. Based on the formalisms, we developed a model checking based discovery method that automatically identifies correct matches.

The implementation of this discovery method was based on conventional Java data structures to represent services and their behavior. We observed that the discovery results can be delivered faster if the data structures are created and populated offline rather than creating them online and on demand. The development of a more efficient service discovery method will be addressed later in this thesis in order to address the remaining requirement R5.

Before focusing on efficiency, we apply in the next chapter the presented service discovery method to atomic services, a subset of services that recently gained a lot of attention. Atomic services implement a single request-response behavior pattern of message exchange. Due to the limited observable interaction with external agents (e.g., the invoking service consumer), it is possible to simplify the behavioral model to a functionality profile.

Chapter 5

Discovery of Atomic Services

In this chapter, we develop a discovery method that is tailored to the specific characteristics of atomic services. Atomic services gained a lot of attention in the Semantic Web community, for example in [BLHL01, AS04, FFST11], and culminated in several modeling frameworks and respective discovery approaches.

We apply the approach presented in the previous chapter, which supports and focuses on services description with complex behavior. However, the provider may not always want to reveal the entire behavior description. The Semantic Web Services profile is an abstraction to an interface description. In this chapter, we focus on including this profile in our description, request, and discovery method. We maintain the presented approach (including service descriptions, description of requests, and the matchmaking method) with respect to the upper model and the non-functional properties. These parts are treated equally for complex and atomic services and we will refrain from repeating them.

Atomic services all feature the same behavior characteristics. This fact allows for the abstraction of the behavior descriptions in favor of a more compact profile called service functionality. This profile description can be created and used for any kind of service, e.g., to hide internal implementation details of the service. However, the functionality is not sufficient to describe executable services that implement a complex, i.e., multi-stage interaction between consumer and provider. In the case of atomic services, which implement a single request-response-pattern, the functionality profile is sufficient to execute the described behavior. Therefore, we shift our attention in this chapter to the functionality profile that plays a central role in the description and discovery of atomic services, like traditional Web services and Web APIs.

Their behavior is characterized by only two means of observable interactions with the consumer. With the service invocation, input parameters can be passed to the service, and after successful service execution, the service returns output parameters back to the consumer who invoked the service. This restriction in the interaction pattern of atomic services allows abstracting from complex behavior descriptions, as the information about the internal behavior of a service is no longer required for using and interacting with the service. The internal behavior cannot be observed externally. Consequently, a profile of the behavior is enough to reason over the functionality offered by a service.

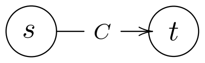


Figure 5.1: State changes caused by a program C expressed by a Hoare triple $\{P\}C\{Q\}$ with $s \models \{P\}$ and $t \models \{Q\}$

We develop a discovery method that matches functionality descriptions of services with the constraints expressed in a search request. We developed and presented our request model and the corresponding discovery method for functionality descriptions in [JAS10]. In [JA10], we extended this approach by adding the property-based service model in order to capture non-functional properties in addition to the functionality. We evaluated and applied this discovery method in the context of the SOA4All project [AJN10a]. It has been further applied to solve the Web service discovery problem in the InterLogGrid [HA12] and MeRegioMobil [WJH11] project.

Many approaches for the semantic description of atomic services and the automated Semantic Web Service discovery have been proposed in recent years. So, we start this chapter with an overview of the state of the art in the area of service discovery in Section 5.1. In Section 5.2, we introduce the service functionality description that extends our property-based service model introduced in the previous chapter. We also clarify how the state changes are modeled in functionality descriptions. Then, we provide a request language for the specification of the desired functionality in Section 5.3 and develop in Section 5.4 a matchmaking method for the functionality property. Finally, we present our implementation and evaluation results of the discovery method for Semantic Web Service profiles in Section 5.5.

5.1 State of the Art

In Section 5.1.1, we have given an overview on the modeling of service functionality. Existing discovery approaches for atomic services are summarized in Section 5.1.2. In both sections, we will only briefly mention the limitations of these approaches. In Section 5.1.3, we explain the limitations and drawbacks of state of the art solutions within our context.

5.1.1 Modeling Functionalities

Software Artifacts

The Floyd-Hoare logic is a formal system developed by Tony Hoare [Hoa69] and is based on a formal system for reasoning over flowcharts [Flo67]. The purpose of the Floyd-Hoare logic is to reason over the correctness of software programs. Hoare proposed to use predicate logic assertions to model preconditions P and postconditions Q of a piece of software C expressed in form of Hoare triples $\{P\}C\{Q\}$. As illustrated in Figure 5.1, each triple models how a program changes the state of the computation. That is, P holds in the state before the execution of C and Q holds afterwards if and only if the execution of C terminates. With the axioms and inference rules that the Floyd-Hoare logic provides, constructs of imperative programming languages are covered by the system.

Design by contract [Mey92] is a software design approach that aims at a formal specification of software interfaces. Based on the Floyd-Hoare logic, the contracts comprise a description of preconditions, postconditions, side effects, and invariants. Many programming languages like Eiffel [Mey91] and Vala[1] have native support for the design by contract principle, and consequently the specification of conditions and effects is programming language specific.

The description [Hoa69] and discovery [ZW97] of software artifacts can be seen as the predecessor of current Semantic Web Service discovery approaches. However, the closed world assumption does not completely hold for Web-based services and systems because the ability to model side effects to the world and the notion of background knowledge were not considered in the specification of software artifacts.

Service Descriptions with Universal Description, Discovery and Integration

Universal Description, Discovery, and Integration (UDDI) [UDD01, CHvRR04] was the first attempt to provide users with a system for finding Web services. UDDI proposes an XML-based standardized description format for service interfaces described with the Web Service Description Language (WSDL) and a platform-independent central registry for service descriptions. It provides means to describe services that are offered and shared. Based on the registered descriptions, other actors can explore, find, and utilize new business partners and their services. For that purpose, UDDI registries consist of White, Yellow, and Green Pages. White Pages contain information about the provider, such as the address, the business name, or contact information. Yellow Pages administer a service classification and technical details like the service binding are filed in the Green Pages.

Although the description format is standardized by UDDI and a standardized service classification schema can be applied, the remaining syntactic descriptions impede a high degree of automation in discovering services from UDDI registries. The support for the use of heterogeneous terminologies and a machine-interpretable description of the service functionality (beyond a service classification) is lacking.

Semantic Web Services

Since the advent of the Semantic Web [BLHL01], many Semantic Web Service approaches for the description, discovery, ranking, selection, composition, and execution of services have been proposed. The use of semantic service descriptions aims at dealing with the heterogeneity in the terminology used in different service descriptions, e.g., by different providers. Many of these approaches focus on a logic-based description of the service functionality, which allows for automated tasks like service discovery and composition.

As elaborated in Section 3.3.1, the Semantic Web community with focus on languages developed several service description approaches that use description logics to describe service profiles [BHL$^+$05, GCTB01, SPAS03]. The objects involved in the service execution are represented as concepts in DL. As done for example in [PS03, LH03], the use of DL is combined with the DARPA Agent Markup Language for Services DAML-S, the predecessor of OWL-S, and the Ontology Inference Layer DAML+OIL. These approaches use description logics to represent the knowledge about a service, e.g., describing input and output parameters,

[1]Vala programming language – http://live.gnome.org/Vala, retrieved 2013-06-11

preconditions and effects. They failed to reason about the dynamics of Web services since DL reasoners cannot reason about changing knowledge bases. As we will show below, it requires additional effort to model the dynamics of services. It requires to model knowledge base changes if the execution of services can cause effects.

The approach in [HZB$^+$06] is limited to Web services that do not change the world. Only inputs, outputs, and the relationships between them are described. A description of the offered service functionality can therefore be described with a DL query. As conditions and effects are not considered by this approach, it is limited to the description of simple information retrieval services.

More expressive service descriptions are the prerequisite for the development of automated search methods and the use of discovered services if services that may cause changes are to be supported, too. An effective service discovery method has to identify services that fulfill the requirements of a search request. Compared to service discovery methods that are restricted to input and output matching, the additional matching of conditions and effects of services leads to a higher accuracy of discovery results, which is mandatory for any automated use of the discovered services and aimed at by our approach as stated in requirement R2.

Functionality-based semantic service descriptions of atomic services can be regarded as the minimal information required for automated tasks like service composition. The common model to describe the functionality of services consists of inputs, outputs, preconditions, and effects, shortly denoted by IOPE. Inputs describe the set of user-provided message parts at service invocation. Outputs describe the set of values returned to the invoker after a successful service execution. While preconditions describe the information state of the service provider before service execution, the effects list the changes to the information state that are caused by the service execution. Some discovery approaches however use this description element to describe the state after execution, i.e., the effects then comprise a description of the postconditions. Both, preconditions and effects are expressed in logic formulae.

Note that the common approach to describe services by IOPE differs from the formal description model for software artifacts. In IOPE-based service descriptions, effects are described instead of postconditions. While effects describe the set of changes introduced by a service, they do not suffice to describe the information state after the service execution. On the other hand, the specification of postconditions with description logics is also insufficient to describe changes of services in general. We will elaborate on this distinction in Section 5.2.

The Unified Service Description Language (USDL) is another service modeling approach that can be considered as a container description format for comprehensive service descriptions [CBMK10]. In comparison to the aforementioned approaches, business related information like pricing and legal details are emphasized. USDL is a normative schema formalized in UML (Unified Modeling Language) for services descriptions that aims at reduced integration costs on a business level. Along with business related information, USDL captures operational and technical information, too. The latter part is designed for IT services only. The operational perspective describes capabilities and dependencies, which can be used to describe compositions. USDL can be extended with additional components. In early research efforts, USDL service discovery is limited to a matchmaking based on functional classifications (e.g., using UNSPSC[2]) or natural language based descriptions [CVW08]. One reason for this choice

[2]United Nations Standard Products and Services Code – http://unspsc.org, retrieved 2013-06-11

is that USDL does not provide any concrete formalism to model the behavior or functionality.

The next evolutionary step of the language is called Linked USDL.[3] As the name suggests, Linked USDL builds upon Linked Data principles and reuses existing vocabularies like the minimal service model (MSM).[4] While the Linked USDL effort mainly remodels USDL as an RDF/S vocabulary, the introduction of MSM requires the use of WSMO-Lite service functionality descriptions. Analogue to USDL, Linked USDL allows for a comprehensive description of perspectives beyond the service functionality.

Modeling State Changes

A general problem that arises with ontology-based semantic service modeling approaches is that description logics cannot capture the dynamics of services. The execution of a service typically changes the knowledge of the service provider. Changes can comprise the creation, an update, or the deletion of a database entry (or any other storage technology). If third party providers are involved in the provision of the service, their knowledge bases can be changed by the service execution, too.

Example 10. *In order to review the importance of changes and their impact on modeling the offered functionality in service descriptions, we list the changes for a simple Web service. Continuing our example of a service that offers the functionality to buy books electronically (cf. the service description in Section 4.1.3) a successful service execution causes the following changes and effects. The book is shipped to the user's home address after her credit card was charged. From the provider's perspective, the amount of available books is reduced, an order and an invoice are created, and the account balance is increased as a payment is expected.*

Using the state-based model of service executions, the change of knowledge of the providers implies that the original state is changed. A corresponding state transition leads to the subsequent state with the updated knowledge base. Representing the knowledge, e.g., with description logics, allows to model only one state. It leads to the problem that the representation of knowledge base changes (state changes) in Semantic Web Service modeling needs to be treated additionally.

The state-based interpretation of service functionality profile descriptions features a clear state separation. The descriptions of the two states should not be represented in the same knowledge base, because this would merge the knowledge of both states into a single knowledge base. This can lead to logic inconsistencies and contradictions. For example, merging the information that is user is initially not subscribed to a service and that the same user is subscribed to a service afterward is inconsistent when both statements are part of the same knowledge base. In order to establish a relation between the states and the state knowledge, WSMO introduces shared variables that are universally quantified over WSMO assumptions, preconditions, postconditions, and effects in order to relate preconditions and assumptions with postconditions and effects [dBBD+05].

Action formalisms like the situation calculus [McC02] and the fluent calculus [Thi98] allow to model and reason over actions and the state changes caused by these actions. In the scope of semantic service modeling, the action formalisms have the advantage that they solve the

[3]Linked USDL – http://www.linked-usdl.org, retrieved 2013-09-23
[4]MSM – http://iserve.kmi.open.ac.uk/wiki/index.php/Simple_vocabulary, retrieved 2013-06-11

frame problem [MH69]. The basic idea is to explicitly describe all updates (changes of fluents) for each successor state in addition to the knowledge of each state. Most formalisms define their own agreements on how to include or exclude state updates explicitly. Consequently, these action formalisms allow for reasoning over actions and their effects. Recent semantic service modeling approaches, formalisms, and frameworks focused on the application of FOL or its fragments like DL for the description of the states. Consequently, it is hard to model actions and their effects of atomic services correctly, when building upon existing standards or approaches. In contrast, action formalisms are widely used for, e.g., the automated composition of Semantic Web Services [SW04, CSHG09, BFM11] and the behavior description [NM02].

5.1.2 Discovery of Atomic Services

After we discussed the state of the art with respect to service descriptions, we now present an overview on service discovery approaches.

Matching Software Artifacts

In the field of software artifact specification and verification, discovery of software components is motivated by their reuse and the ability to substitute them. Components are retrieved from component libraries based on their functionality descriptions. Matching the descriptions of two components allows computing their relationship, which can indicate the substitutability of one component by another or the subtyping as used in object-oriented programming. Different degrees of matches are useful for different tasks. E.g., different matching types based on the implication relations between preconditions and postconditions are considered in [ZW95, ZW97]. The most restrictive matching type is the exact match with logically equivalent pre- and postconditions. Plug-in matches, and more relaxed flavors of the plug-in match, are defined for the comparison of either two components or one component and a request. The latter comparison is applicable to retrieve software components from a library.

The plug-in match increases the recall of the result set while the precision decreases simultaneously. This can be acceptable in the case of computing potential substitutes of software components, because substitutes do not have to provide the exact same functionality. It is sufficient that the substituting component provides the same functionality in the context of the intended software system. The subsume match, where the software component offers less functionality as requested and the software system needs to be adapted in order to integrate the identified component, was also considered for matchmaking of software components, e.g. in [KDJ06].

UDDI Matchmaking

UDDI is the most prominent example for service discovery based on rather syntactical service descriptions. As already indicated above, missing semantics requires a fairly high amount of manual interventions for finding the right services, mainly due to its lack of support for use of heterogeneous terminologies and the lack of formal description of the functionality of Web services in its underlying model. Even though the API provided by UDDI [UDD01] allows random searching for businesses, it does not allow for the selection of new business partners dynamically and automatically by a program. Furthermore, UDDI allows only keyword-based syntactic search, which is a problem, e.g., when the requester and the provider use different terminology for describing the same concept or use a common terminology for describing different things.

Logic-based Matchmaking of Semantic Web Service Profiles

In recent years, many so called Semantic Web Service discovery approaches have been introduced in order to overcome the drawbacks of UDDI as well as to cope with the ever increasing need for more automation. Unfortunately, existing SWS discovery approaches still cannot address the professional needs and are not easily comprehensible for the potential users.

Functionality-based semantic service discovery is the enabler for many automated tasks in service-oriented systems and architectures. For instance, service composition highly depends on efficient and effective service discovery methods in order to locate additional or substituting services and integrate them successfully into larger service networks or compositions automatically.

Stateless services. In contrast to the state-based modeling and discovery approaches, inputs and outputs of stateless services can be modeled by description logic concepts in service descriptions and requests. Then, matching a request against services can be as simple as DL subsumption checks of the input and output types [GCTB01, PKPS02, BHL$^+$05, HZB$^+$06]. Along this line, Linked Data services [SH11] in its current state of development are similar to the above approaches as only inputs, outputs, and their relationships are described applying Linked Data principles [BHBL09].

Stateful services and DL. More recent research activities concentrate on more detailed formalisms like the state-based perspective on services. It allows separating the information of distinct states. These models aim at modeling the dynamics of services [PKPS02, LH03, KFS06, KK06]. Consequently, the service descriptions that are used for discovery are often similarly structured, such as IOPE service profiles. Since most of the approaches use the same formalism for service requests, too, matchmaking determines various degrees of the intersection of service offer and request and is reduced to checking subsumption of input and output types.

OWL-S OWL-S Matchmaker [SPS04] uses OWL-S Profile [SPAS03] for describing Web service offers as well as requests [PKPS02, MPM$^+$04]. Even though OWL-S Profile has elements for preconditions and effects, the OWL-S matchmaker [SPS04] uses types of input and output parameters only. The approach presented in [LH03] models Web services as well as requests as description logic (DAML+OIL) classes and bases the matchmaking on the intersection of service offer and request, which is computed by a DL reasoner. These particular approaches are only applicable to stateless services, as DL reasoners cannot reason about state changes.

The discovery approach presented in [MPM$^+$04] interprets preconditions as a description element, which only contains constraints on the state of the requester, but not on the provider's state. Also, effects are logical formulae describing the end state, while OWL-S effects were interpreted as side effects of the service execution on the world. The given perspective supports discovery use cases in which users may search for services based on the requirements that they have to fulfill. However, the described preconditions cannot be evaluated by the service provider and, thus, have the purpose to inform consumers.

WSMO Likewise, several service discovery methods based on WSMO service descriptions have been proposed. Expressing service requests with WSMO Goals differs from OWL-S-based discovery conceptually. On the one hand, different formalisms are used for service descriptions and requests and, on the other hand, requests are goal-oriented because they only constrain the state after the service execution.

Goal-driven approaches like [LCC06, LCC08, SHF11] do not explicitly specify inputs as parts of the goal. However, a goal needs to be mapped to a request in order to find appropriate Web services. In such a request, constraints on inputs can be useful, in particular if a user wishes to exclude a particular input parameter. Inputs are also essential for a formal description of the desired functionality (as opposed to a description of the desired end state).

An efficient (and logic-based) semantic discovery approach that can deal with the functionality of Web services is presented in [VKVF08, SHF11]. Although the approach uses an abstract state space as the underlying formal model of service descriptions [KLS06], the discovery algorithm relies on the assumption that the precondition ϕ logically implies the effect ψ of a Web service execution. However, modeling a service execution as a logical implication is problematic with respect to the state changes. Logical implications can lead to inconsistencies if, for instance, the effect specifies the opposite of what is specified in the precondition. Example 11 on page 89 shows a concrete example in which the implication $\phi \Rightarrow \psi$ introduces a contradiction.

Matching Non-Functional Properties

We now turn our attention to the non-functional properties and their role in service discovery. The previously mentioned existing discovery approaches did not consider them so far. In general, non-functional properties are often treated separately and are not part of logic basic service matchmakers that are limited to the capability of state-based functionality matching.

In OWL-S, NFPs contain unstructured metadata such as the name of a service or the provider offering ancillary information to humans. O'Sullivan compiled a list of domain-independent non-functional properties relevant for Web services and categorized them according to availability (both temporal and spatial), payment, price, discounts, obligations, rights, penalties, trust, security, and quality [O'S06]. This approach is still limited to keywords. In order to enable automation of tasks like service selection, the predefined non-functional properties were partly formalized in a WSMO deliverable [TF06].

The Web Service Modeling Language WSML is used to specify the Web Service Modeling Ontology WSMO. WSML does not include non-functional properties into the logical model. Consequently, the ontology reasoners of the WSML2Reasoner framework [WP10] did not support reasoning on them. Nevertheless, the WSMO specification equips all the WSMO elements with non-functional properties, e.g., in order to enrich service and goal descriptions. There is so far no prominent discovery implementation available that considers non-functional properties.

Service discovery based on non-functional properties, often referred to as quality of service (QoS) based service selection, is mainly influenced by ontology- or constraint programming-driven approaches. While the use of ontologies for modeling QoS attributes and their instances in service descriptions delivers accurate discovery results, the approaches based on constraint programming typically compute results faster.

If hundreds of services offer equivalent functionality, the sole use of functionality matchmaking for service discovery is not adequate [KP07]. This problem is solved by selecting services based on quality of service (QoS) attributes. According to their study [KP07], their semantic QoS model [KP06] was the first one reported to fulfill the requirements of semantic service description and discovery. In [KP09, KP12] they apply mixed-integer programming as a matchmaking technique in order to solve complex problems (complex constraints over non-functional properties) for a large set of service offers efficiently. As we do in our property-based service model, the desired property values has been expressed by ranges, while earlier ontology-based approaches like [FFH$^+$03] used the key-value-structure in offers and requests.

The trust in the description of non-functional properties of service offers has been neglected so far. Current approaches for describing non-functional properties abstract from the issuer. To continue our book shop example, every book shop will probably advertise that it has great atmosphere and very friendly and knowledgeable salespeople, etc. If only service providers describe their services, a description of the credentials of the services will hardly be of any practical use from the user's perspective. Yet, this thesis will focus on the development of an efficient discovery method for services and their properties in general. At this stage, we can only acknowledge and reinforce the need for techniques in which parties different from the providers issue credentials to services, and users can build trust in services on the basis of such credentials.

5.1.3 Limitations of Existing Service Discovery Approaches

After a broad overview on semantic service modeling and discovery, we continue with a discussion about the problems and limitation of existing works with a focus on our requirements and targeted scenarios. In this section, we focus on explaining the problems that we identified as relevant obstacles and that are tackled in our work. Of course there are many more open questions and challenges that need to be solved for an envisioned Internet of Services or an Internet of Things with a high degree of automation.

A common problem of many of the existing service discovery approaches is that they apply the same formalism for describing service offers and service requests. We briefly summarize in the following paragraphs why this can be problematic in the context of atomic services for attaining the requirements from Section 2.2 of our approach. Afterward, we will discuss further problems of previous works that we will tackle in our approach.

Common formalism for offers and requests. We observed that a single formalism is often used to model and describe both the offered services and the service requests. We already thoroughly motivated the need for a dedicated request formalism in Section 4.2.1. The same line of argumentation is valid for atomic services and the service functionality.

While intersection-based matchmaking (as in [PKPS02, LH03, CBF05, SHF11]) offers more flexibility, it also requires further effort to evaluate matching Web services in order to assess the applicability for the desired tasks at hand. Consequently, the freedom provided by the different matching degrees is not practicable for the purpose of automated service invocation. Another drawback is the lack of support for the exclusion of certain properties in intersection-based matchmaking approaches.

Support for state-changing services. We want to develop a discovery method for atomic services that reasons on the functionality profile description and additionally interprets the changes such services can cause. Both service descriptions and requests therefore have to capture the dynamic nature of services.

However, the existing semantic discovery approaches do not support the modeling of the changes caused by a service execution to the extent that we aim at, nor do they support specifying desired and undesired changes in requests [SPAS03, LH03, SHH07]. Existing approaches do not clearly specify how preconditions, postconditions, and effects should be used and interpreted. E.g., it is not specified whether a client's state information is included in OWL-S preconditions or not. Also, some approaches used the effects to include a description of the state after execution (which actually is the intention behind postconditions). This lack of specification makes it hard to develop automated search techniques that are comprehensible for developers and users. Some approaches even use the terms "postconditions" for "effects" or vice versa, leading to even bigger confusion. However, a commonality in all existing approaches is that preconditions, effects, and postconditions are modeled as logical formulae.

Regardless of the mixed-up nomenclature, the logical expression representing a postcondition describes the final state of the service. However, using only postconditions to describe the final state of the service execution leads to the well-known *frame problem* [MH69]. The frame problem states that in order to enable correct deductive reasoning about the functionality of software operations, preconditions and postconditions of an operation should not only contain information about which changes an operation causes, but also about which changes an operation does not cause. This requirement makes it impossible to use only preconditions and postconditions for specifying the functionality of atomic services. Note the even higher complexity in the case of atomic services compared to software operations, as the knowledge about the complete set of all services in the Web cannot be assumed.

Effects of a service execution can hardly be expressed in the same way as postconditions. In some cases, effects can partially describe the final state of the execution by applying the effects on the start state description. However, logic formulae are not the proper formalism to express effects in general.

Example 11. *Consider a Web service that removes a user from a mailing list. The precondition ϕ of the service could be expressed by the following axioms.*

$$\phi \equiv \mathit{User}(u) \wedge \mathit{MailingList}(l) \wedge \mathit{memberOf}(u, l)$$

The following effect ψ states that the user is not subscribed to the mailing list anymore.

$$\psi \equiv \neg \mathit{memberOf}(u, l)$$

Simply adding the effect ψ to the start state knowledge expressed by the precondition ϕ will lead to an inconsistent description of the final state.

Thus, applying the effect on the start state description is not a proper way. In order to allow a reasoner to construct correct models, the effects need to be specified differently. An additional specification of the treatment of effect formulae becomes necessary, e.g., by declaring that the information from a previous state can be assumed to be unchanged (cf. fluents in the situation calculus for example).

Non-functional properties. Service discovery based on a unified model for the description and matchmaking of functional as well as non-functional properties has not been developed so far. Individual approaches for service matchmaking based on the functionality, and service ranking based on non-functional properties can be combined into a single search framework. However, filtering services based on non-functional constraints, as it is done with functional constraints, cannot be achieved if NFPs are only used to compute a ranking. The reason is that the service ranking methods compute an order of services based on preferences over NFPs. The formalisms to express preferences are not appropriate for expressing non-functional constraints. Preferences state which properties are considered as more appropriate and allow for ordering the services. However, these preference formalisms do not provide the required expressivity to state which properties are not desired. Consequently, requests based on preferences do not allow identifying accurate discovery results. However, augmenting functional with non-functional constraints will allow us to precisely describe the requirements in requests and the retrieve appropriate search results (see the discussion in Section 4.1.1).

By an incorporation of the non-functional properties into the service discovery process, consumers are able to specify more expressive requests, e.g., to restrict the search results to services with high availability. Furthermore, an early incorporation of the evaluation of non-functional requirements can reduce the search space for further more expensive matchmaking operations like the verification of complex pre- and postconditions or the computation of a service ranking.

Feasibility and performance. Aside from issues related to the expressivity and the underspecification of existing approaches that we already discussed, a practical service discovery method should be easily implementable. Additionally, in order to be advantageous and usable, a service discovery system shall exhibit an adequate performance for most common use cases.

In existing approaches, preconditions and effects are modeled as formulae. Such a formula specifies the knowledge of a state declaratively. However, for building a repository of service descriptions that can be directly queried, a declarative specification of states is not very practical. One reason for this is that formulae are hard (if not impossible) to connect with the services that are modeled as instances in the repository.

Specifically, matchmaking is implemented as a subsumption check computed by description logic reasoners (except for the query containment check that was used for state-less services [HZB$^+$06], which is NP-hard and decidable). Subsumption checking has theoretically been proven to be very expensive. The worst case complexity of OWL-DL concept satisfiability and concept subsumption lie within the complexity class NExptime, whereas other techniques like model checking exhibit better performance. In [EL86, ZSS94], it has been shown that model checking with negligible costs for the evaluation of propositions is P-complete (cf. end of Section 4.3.3).

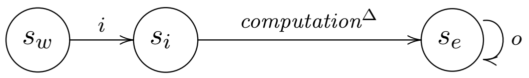
5.2 Semantic Modeling of Atomic Services

For the description of atomic services, we adhere to the property-based service model from Section 4.1.1 as our upper model. We waive the description of the service behavior and add the functionality property to the service description instead.

5.2.1 A Formal Model of Functionalities

In this section we first introduce our formal model of atomic services and then present the formalism to describe service functionality descriptions. The functionality of an atomic service is described by a set of inputs, a set of outputs, a description of the start and end state, and a description of the changes caused by the execution. We consider atomic services that accept user inputs at service invocation time and provide outputs at the end solely. Because there are no further user interactions amid the execution permitted, the service functionalities can be described by the states before and after execution without stating anything about the intermediate behavior and states.

Service functionality property. In Figure 5.2, we show the reduced model of the behavior, which constitutes the functionality of a service profile description. The functionality is modeled as a simplified labeled transition system. The states of the service execution still represent the information state KB_α of the agent α providing the service. The first transition $s_w \rightarrow^i s_i$ denotes the reception of the input parameters. Then, the execution of the services is modeled by a single operation $computation^\Delta$ representing the service computation as a black box. It may cause changes Δ to the knowledge base KB_α of the service provider. Consequently, the changes by the execution of the operation *computation*, which is a local action in our original π-calculus-based behavior model, are encoded into the respective transition label of the LTS. After the computation, a final transition $s_e \rightarrow^o s_e$ represents the output of the results, which does not change the provider knowledge.



$$\begin{tikzcd} s_w \arrow[r, "i"] & s_i \arrow[r, "computation^\Delta"] & s_e \arrow[loop right, "o"] \end{tikzcd}$$

Figure 5.2: Formal functionality model of atomic services

The service functionality is often explained by its purpose to describe what a service does. The functionality represents a profile of the behavior description. It typically contains the description of input and output parameters as well as a description of preconditions and effects (IOPE).

Based on our formal model of service behavior from above, we introduce a profile describing the functionality $(I, O, \phi, \psi, \Delta)$ of atomic state changing services. We describe in the following how each element of the profile can be applied.

Input Parameters I The set of input parameters that is required for (successfully) invoking and executing the service. A set of variable names specifies the input parameters I.

Start State Description ϕ After a service has received the invocation (input) parameters, it is transitioned to the state s_i, from which the actual execution of the service starts. This state contains:

- Individuals describing the initial knowledge of the service before the input action at s_w. Again, we assume that the terminological knowledge of the domain of discourse does not change during the execution and is equal in every state.
- Individuals that represent the input variables. Note that even though the inputs I are concrete values that are unknown at design time, we can differentiate inputs from other individuals as the variable names of I mark them as input parameters.
- Formulae (i.e., logic expressions) that are derived as the overall condition from the conditions $\omega_1, \ldots, \omega_n$ and actions $f_1, \ldots, f_n$ (from the extensive state-based model of a service execution as depicted in Figure 4.2).

Output Parameters O Analogue to the input parameter description, the output parameters of a service are described as a set of individuals, which are sent to the consumer after the service execution finished.

End State Description ψ After a service has performed all the operations (computing the outputs and performing changes), the service reaches the end state s_e.

ψ describes end state s_e that is obtained after performing the operations $f_1, \ldots, f_n$ to the state s_i and the subsequent internal (not observable) states $s_1, \ldots, s_{n-1}$.

Changes Δ The changes caused by the service execution, i.e., updates of the provider's knowledge base, are added to the transition label of the local action representing the service computation.

5.2.2 Description Language

We now turn our attention the concrete description language and the ontological modeling of the service functionality. Note, that functionality is just another property in our model of services. That is, there is a property supSvc:hasFunctionalProperty with range concept supSvc:FunctionalProperty. The vocabulary necessary to describe the service functionality is inherited from WSMO-Lite [VKVF08], SAWSDL[5] [KVBF07], and POSM.[6] In addition, a domain ontology O_ω that provides us with the concepts to model service resources in a description is assumed to be given.

In order to describe a service ω semantically by means of an ontology $O_{\mathcal{D}_\omega}$, the complex property that describes the service functionality is specified by the following sub-properties.

Input Parameters: Ontology instances that are member of the class posm:Message. The instance that represents the service itself is related to the input parameters by the posm:hasInputMessage ontology property.

Output Parameters: Ontology instances that are member of the class posm:Message. The service instance is related to them by the posm:hasOutputMessage ontology property.

[5]Semantic Annotations for WSDL – http://www.w3.org/2002/ws/sawsdl, retrieved 2013-08-15

[6]The Procedure-Oriented Service Model (POSM) ontology is a lightweight approach to the structural description of procedure-oriented Web services, compatible with WSMO-Lite annotation. See http://www.wsmo.org/ns/posm/0.1 for the specification, retrieved 2013-08-15.

Start State Description: The state represents the knowledge base KB_α from the perspective of a provider α. It is described by a set of ontology axioms that specify the instances that represent input parameters and other resources of this state. In $O_{\mathcal{D}_\omega}$, the instances of the ontology concept wsl:Condition represent the descriptions of a start state. The service instance is related to them by the sawsdl:modelReference ontology property. The concept wsl:Condition originates from the WSMO-Lite ontology used for preconditions of services. The property sawsdl:modelReference is defined in SAWSDL and also equally used in WSMO-Lite service descriptions.

End State Description: This state is described by a set of ontology axioms that may refer to the instances that represent input and output parameters. Instances of the WSMO-Lite ontology concept wsl:Effect represent descriptions of the end state. The service instance is related to them by the sawsdl:modelReference ontology property.

Changes: Each change is described by its type (using the concepts AddChange, UpdateChange, and DeleteChange from our process modeling ontology) and an ontology axiom that is linked to by the property hasChange. The axiom describes results from the change and holds in the subsequent end state.

The input (analogously output) parameters are modeled as instances of the ontology concept describing service inputs and the concepts describing the types of the parameters. Latter concepts are again assumed to be defined in the domain ontology O_ω. The start and end states are described by logical formulae, i.e., statements about the input and output parameters, their relationships to each other and to other individuals. Individuals different from the input and output parameters can be additionally added and used to describe the offered functionality. We denote changes by a logical formula and its change type.

These above properties and concepts assemble the functionality of atomic services. Marking the instances as input or output parameters allows us to distinguish them from other resources that are involved in the provision of the service and already exist in the provider's knowledge bases.

Modeling changes. Causing changes is an essential feature of useful services, e.g., by creating a new book order. We need to capture the dynamic nature of services in both service descriptions and requests. Existing Semantic Web Service discovery approaches provided limited support to model and reason on the changes caused by a service execution.

Existing approaches do not clearly specify how preconditions, effects (or sometimes referred to as postconditions) should be interpreted. This makes it very hard to develop automatic search techniques that are comprehensible for developers and users. A commonality in all existing approaches is that preconditions, effects (and also postconditions) are expressed by formulae in a logic like DL or FOL. However, some approaches even use the terms "postconditions" for "effects" or vice versa leading to even bigger confusion. We now discuss both options.

If the logical expression is interpreted as a postcondition, then it describes the state after the service output. However, this suffers from the frame problem. Consequently, it is almost impossible to only use preconditions and postconditions to specify the functionality of Web services. It is an even more challenging problem in comparison to software operations because the knowledge about the complete set of all Web services in the Web cannot be assumed.

If the logical expression is interpreted as the effects of a service, then the end state description is derived by applying the effects to the preconditions describing the start state. However, logic formulae are not the right formalism for specifying effects, since it is generally hard to derive steps required for constructing the state after output from the precondition and effects. For example, a Web service that removes a user from a mailing list (see Example 11). Then, simply adding the effect ψ to the precondition ϕ in order to derive the end state knowledge leads to an inconsistent state description, because it contains a contradiction. In order to allow a reasoner to construct correct models efficiently, the effects need to be specified differently.

In our approach, we model the changes Δ of the service execution in addition to the end state descriptions ψ. Changes describe the effects of (local) actions that lead to state changes during the service execution. In some cases, the effects caused by a service execution can be described implicitly as the "difference" between the start and end state descriptions (ϕ and ψ, respectively). However, a simple change like adding instances to the knowledge base cannot be modeled implicitly due to the underlying open world assumption of ontology reasoners. That is, from absence of an instance x in the start state and the presence of x in the end state, we cannot conclude that x was created by the service execution. The same argumentation applies for changes that update a property of an instance. For example, a product might have multiple prices (represented by multiple ontology object properties) when the price of a product is updated in the knowledge base by adding another property value. If we cannot distinguish between the different price properties, the effect of updating a price cannot be modeled implicitly. Consequently, we need to explicitly model the effects Δ of the service execution.

5.2.3 Modeling Example

We consider an atomic book selling service. The service provides the same functionality as the example service from Section 4.1.3. However, the behavior is based on the single-request-response pattern now, which allows us to describe the service functionality using the above model of a service profile.

Imagine that the atomic service ω requires the following four input parameters with the invocation: user ID, password, and a book's author and title. The service creates a shipping order for the given book. It is shipped to the address that is stored and associated to the user account. The service finally returns an invoice to the user about the order and the book details.

The input parameters I and the output parameters O of the service are specified as follows:

$$\begin{aligned} I &= \{\mathsf{ex{:}id}, \mathsf{ex{:}pwd}, \mathsf{ex{:}author}, \mathsf{ex{:}title}\} \\ O &= \{\mathsf{ex{:}book}, \mathsf{ex{:}invoice}\} \end{aligned} \tag{5.1}$$

The description of the start state ϕ is given below. It states that the described service requires that the user ex:user with ID ex:id is registered and authenticated by its password ex:pwd. For a successful execution, the service furthermore requires that the book ex:book with the specified author ex:author and title ex:title is available.

In addition to these generic features of the book selling functionality, we consider reward points that the user collects with every purchase. It serves as an example to highlight how we model changes. In the start state description in Equation 5.2, rwd denotes a data value representing the number of existing reward points, which will be increased by the amount of the money spent in every purchase.

$$\begin{aligned}\phi \equiv\ & \mathsf{ex{:}Author(ex{:}author)} \wedge \mathsf{ex{:}Title(ex{:}title)} \wedge \exists \mathsf{ex{:}book}\,.\,\mathsf{ex{:}Book(ex{:}book)} \wedge \\ & \mathsf{ex{:}hasAuthor(ex{:}book, ex{:}author)} \wedge \mathsf{ex{:}hasTitle(ex{:}book, ex{:}title)} \wedge \\ & \mathsf{ex{:}isAvailable(ex{:}book)} \wedge \mathsf{ex{:}UserId(ex{:}id)} \wedge \mathsf{ex{:}Password(ex{:}pwd)} \wedge \\ & \exists \mathsf{ex{:}user}\,.\,\mathsf{ex{:}User(ex{:}user)} \wedge \mathsf{authenticatedBy(ex{:}id, ex{:}pwd)} \wedge \\ & \mathsf{ex{:}hasID(ex{:}user, ex{:}id)} \wedge \mathsf{ex{:}hasRewards(ex{:}user, rwd)} \end{aligned} \tag{5.2}$$

The end state description ψ states that there exists an order $\mathsf{ex{:}order}$ for the product $\mathsf{ex{:}book}$. The conditions from ϕ assure that the resource $\mathsf{ex{:}book}$ denotes the desired book with the specified author and title. The book is shipped to the user's address $\mathsf{ex{:}address}$. Furthermore, it states that there exists an invoice about the order and price, which equals the price of the book. Also, ψ guarantees that the user receives for each Euro spent for the book another reward point to its balance after service execution.

$$\begin{aligned}\psi \equiv\ & \mathsf{ex{:}Order(ex{:}order)} \wedge \mathsf{ex{:}containsProduct(ex{:}order, ex{:}book)} \wedge \\ & \mathsf{ex{:}isShipped(ex{:}order, ex{:}address)} \wedge \mathsf{ex{:}Invoice(ex{:}invoice)} \wedge \\ & \mathsf{ex{:}containsPrice(ex{:}invoice, ex{:}price)} \wedge \mathsf{ex{:}containsOrder(ex{:}invoice, ex{:}order)} \wedge \\ & \mathsf{ex{:}hasPrice(ex{:}book, ex{:}price)} \wedge \mathsf{ex{:}hasAddress(ex{:}user, ex{:}address)} \wedge \\ & \mathsf{ex{:}hasValue(ex{:}price, priceVal)} \wedge \mathsf{ex{:}hasRewards(ex{:}user, rwd + priceVal)} \end{aligned} \tag{5.3}$$

As the changes Δ caused by the service execution cannot be derived from the difference of the start and end state descriptions, we explicitly describe them in our example as follows:

$$\begin{aligned}\Delta = \{\ & + \mathsf{ex{:}Order(ex{:}order)}, \\ & + \mathsf{ex{:}Invoice(ex{:}invoice)}, \\ & \sim \mathsf{ex{:}hasRewards(ex{:}user, rwd + priceVal)}\ \} \end{aligned} \tag{5.4}$$

The changes Δ describe that two instances modeling the respective order and the invoice are created and added to the knowledge base of the end state. Furthermore, the reward points of the user account have been increased by the service execution. That is, the initial value rwd was updated to $\mathsf{rwd + priceVal}$. In contrast to [Aga07a], we introduce the update concept, symbolized $\sim$. Updates can be modeled by deleting the statement with an old value followed by adding the statement with an updated value. However, since Δ is a set of changes, the order of delete and add statements to simulate an update is not guaranteed by the description and, hence, the notion of update is required.

Altogether, the functionality description $(I, O, \phi, \psi, \Delta)$ of our example Web service means that the service requires a valid user identification $\mathsf{ex{:}id}$, a password authentication $\mathsf{ex{:}pwd}$ as well as a valid author name $\mathsf{ex{:}author}$ title $\mathsf{ex{:}title}$ for a successful invocation. The service then creates an order $\mathsf{ex{:}order}$ and returns a description $\mathsf{ex{:}book}$ of the desired book and the invoice $\mathsf{ex{:}invoice}$ to the invoker after the execution of the service.

5.3 Functionalities in Service Requests

In compliance with our generic model of service requests that we introduced in Section 4.2.2, we now introduce a formalism to describe the desired service functionality as a property of the request. Following the paradigm that desired property instances in requests describe a set of desired property values, we introduce a formal model and a language for the declarative description of desired functionalities in the subsequent paragraphs. In Section 5.3.2, we will continue our example and show how this request model for functionalities is applied in practice.

5.3.1 A Formal Model of Functionality Constraints

Within this section, we only consider the constraints on the service functionality. We already discussed the remaining constraints on any other service property in Sections 4.2.2 and 4.2.3. It is possible to combine such constraints with the constraints on the functionality by using the methods presented above.

In a service description, the service functionality $(I, O, \phi, \psi, \Delta)$ comprises five sub-properties for the description of start and end states along with the description of input and output parameters and the changes introduced by the service computation.

In a request, the desired functionality is modeled accordingly by five corresponding constraints $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$ on the sub-properties of the functionality description. Each single constraint is a query that describes a set of desired values of the respective sub-property. We explain each constraint in the following:

- Constraint $\mathbb{I}$ is a query expressed as a logic expression that describes desired sets of input parameters.
- Constraint $\mathbb{O}$ describes the desired set of output parameters. The query $\mathbb{O}$ is specified analogously to constraint $\mathbb{I}$.
- Constraint Φ describes the desired start state description of services.
- Constraint Ψ describes the desired end state description of services.
- Constraint Λ describes the desired changes of a service execution.

Each constraint specifies the set of properties values that are allowed. That is, a matching service should have properties with values from the requested sets. The $\mathbb{I}$ and $\mathbb{O}$ describe a set of desired input and output sets. We allow for the expression of conjunctions, alternatives, and exclusions of inputs and outputs using Boolean expressions as we already introduced in Section 4.2.3 for the combination of generic service properties in requests. Analogous to the constraints on inputs and outputs, Φ and Ψ are queries that express constraints on a set of desired start and end state descriptions, respectively. The interpretation of both queries Φ and Ψ is the set of services that fulfill the requested conditions in the states before and after service execution (respectively described by ϕ and ψ in the description). The same interpretation is applied to the constraints Λ on the changes.

A service request expresses conditions over the states by means of logic expressions. The intention of Φ and Ψ is that both conditions must hold in the start and end state descriptions of desired services, respectively. Using an expressive logic like first-order logic to specify the set of desired start and end states not only allows for expressing which conditions are provided by service offers, but also for excluding services with undesired conditions. In general, first-order logic expressions can be used within our approach. In a practical realization of the discovery method that we develop in this thesis as well as in our illustrated examples we restrict the expressivity to description logics. This restriction is practical as description logic reasoning remains decidable and existing reasoners implementing the semantics of description logics can be utilized for realizing our discovery method.

Interpretation of a Request

Intuitively, a service request $\mathcal{R}$ is interpreted as the set of desired services, denoted by $\mathcal{R}^{\mathcal{I}}$. $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ remains the interpretation of a request, where $\cdot^{\mathcal{I}}$ maps each requested property instance, i.e., a property-value-set, into a set of desired property-value pairs. Translated into the formal model, a request is a power-set of property instances $q \in Q$, each of which is of a type t_q and assigned to a value $v \in V_{R,q}$ that is a member of the desired value set $V_{R,q}$.

Since a request describes a set of services, a set $\mathcal{L}$ of labeled transition systems model several potential services. The set of possible values of the property instance that models the functionality is the set of all labeled transition systems. Each desired service execution that fulfills the request $\mathcal{R}$ is modeled by one LTS in the set $\mathcal{L}$. The interpretation function assigns to the requested functionality $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$

1. a set of input parameter sets described by $\mathbb{I}$,
2. a set of output parameter sets described by $\mathbb{O}$,
3. a set of start states s_w described by Φ,
4. a set of end states s_e described by Ψ, and
5. a set of changes Δ described by Λ.

This interpretation provides us means to formalize the desired value set of the functionality property as follows. The request $\mathcal{R}$ with constraints on the functional property describes desired sets of input parameters, output parameters, start and end states, and introduced changes, respectively. The desired functionality $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$ is translated into a set $\mathcal{L}$ of labeled transition systems. The set $\mathcal{L}$ is used to formally model the set of possible service functionalities described by $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$.

An LTS $L^R \in \mathcal{L}$ models a service functionality in terms of the formal model depicted in Figure 5.2. We say that L^R is an element of the interpretation $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)^{\mathcal{I}}$ of the request. In other words, $L^R = (S^R, \mathcal{W}^R, \rightarrow^R)$ models the functionality of one particular and desired service configuration R described by the request and is defined as follows:

$$\begin{aligned}
S^R &:= \{s_i^R, s_e^R\} \\
\mathcal{W}^R &:= \{c^R\}, \text{ with } c^R := (\mathsf{local}, \Delta^R) \\
\rightarrow^R &:= \{(s_i^R, c^R, s_e^R)\}
\end{aligned}$$

Because the request $\mathcal{R}$ constrains the start and end states of L^R, too, the states of L^R fulfill the requirements and constraints of Φ and Ψ.

The LTS L^R, which fulfills the requirements of the request, comprises a start state s_i^R (i.e., the state after invocation), one end state s_e^R, and a transition $(s_i^R, c^R, s_e^R) \in \rightarrow^R$ between the two states. The label $c^R \in \mathcal{W}^R$ with $c^R = (\mathsf{local}, \Delta^R)$ is associated with the transition and describes the local action that introduces changes Δ^R.

For each state s_i that entails the requested start state constraint Φ, and for each state s_e that entails the requested end state description Ψ, there exists an LTS $L \in \mathcal{L}$ with a transition $(s_i, c, s_e) \in\rightarrow$ in L. Furthermore, a set of desired changes is attached to the label c of this transition. That is, there exists an LTS in $\mathcal{L}$ for each combination of a particular start and end state, and a set of changes. A start state s_i represents a possible answer of the knowledge base query Φ. Analogously, each knowledge base answer to the requested effect Ψ introduces one end state s_e.

5.3.2 Modeling Example

We continue our example of an atomic book selling service. We presented its functionality description in Section 5.2.3.

A library that frequently places book orders is looking for new and alternative online retailers that sell and ship books. A service request for such service functionalities may have the following input specification. Disregarding the remaining request description, the input specification $\mathbb{I}$ corresponds to the set of services that can either identify a book by its ISBN i or alternatively by the author a and the title t of the book to order. The requested service, however, should not require any credit card information cc. Coherences between the inputs or outputs are expressed in the formulae that model requested start and end states, respectively.

$$\begin{aligned} \mathbb{I} &= \neg\{\mathsf{cc}\} \wedge \big(\{\mathsf{i}\} \vee \{\mathsf{a}, \mathsf{t}\}\big) \\ \mathbb{O} &= \{\mathsf{inv}\} \end{aligned} \tag{5.5}$$

The expected outputs shall comprise the invoice inv of the book. The start and end states are restricted by the following constraints. Within the desired start state description, the relations among the resources describing the book are specified and no further assumptions have to hold. After the service execution, the specified book has been sold.

$$\begin{aligned} \Phi \equiv\ & \exists \mathsf{book}\,.\, \mathsf{ex{:}Book(book)} \wedge \mathsf{ex{:}hasAuthor(book, a)} \wedge \mathsf{ex{:}hasTitle(book, t)} \wedge \\ & \mathsf{ex{:}Author(a)} \wedge \mathsf{ex{:}isAvailable(book)} \\ \Psi \equiv\ & \exists \mathsf{inv}\,.\, \mathsf{ex{:}Invoice(inv)} \wedge \mathsf{ex{:}hasItem(inv, book)} \wedge \ldots \end{aligned} \tag{5.6}$$

The invoice and the shipped book can be further specified in Ψ, e.g., to express that the book is shipped to the right address, the invoice is issued to the buyer, and the taxes are shown separately. However, in order to ensure that the mentioned invoice inv was created for this particular order, a request has to further describe this change explicitly. It prevents that the mentioned invoice inv in the end state refers to an object that was previously created for an order of the same book by the same buyer.

$$\Lambda = \{+\mathsf{ex{:}Invoice(inv)}\} \tag{5.7}$$

5.4 Discovery of Atomic Services

The discovery of atomic services differs from the model checking based matchmaking technique for generic services (introduced in Section 4.3) as it applies a matchmaker for the service functionality instead of the service behavior. We still apply the generic matchmaking method to match non-functional properties of atomic services and to evaluate combinations of desired properties of a service request. In comparison to the matchmaking of the behavior, matching the functionality becomes conceptually easier: only two states need to be considered in functionality profile descriptions.

In the present section, we focus on matching the requested with the offered functionality of services. That is, we introduce an automated method for the matchmaking of one particular service property.

Assume that we have a service repository $\mathbb{D}$ that contains a finite number of semantic descriptions of atomic services and the domain ontologies used to describe each of the services. Our discovery approach employs a logic-based matchmaking technique in order to compare each offered service functionality description with a desired functionality description. As there are no dependencies between different services with respect to the offered functionality, each service can be matched individually against the request. Hence, a matchmaker can check the services in $\mathbb{D}$ sequentially or in parallel. The discovery result for each service request $\mathcal{R}$ is a set of services that fulfill all requirements expressed by $\mathcal{R}$. The logic-based matchmaking algorithm produces Boolean decisions, that is, a service is considered to be a match or not.

We use the presented semantics of the description and request formalisms. Based on the labeled transition systems as their common formal model, the requests are evaluated over the LTS representation of each service description. The matchmaker checks for a membership relation between the interpretations of service descriptions and requests, as we consistently modeled each property of a request as a set of desired values. This applies to all property-value pairs including the LTS-based formal interpretation of offered and requested functionalities. If the offered service is in the set of desired services described by the request, our matchmaker can confirm that all the constraints of the request are fulfilled in the LTS of the offered service and adds the service to the set of matches.

Model Checking Based Functionality Matchmaking

We define a match between an offered service functionality $(I, O, \phi, \psi, \Delta)$, which is considered to be a value of the functionality property of a service description, and a functionality request $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$, which is considered to be a value set.

Let L^ω denote the LTS interpretation of the functionality description $(I, O, \phi, \psi, \Delta)$ of service ω, i.e., L^ω is the value of the functionality property. Also, let the set $\mathcal{L}$ of LTS denote the interpretation of the requested functionality $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$. Analogously, $\mathcal{L}$ corresponds to the set of desired values of the service functionality property. Then, a service functionality matches a requested functionality if and only if $L^\omega \in \mathcal{L}$. That is, service ω matches a request $\mathcal{R}$ if and only if L^ω is a model of $\mathcal{R}^\mathcal{I}$. In order to define a match, we first clarify the relation between the behavior descriptions and requests and the functionality descriptions and requests. If we can translate functionality descriptions and requests into behavior descriptions and properties (requests), then we can apply the same model checking based matchmaking approach to both complex and atomic services. Then, the accuracy of the results guaranteed by the logic-based matchmaking is conserved.

Applicability of model checking. In our approach, every functionality description is also a valid behavior expression. In both cases we use labeled transition systems as the underlying structure. The formal model of functionality (cf. Figure 5.2) consists of an input action, a local action causing effects, and an output action.

A similar relation between requested behavior and functionality exists. Functionality requests are a subset of behavior properties in requests. A functionality request $(\mathbb{I}, \mathbb{O}, \Phi, \Psi, \Lambda)$ can be translated to the following behavior constraints:

- There has to be an initial input action i with the parameters specified in $\mathbb{I}$. We specify the start state, i.e., the state subsequent to the input action and denoted by s_i in our model depicted in Figure 5.2, by the proposition $\lambda(s_i) = \Phi$. It is translated into the following behavior constraint $[i]\Phi$, where i is an input action.
- The requested changes Λ are modeled as a property that demands the existence of a local action $s_i \to^c s_e$. In conjunction to the previous constraint, we specify the constraints as follows:

$$[i]\big(\Phi \wedge_\mu [\mathsf{c}^\Lambda]\mathbf{true}\big)$$

It states that the condition Φ holds in the state after input action i. Further, we specify that the local action c with the changes Λ has to take place in the state after the input action. There are no constraints (expressed by **true**) on the desired end state after any local action c.
- After the changes have been performed, the service reaches its end state that is described by $\lambda(s_e) = \Psi$ and performs an output action o, which returns $\mathbb{O}$. After the output, the condition Ψ remains true. We combine the above developed constraints with the end state constraints by replacing **true** with $\Psi \wedge_\mu [o]\Psi$. We obtain the following single behavior constraint specification:

$$[i]\Big(\Phi \wedge_\mu [\mathsf{c}^\Lambda]\big(\Psi \wedge_\mu [o]\Psi\big)\Big) \tag{5.8}$$

Note that we do not make assumptions about the initial state s_w before the input action i, and the output action o does not change the end state s_e. Consequently, the condition Ψ holds subsequent to the local action and the output action.

Because the functionality descriptions are behavior descriptions, too, and we can translate functionality requests into behavior constraint specifications, we can also apply the presented model checking based matchmaking technique of the previous chapter to atomic services. We can simplify the model checking process since both the LTS and the request structures are very restricted.

Matchmaking semantics. The correct order of input, local, and output actions does not have to be evaluated. The formal model of the functionality of atomic services already defines the structure. Consequently, we only need to evaluate the set of input and output parameters, start and end state propositions, and the changes.

Input and output parameters in descriptions and requests refer to the parameter names, i.e., without any semantic annotations like types and properties. We check if there is a variable mapping between the names used for the parameters in a request and the ones used in the service description. The variables V considered in a mapping are either free variables V, inputs, or outputs. For every possible mapping m, with

$$m : V \cup \mathbb{I} \cup \mathbb{O} \rightarrow V \cup I \cup O,$$

we have to evaluate the start and end states and changes as follows.

The start state proposition Φ *holds* iff the start state s_i of a service entails the proposition. Similarly, the end state proposition Ψ holds iff the end state knowledge entails the propositions. We apply the mapping m to all the parameters in the request.

$$\begin{aligned} &KB(s_i) \models \Phi_{[\forall x \in I : x \leftarrow m(x)]} \\ &KB(s_e) \models \Psi_{[\forall x \in I \cup O : x \leftarrow m(x)]} \\ &\forall \delta \in \Lambda : \exists \delta' \in \Delta : type(\delta) \equiv type(\delta') \wedge KB(\delta') \models KB(\delta_{[\forall x \in I \cup O : x \leftarrow m(x)]}) \end{aligned} \tag{5.9}$$

The changes have to be checked in another knowledge base, as they are not included in the end state. For every requested change $\delta \in \Lambda$ and a mapping m, we create a knowledge base $KB(\delta_{[\forall x \in I \cup O : x \leftarrow m(x)]})$. We check for change type equivalence ($type(\delta) \equiv type(\delta')$) and evaluate whether there exists a proper change δ' caused by the service for every requested change δ.

An atomic service matches a requested functionality if there exists a mapping such that the matching conditions in Equation 5.9 are fulfilled. That is, matching services fulfill the specified constraints.

5.5 Implementation and Evaluation

We developed, implemented, applied, and evaluated our discovery method for atomic services in the context of the SOA4All project. We report in Section 5.5.1 on the integration of the discovery engine into the developed service platform including the alignment to the used Semantic Web technologies. In Section 5.5.2, we present technical details of the implementation. We explain how the knowledge bases of a given set of semantically described Web service descriptions are created, imported into a description logic reasoner, and used during matchmaking. Based on the implementation of our discovery method, we have conducted several experiments to measure the performance of the engine. In Section 5.5.3, we present and discuss the measured performance results.

5.5.1 Service Discovery in SOA4All

The presented discovery method for atomic services has been implemented and evaluated using the techniques, tools, languages, and data sets given by the project. For instance, we built our discovery engine on top of the WSML reasoner framework.[7] Therefore, descriptions and queries are bound to the syntax of the Web Service Modeling Language WSML [dBFK$^+$08]. More precisely, we used the WSML-Flight language dialect [BFH$^+$09] for Web service descriptions within the project. Of course, our discovery method is not restricted to this choice. It is possible to use different description logic reasoners and syntaxes.

[7]WSML reasoner framework – http://tools.sti-innsbruck.at/wsml2reasoner, retrieved 2013-06-16

WSML-Flight is a language based on WSML-Core, with an extension towards logic programming. WSML-Core was extended by a rule language that allows efficient and decidable reasoning and supports Datalog rules with locally stratified negation [LdBPF05]. The basic WSML-Core covers the intersection of description logic and Horn logic [GHVD03]. The WSML reasoner that we used also supports data types, which is useful for practical applications of the discovery engine.

The semantic discovery component is developed as a Web service so that it can be used by other SOA4All components with standard Web protocols. The discovery service interface offers a range of operations for various purposes related to the discovery of Web services.

Use of service discovery. The service ranking and selection component ranks the relevant services that were identified by the service discovery system and selects the most appropriate results. From an architectural perspective, service discovery is expected to return a set of services whose service descriptions match the request. The service discovery solution also enables the construction of services using dynamic and adaptive composition, and the reconfiguration of constructed services in reaction to environmental changes. At binding time, parametric templates that represent service compositions as well as the composition optimizer require concrete services that are identified by the service discovery engine.

Also, all use cases directly depend on service discovery. We already presented how service discovery was applied, e.g., to locate relevant services within enterprise and e-government service portals or to dynamically locate and offer complementary third party functionalities (like sending SMS) in geographical regions in which a telecommunication provider does not operate (see Section 2.1.1).

Service descriptions. Semantic Web Service descriptions use the WSMO-Lite service ontology [VKVF08] in conjunction with the Procedure-Oriented Service Model (POSM). POSM is a service model that represents the structure of a service description containing concepts like Service, Operation, etc. POSM refers to the concepts provided by WSMO-Lite. By this, POSM allows us to describe services semantically using WSMO-Lite Annotations without annotating a WSDL document. In order to bridge the gap between the formal models on a theoretical level that were introduced above and the concrete implementation, we provide the mapping between the two levels as depicted in Table 5.1 ($\mathsf{posm}:$ and $\mathsf{wsl}:$ abbreviate the namespaces of POSM and WSMO-Lite, respectively). Consequently, we aligned the vocabulary of the property-based service model to the given concepts from WSMO-Lite and POSM ontologies.

We decided to attach the end state description ψ to the effect element of the WSMO-Lite service ontology, because WSMO-Lite effects are specified as conditions that hold in a state after the service invocation [FFK+10]. The changes of the service descriptions could not be linked to the service instance using existing service modeling ontologies. Therefore, we used own vocabulary (such as supProc:hasChange) from the suprime behavior modeling ontology.

Integration with SOA4All service repository. The semantic descriptions are stored in the SOA4All service repository. Each service is described by one ontology document (denoted

Table 5.1: Mapping between the sub-properties of the formal model and representation in service descriptions

Formal Model	**Semantic Service Description**
Inputs I	posm:hasInputMessage and posm:Message
Outputs O	posm:hasOutputMessage and posm:Message
Start state ϕ	posm:hasCondition and wsl:Condition
End state ψ	posm:hasEffect and wsl:Effect
Changes Δ	supProc:hasChange and supProc:Change
NFPs	wsl:NonFunctionalParameter

by $O_{\mathcal{D}_\omega}$ earlier). The external storage location of the domain ontologies (O_ω) are referenced in the header of the service description.

Semantic service descriptions can be created by means of the SOA4All Studio: Given OWL-S Profile, SAWSDL, and WSDL service descriptions can be imported. Semantic service descriptions within the SOA4All service repository are modeled using the minimal service model, which is the vocabulary supported by the repository. The slightly different POSM-based service descriptions, which are used by the discovery and the remaining SOA4All components, were easily derived from MSM-based descriptions.

Service descriptions are retrieved from the repository in the form of WSML ontologies serialized in RDF. We use the RESTful service interface of the service description repository in order to obtain a list of currently registered Web service descriptions. Each service description retrieved from the repository is parsed using the wsmo4j API,[8] which is a Java API and object model for WSMO-Lite service descriptions. This then allows us to serialize the parsed service description into an RDF representation of the WSML ontology using the POSM service model.

Integration with WSML2Reasoner. To achieve the requirement for accurate results, the discovery solution must integrate and use the reasoning support. A clearly defined interaction that results in a well-defined interface between the two components was employed. The service discovery component uses the reasoner to provide an automated method for the matching of requests and services. Different reasoning functionalities, such as subsumption reasoning, satisfiability checking, and instance retrieval, are used by the discovery component.

The translated service description ontologies retrieved from the service repository are imported into the WSML reasoner instance using the reasoner's RDF parser functionality. These ontologies build the knowledge base on which reasoning tasks are later performed. In order to separate the different states of the functionality, we create two knowledge bases. Each knowledge base is loaded into a separate reasoner instance. By this, we avoid introducing logic inconsistencies by contradicting state descriptions caused by the changes of a service. We elaborate this approach when we present the implementation details in Section 5.5.2.

Service templates. A simple graphical interface allows users to formulate and submit service requests. The requests can combine constraints on the desired NFPs, inputs, outputs,

[8]wsmo4j – http://wsmo4j.sourceforge.net, retrieved 2013-06-11

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix st: <http://www.wsmo.org/ns/service-template/0.1#> .

st:ServiceTemplate rdf:type rdfs:Class.
st:hasFunctionalCategory rdf:type rdfs:Property.
st:hasInput rdf:type rdfs:Property.
st:hasOutput rdf:type rdfs:Property.
st:hasPreference rdf:type rdfs:Property.
st:hasRequirement rdf:type rdfs:Property.
```

Listing 5.1: RDF Schema for service template

and start and end states. The request information is encapsulated into a so-called service template object. A service template is defined by an RDF/S ontology (see [KDPS10], though the definition has evolved) and contains the elements inputs, outputs, requirements, and preferences. These elements of a service template are defined by the respective properties as summarized in Listing 5.1 taken from [KDPS10].

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wsml: <http://www.wsmo.org/wsml/wsml-syntax#> .
@prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix st: <http://www.wsmo.org/ns/service-template/0.1#> .
@prefix sf: <http://www.service-finder.eu/ontologies/ServiceCategories#> .
@prefix sr: <http://seekda.com/ontologies/RankingOntology#> .
@prefix pref: <http://www.wsmo.org/ontologies/nfp/preferenceOntology#> .
@prefix bs: <http://www.example.com/bookselling#> .
@prefix ex: <http://localhost:8081/DisCloud/serviceTemplates/BSS#> .

ex:stBSS a st:ServiceTemplate ;
    st:hasFunctionalCategory sf:BookSellingService ;
    st:hasInput bs:isbn , bs:id ;
    st:hasOutput bs:book , bs:inv ;

    st:hasRequirement [
        rdf:type wsl:Condition ;
        rdf:value "?isbn␣memberOf␣␣\"http://www.example.com/bookselling#ISBN\"␣and␣␣?user[␣\"http://www.example.com/bookselling#hasRewards\"␣hasValue␣?rpre]␣memberOf␣␣\"http://www.example.com/bookselling#User\""^^wsml:AxiomLiteral
        ] ;

    st:hasRequirement [
        rdf:type wsl:Effect ;
        rdf:value "?book[␣\"http://www.example.com/bookselling#hasISBN\"␣hasValue␣?isbn]␣memberOf␣␣\"http://www.example.com/bookselling#Book\"␣and␣?inv␣memberOf␣␣\"http://www.example.com/bookselling#Invoice\"␣and␣?user[␣\"http://www.example.com/bookselling#hasRewards\"␣hasValue␣?rpost]␣and␣?rpre[␣\"http://www.example.com/bookselling#lessThan\"␣?rpost]"^^wsml:AxiomLiteral
        ] ;

    st:hasRequirement rdf:value "?s[␣\"http://www.example.com/bookselling#hasDeliveryTime\"␣hasValue␣?dt]␣and␣?dt[␣\"http://www.example.com/bookselling#lessThan\"␣␣int(\"7\")]]"^^wsml:AxiomLiteral] ;

    st:hasRequirement rdf:value "?s[␣\"http://www.example.com/bookselling#acceptCreditCard\"␣hasValue␣boolean(\"true\")]"^^wsml:AxiomLiteral] .

     ...
```

Listing 5.2: Example of a service template

Obviously, inputs $\mathbb{I}$ and outputs $\mathbb{O}$ of a service request are captured by the respective elements within a service template. Desired start and end state descriptions as well as non-functional requirements are encapsulated in the requirements field of service templates by using multiple instances of the hasRequirement property. Start and end state descriptions are logical formulae and encoded as string representations of WSML axioms. The preferences are used for service ranking. The functional category is used for a simple classification-based discovery, as presented in [AJF+09] and depicted in Figure 5.4a.

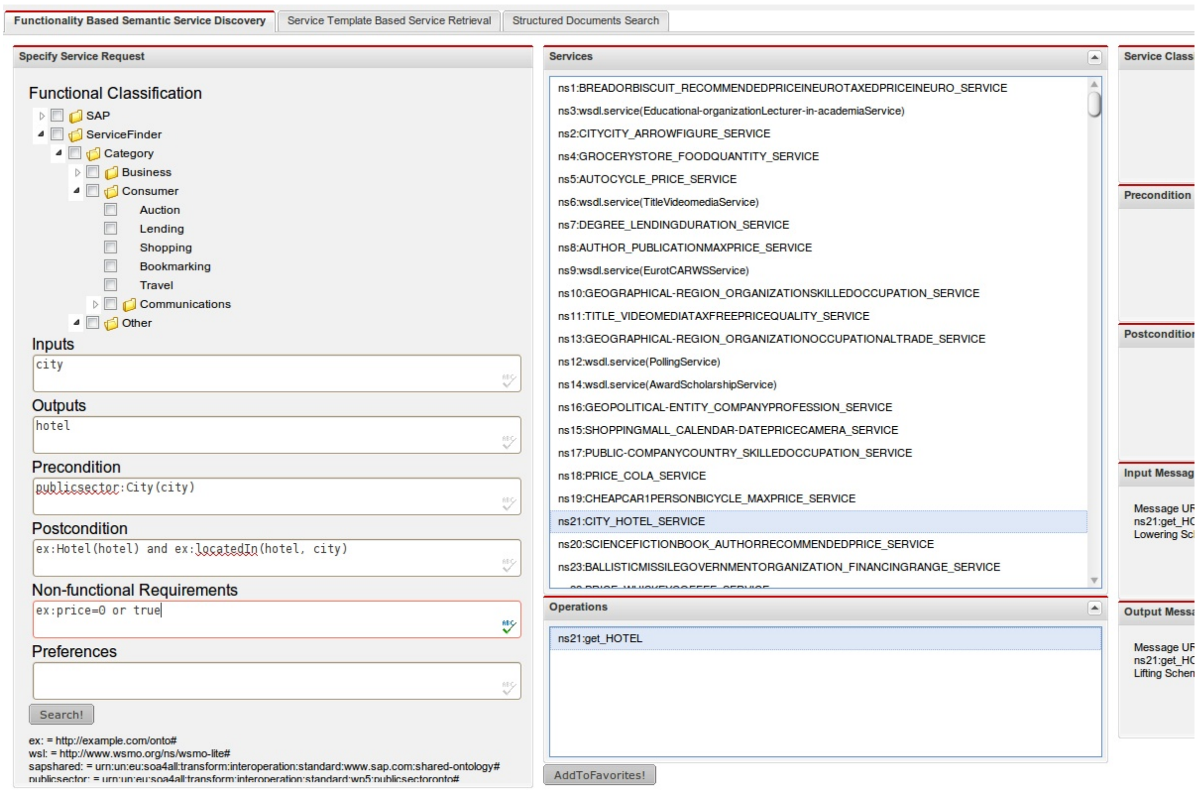
We give an example of a request serialized as an RDF/S ontology using the service template schema in the following Listing 5.2. Here, an instance ex:stBSS of the service template represents the request for a book selling service. A desired service should accept two input parameters bs:isbn and bs:id, which represent the desired book and a user log-in, respectively. The first requirement in line 18ff. (Listing 5.2) encodes the description of the start state. It specifies input parameter types using the concepts of the domain ontology and refers to the user's account balance. The end state description in line 23ff. is the second requirement listed in the example service template and specifies that an invoice for the ordered book is created and reward points are collected.

The remaining two requirements in Listing 5.2 specify non-functional requirements. It states the service accepts credit card payments and the service promises to deliver in less than seven days.

User interface. The service discovery component of SOA4All comes with a Web-based graphical user interface. In Figure 5.3, the discovery interface is shown for a hotel search service request. In this shown example, it is specified that desired services should return hotels that are located in a specified city. Figure 5.4b shows the widget for the functional requirements in detail.

The discovery user interface is an integrated module of the SOA4All Studio and was implemented using the Google Web Toolkit (GWT). The user interface allows users to enter a request for service functionalities by specifying:

- The functional classification of the service. Classification-based discovery was introduced in [AJF+09]. It allows selecting a set of classes displayed to the user as depicted in Figure 5.4a. Service descriptions contain an assignment to a subset of available classes. Each class is represented by a human-readable name and also defined by an ontology class. These classes can be organized in the form of a hierarchy by defining sub-class-of relationships between them. However, no semantics with respect to the functionality is considered in such class definitions.
- The desired functional and non-functional properties of a request. It includes the fields for inputs, outputs, start state conditions (preconditions), end state conditions (effects), and non-functional requirements. In order to support users in expressing these conditions, concepts and properties of registered (domain) ontologies are automatically suggested for completion as depicted in Figure 5.4b. The suggestions also provide help with the syntax, e.g., by suggesting applicable operators.
- Preferences that are used to rank the set of results according to what the user describes in this field. In our fuzzy logic based service ranking approach presented in [AJN10b,


Functionality Based Semantic Service Discovery | Service Template Based Service Retrieval | Structured Documents Search

Specify Service Request

Functional Classification
- SAP
- ServiceFinder
  - Category
    - Business
    - Consumer
      - Auction
      - Lending
      - Shopping
      - Bookmarking
      - Travel
      - Communications
    - Other

Inputs
```
city
```
Outputs
```
hotel
```
Precondition
```
publicsector:City(city)
```
Postcondition
```
ex:Hotel(hotel) and ex:locatedIn(hotel, city)
```
Non-functional Requirements
```
ex:price=0 or true
```
Preferences

Search!

ex: = http://example.com/onto#
wsl: = http://www.wsmo.org/ns/wsmo-lite#
sapshared: = urn:un:eu:soa4all:transform:interoperation:standard:www.sap.com:shared-ontology#
publicsector: = urn:un:eu:soa4all:transform:interoperation:standard:wp5:publicsectoronto#

Services

ns1:BREADORBISCUIT_RECOMMENDEDPRICEINEUROTAXEDPRICEINEURO_SERVICE
ns3:wsdl.service(Educational-organizationLecturer-in-academiaService)
ns2:CITYCITY_ARROWFIGURE_SERVICE
ns4:GROCERYSTORE_FOODQUANTITY_SERVICE
ns5:AUTOCYCLE_PRICE_SERVICE
ns6:wsdl.service(TitleVideomediaService)
ns7:DEGREE_LENDINGDURATION_SERVICE
ns8:AUTHOR_PUBLICATIONMAXPRICE_SERVICE
ns9:wsdl.service(EurotCARWSService)
ns10:GEOGRAPHICAL-REGION_ORGANIZATIONSKILLEDOCCUPATION_SERVICE
ns11:TITLE_VIDEOMEDIATAXFREEPRICEQUALITY_SERVICE
ns13:GEOGRAPHICAL-REGION_ORGANIZATIONOCCUPATIONALTRADE_SERVICE
ns12:wsdl.service(PollingService)
ns14:wsdl.service(AwardScholarshipService)
ns16:GEOPOLITICAL-ENTITY_COMPANYPROFESSION_SERVICE
ns15:SHOPPINGMALL_CALENDAR-DATEPRICECAMERA_SERVICE
ns17:PUBLIC-COMPANYCOUNTRY_SKILLEDOCCUPATION_SERVICE
ns18:PRICE_COLA_SERVICE
ns19:CHEAPCAR1PERSONBICYCLE_MAXPRICE_SERVICE
ns21:CITY_HOTEL_SERVICE
ns20:SCIENCEFICTIONBOOK_AUTHORRECOMMENDEDPRICE_SERVICE
ns23:BALLISTICMISSILEGOVERNMENTORGANIZATION_FINANCINGRANGE_SERVICE

Operations

ns21:get_HOTEL

AddToFavorites!

Service Class

Precondition

Postconditio

Input Messag

Message UF
ns21:get_HC
Lowering Sc

Output Messa

Message UF
ns21:get_HC
Lifting Schen

Figure 5.3: SOA4All service discovery user interface: The request is specified on the left and the search results are displayed on the right.

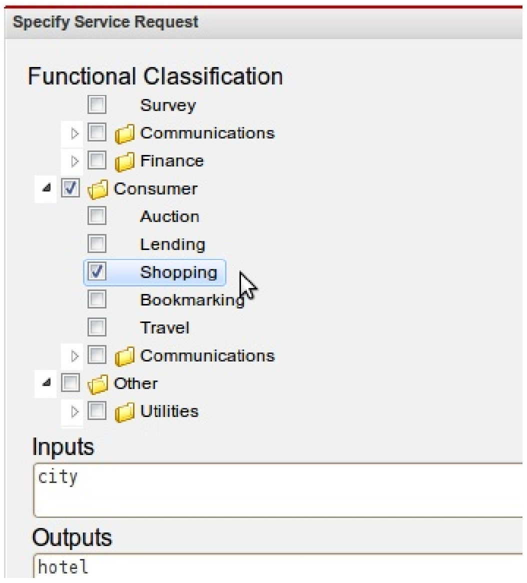
GJR+13], we allow for fuzzy preferences that combine preferences over multiple service properties.

After passing the request in form of a service template to the discovery service, the user interface retrieves a set of services (in fact, we used the notion of operations in SOA4All) that fulfill the request. If preferences were specified, then the displayed result set is an ordered list with descending adherence to the preferences. In addition, some further information about a selected service and operation are displayed on the right side of the Web interface, which is omitted in Figure 5.3 to improve the readability.

If the requesting user is logged on to the SOA4All Studio with an OpenID account, the discovered services and operations can be added to the personal list of favorite services. This favorites list, which is another module of the SOA4All Studio, can then be used in other modules like the SOA4All Process Editor and allows binding abstract process activities to one of the elements on the user's list of favorites.

5.5.2 Implementation Details

As shown above, a simple user interface allows specifying a request. After submission, two reasoners compute the list of matching services from the repository of Web service descriptions.


Specify Service Request

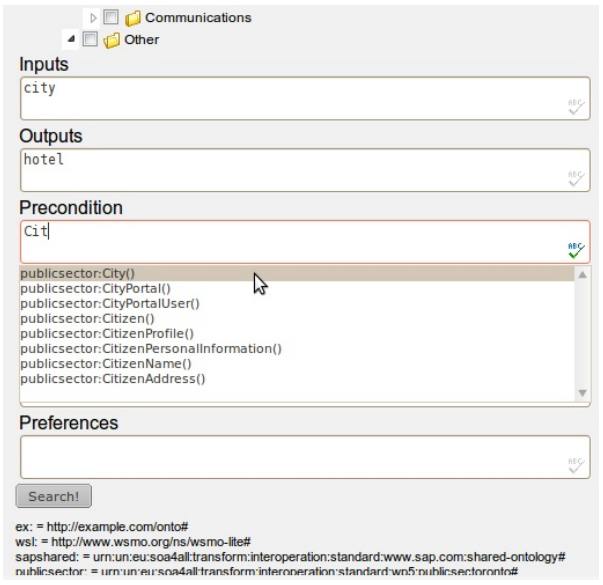
Functional Classification

- [ ] Survey
- [ ] Communications
- [ ] Finance
- [✓] Consumer
  - [ ] Auction
  - [ ] Lending
  - [✓] Shopping
  - [ ] Bookmarking
  - [ ] Travel
  - [ ] Communications
- [ ] Other
  - [ ] Utilities

Inputs: city

Outputs: hotel

(a) Selection of classes from a given classification


- [ ] Communications
- [ ] Other

Inputs: city

Outputs: hotel

Precondition: Cit

```
publicsector:City()
publicsector:CityPortal()
publicsector:CityPortalUser()
publicsector:Citizen()
publicsector:CitizenProfile()
publicsector:CitizenPersonalInformation()
publicsector:CitizenName()
publicsector:CitizenAddress()
```

Preferences

Search!

```
ex: = http://example.com/onto#
wsl: = http://www.wsmo.org/ns/wsmo-lite#
sapshared: = urn:un:eu:soa4all:transform:interoperation:standard:www.sap.com:shared-ontology#
publicsector: = urn:un:eu:soa4all:transform:interoperation:standard:wp5:publicsectoronto#
```

(b) Specification of functional and non-functional requirements

Figure 5.4: Request specification in the SOA4All user interface

In the following, we explain how the two knowledge bases are constructed from the service descriptions and how the matchmaker uses them. Services that were identified as match by both reasoners are results of the request and are thus displayed to the user.

Knowledge base construction. The functionality of a service ω describes two states. We create two knowledge bases $KB(s_i)$, $KB(s_e)$ that model the state before and after execution, respectively. Service properties are modeled as properties of the service instance within the respective knowledge bases.

The input parameters, described in I, are modeled as ontology instances and added to both knowledge bases. Output variables in O are also modeled as instances but only added to $KB(s_e)$. Adding the inputs to $KB(s_e)$ allows us to reason on the relationships between input and output parameters.

The start state description ϕ is added as an axiom to $KB(s_i)$, which models the state before the service execution, and the end state description ψ is added as an axiom to $KB(s_e)$. The same procedure is applied for adding further service descriptions to the same two knowledge bases $KB(s_i)$, $KB(s_e)$.

We prevent inconsistencies due to adding multiple service instances to the same knowledge bases by assuming and ensuring unique names of individuals. That is, it is not possible that the same ontology instance is used in multiple service descriptions. In case of any conflicts, a substitution by renaming the instances resolves a clash without altering the semantics of the service description. Aside from these two knowledge bases, we create a knowledge base $KB(\delta)$ for every change $\delta \in \Delta$ of a service.

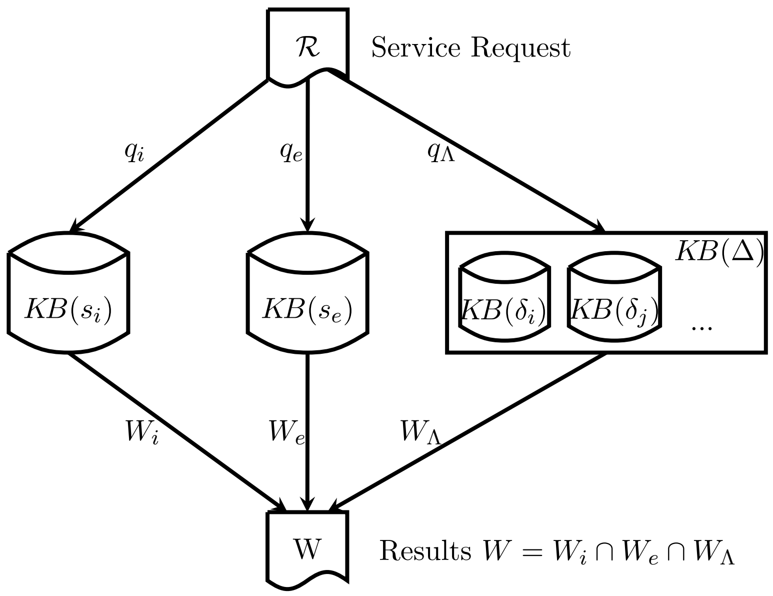


Figure 5.5: Decomposition of a service request $\mathcal{R}$ into knowledge base queries

Matchmaking. The discovery engine receives a request from the user interface and translates it into two queries q_i and q_e expressed in the WSML query language syntax. Figure 5.5 depicts the decomposition of the request into the knowledge base queries. The query q_i is created from the requested inputs $\mathbb{I}$ and the requested start state description Φ. It is sent to the first reasoner instance that models $KB(s_i)$. The query q_e is created from the requested outputs $\mathbb{O}$ and end state description Ψ. The query may also comprise input parameters from $\mathbb{I}$ within the formula Ψ. This query is sent to the second reasoner instance that models $KB(s_e)$.

Both reasoner instances execute the respective queries q_i, q_e on their knowledge bases that contain several services. In order to answer the query q_i, the first reasoner determines for each service ω modeled in $KB(s_i)$, whether required inputs with appropriate parameter types and the condition Φ holds in the start state. That is, the reasoner checks whether the requested start state Φ of a request is fulfilled by the facts in the ABox that were introduced by the start state description ϕ of the service ω. In order to do this, the reasoner checks all potential variable mappings between query and service description including input variables. If there is a variable mapping that fulfills the constraints on the start state descriptions and the requested set of I, a match is identified. Query q_e is processed analogously by the second reasoner instance with $KB(s_e)$.

Below, a fragment of an example query that is sent to the first reasoner instance is presented in WSML syntax. The example query is shown in the syntax that is also accepted by the WSML-Flight reasoner for retrieving instances. The instance retrieval reasoning task is used to ask for services (`?w`) in the knowledge base that fulfill any requirement specified in the remaining query. In WSML syntax, the question mark denotes variables.

```
?w memberOf posm#Service and ?w[posm#hasCondition hasValue ?p] and
?w[posm#hasInputMessage hasValue ?t] and ?t memberOf ex#Title and
?w[posm#hasInputMessage hasValue ?a] and ?a memberOf ex#Author and
?p[posm#hasVariable hasValue ?b] and ?b memberOf ex#Book and
?b[ex#hasTitle hasValue ?t] and ?b[ex#hasAuthor hasValue ?a] and ...
```

The shown query contains the specification of requested inputs, their types, and the start state. In WSML syntax, `posm#` abbreviates the POSM namespace. The shown query asks for services that have two inputs `?a` and `?t` of type Author and Title of an example domain ontology with the prefix `ex#`, respectively. The inputs describe a book `?b` of type Book. The continuation of the example may also specify further conditions on the book `?b` et cetera.

After sending the queries q_i and q_e to the corresponding reasoners, the reasoners bind the variable `?w` to ontology instances that represent services and fulfill the queries q_i and q_e, respectively. We aggregate the intermediate result sets $W_i \subseteq \mathbb{D}$ and $W_e \subseteq \mathbb{D}$, i.e., bindings for `?w`, by computing the intersection of the set. The query q_Λ is sent to the box on the right of Figure 5.5. Here, each reasoner models in $KB(\delta_i)$ a single change δ_i of a service. If the requested changes in q_Λ match the all the changes caused by a services, then this service is part of the intermediate results $W_\Lambda \in \mathbb{D}$. A service is a match for a given request, if the service is an answer to for all queries q_i, q_e, and q_Λ, i.e., the service is identified as a match by the reasoners. The user interface of the discovery engine receives the list of matching services and displays them to the user.

The above described prototypical implementation was published by the SOA4All consortium under an open source license.

5.5.3 Performance Results

We performed several tests of the implementation of the semantic service discovery method. As the presented formal approach already guarantees the result accuracy and therefore the applicability of discovered services for a given task at hand and described by the request, it is not necessary to assess the result quality, e.g., based on precision and recall. Instead, we tested the performance of the discovery implementation in order to examine its feasibility and tractability. That is, we measured the query answering time, specifically the time between submitting a request and the retrieval of the discovery results. Typically, semantic matchmaking highly depends on the performance of the reasoner and the reasoner performance again depends, among other things, on the size of the knowledge base. In [WP10], the performance of the WSML reasoner was evaluated.

In the context of the SOA4All project, the Web service description repository was populated by seekda, a company that crawls the Web for Web services. However, given that service descriptions crawled by seekda mainly represent the information derived from WSDL service descriptions, we decided to synthesize rich semantic service descriptions in a fairly large scale to carry out our evaluation experiments. The SOA4All service description repository also did not provide a large number of semantic service descriptions at that time. Therefore, we created a set of randomly generated service descriptions with varying size ranging from 1,000 to 30,000 descriptions, which is approximately the number of currently available Web services according to seekda.[9] We used the Semantic Web for Research

[9]Trends available at http://webservices.seekda.com/about/web_services, retrieved 2013-08-15

Community (SWRC) ontology [SBH$^+$05] as domain knowledge to model service descriptions. It provides terminological knowledge (TBox) comprising ontology concepts and properties. We use this domain ontology to model types of instances (like input and output parameters) and to express the conditions used to describe the states. We measured the reasoner's mean query answering time of 100 repetitive runs.

As we abstract from the distinction between services and operations, the synthetic semantic service descriptions provide one operation each. An operation expects 1 to 8 inputs and returns 1 to 8 outputs. The precise numbers were chosen randomly. Note that we refer to message parts of the input or output message of the service operation. Each input and output is assigned to a random concept of the SWRC ontology within the description of start and end states, respectively. Then, we randomly generated up to 8 further variables for each state description. Each variable or input/output parameter can be related with other ones within the start or end state. Further, we generated up to 6 non-functional properties per service. Non-functional properties are modeled by an instance of an ontology concept, which is associated with a random precise value within the range of property values. The service description generator is available at http://sf.net/p/svcgenerator/.

We conducted these experiments on two different machines and examined the ability to handle as many service descriptions as possible. The measured query answering time provides us information about the feasibility of the discovery method. As we stated in Section 2.2, the matching services have to be delivered by the discovery engine in a time frame that will not hamper the software development process, e.g., of Web applications. We deployed our system to a commodity laptop in Setup 1. As we quickly experienced that the ontology reasoner requires more main memory to load and reason over large ontologies, which is required to support large service repositories, we deployed the discovery engine to a more powerful computer in Setup 2.

Setup 1

Both queries q_i and q_e are sent in parallel to the reasoners, which compute the answers on a commodity laptop with dual core 2.4GHz CPU and 4GB of main memory. Both knowledge bases contain 1000, 2000, 3000, 4000, and 5000 Web service descriptions. Queries of three different sizes are sent to each knowledge base. Small (S_1), medium (M_1), and large (L_1) queries with 1, 2, 3 instances and 2, 4, 6 properties on those instances are issued, respectively. Figure 5.6 shows the mean time in milliseconds for different knowledge base and query sizes.

Although the curves presenting our measurements in Figure 5.6 almost indicate a linear relationship within the tested range, we cannot assume linear or polynomial discovery complexity. The upper bound complexity of description logic reasoning is often Exptime-hard or even worse [Don03]. Therefore, we cannot claim scalability based on the observed measurements. However, we can derive that the feasibility of the presented approach applied to rather small service repositories can be observed. The figure reveals that the query complexity and the number of Web service descriptions loaded into the knowledge bases of both reasoner instances affects the time to compute the search results. However, the main memory usage of the ontology reasoners did not allow increasing the number of service descriptions further. Therefore, we repeated this experiment on a more powerful computer in Setup 2.

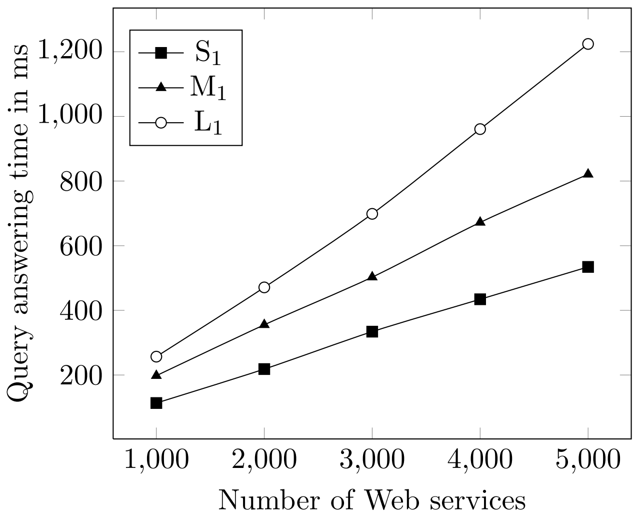


Figure 5.6: Setup 1: Mean query answering time with increasing number of Web service descriptions for three query sizes

Table 5.2: Query sizes tested in the experiment (setup 2)

	Small (S_2)	**Medium (M_2)**	**Large (L_2)**
Variables	6	9	12
Relations	9	12	15
NFRs	2	4	6

Setup 2

We measured the mean query answering time of the reasoners on a quad core Xeon CPU (2.33GHz) powered machine with 48GB main memory. We were able to load and reason over 30,000 Web service descriptions on this machine.

We also increased the query complexity in order to measure query answering times for even more complex use cases. Queries of three different sizes were sent to each knowledge base, as listed in Table 5.2. Small (S_2), medium (M_2), and large (L_2) conjunctive queries with respectively 6, 9, and 12 variables and 9, 12, and 15 properties within the desired start and end state descriptions. Furthermore, each of the queries contained 2, 4, and 6 NFRs, respectively. As depicted in Figure 5.7, the time to answer these queries ranges from 2.8s, 4.2s, and 5.0s for 5,000 service descriptions to 17s, 23s, and 33s for 30,000 descriptions for small, medium, and large sized queries, respectively.

Note, the purpose of Figures 5.6 and 5.7 is to show the feasibility of the presented discovery approach. It is clear that the query answering time highly depends on size and structure of the used domain ontologies, size and complexity of the query and service descriptions.

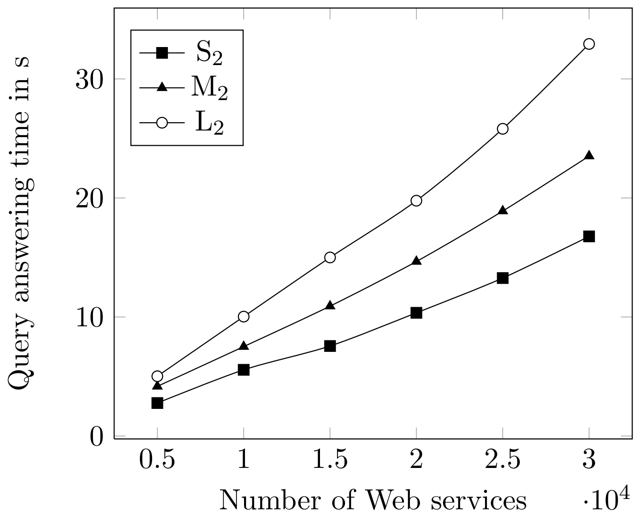


Figure 5.7: Setup 2: Mean query answering time with increasing number of Web service descriptions for three query sizes

Discussion

In order to assess the results of our experiments, we want to provide some references to related semantic discovery approaches that provide performance results. Such results comprise time measures for the computation of the desired service functionalities. We want to emphasize that results, especially the query answering time, cannot be easily compared. The usage of different hardware, the varying size of semantic service descriptions, requests, and domain ontologies are technical reasons that lead to the incomparability of different approaches, at least with respect to the query answering times. The use of standardized benchmarks or so-called discovery challenges, e.g., the Semantic Web Service Challenge,[10] [GCGPP$^+$08] can often overcome these problems. However, the use of different models, expressivity, and complexities disallows the comparison of different discovery approaches based on their query answering times. Approaches that differ in this sense provide different capabilities and probably aim at different use cases. However, at this stage, we can show the feasibility of the presented discovery approach.

Clearly, the measured query answering time depends on size and structure of the used domain ontologies as well as size and complexity of the query and service descriptions. The current implementation did not focus on efficiency and scalability, yet. Nevertheless, these results can be significantly improved by various options, such as introducing indexing structures, increasing the computational power, and distributing the reasoning process [Boc08].

In related works, the measured query answering time of alternative Semantic Web Service discovery approaches were reported. For example, [KMP06] provides experimental results from a test bed related to the European INFRAWEBS project. The authors conclude that matching a WSML Goal against semantic service descriptions scales up to 1,000 service descriptions while it returns results within 5 seconds.

[10]Semantic Web Service Challenge – http://sws-challenge.org, retrieved 2013-06-15

Stollberg et al. report mean query answering times of 72 seconds against 2,000 Web service descriptions when using a naive approach that they reduced to 0.3 seconds by using semantic discovery caching (SDC) [SH07]. As we will further discuss in the related work section of this chapter, the SDC technique relies on a precomputed caching structure within a global hierarchy of goals. The query is answered by a simple lookup which leads to the fairly fast response time.

Another example for an approach that relies on a precomputation phase is provided in [Lar06]. An experiment shown in [Lar06] with 2,000 Web service descriptions measured a classification phase with more than 160 seconds and query answering time of about 20ms.

5.6 Related Work

The development of automated methods for the discovery of services is a fundamental enabler for the development of intelligent systems based on services. Quite a few semantic service modeling and discovery approaches have been proposed in the past. Although extensive research has been carried out in the field of Semantic Web Service discovery, we initially identified requirements on a discovery method that are not completely fulfilled by existing approaches. The inadequacies of existing Semantic Web Service discovery approaches can be best observed when considering our scenarios and the requirements that can be derived from them. Often, the related approaches were developed for human use, which benefits from more flexible matchmaking methods and less expressive request languages that can be easier presented to end users. Other discovery approaches that were intended for automated use, e.g., for an automated service composition, often provide less expressive formalisms for service descriptions and requests than our approach, and aim at the development of tractable solutions of the composition problem. It is important to keep in mind that when we relate our approach to existing ones, we will also highlight why they are not the best solution applicable to our scenarios.

Service Profile Descriptions

Service profile descriptions haven been broadly applied to describe the functionality of Web services. We explain how different approaches or service modeling frameworks describe the functionality, including state changes if applicable.

The Semantic Web community with focus on languages provides description logic based approaches [BHL$^+$05, SPAS03, GCTB01]. The approach by Li et al. in [LH03] represents objects like inputs and outputs as concepts in description logics. This approach further combines the use of DL with DAML+OIL and DAML-S and defines different matching degrees. Service description and request are similarly structured comprising inputs, outputs, preconditions, and effects. Discovery, i.e., matchmaking, is based on the intersection of service offer and request and is reduced to checking subsumption of input and output types. However, DL-based approaches fail to reason about the dynamics of Web services, since DL reasoners cannot reason about changing knowledge bases. Consequently, more recent research activities concentrate on more detailed formalisms, for instance the state-based perspective on Web services that is discussed below. These models allow modeling the dynamics of Web services.

The OWL-S Service Profile comprises three pieces of information describing (i) the service provider organization, (ii) the function that the service computes, and (iii) a host of features that specify characteristics of the service [MBH$^+$04]. Within the scope of this chapter, we focused on information that can be expressed by the OWL-S functional description of a service. OWL-S proposes to model Web services semantically with inputs, outputs, preconditions and effects [MBH$^+$04, SPAS03]. The set of inputs is required for (successful) execution, and the precondition must be true from the provider's perspective in order to (successfully) execute the service. The expected effects result from the execution, e.g., stating that a credit card was charged. The parameters and the condition and effect expressions can be described in the process model and referenced in the Service Profile. Preconditions and effects are logical formulae that can be expressed in any language. The Service Profile represents both service offers and requests [MBH$^+$04]

Changes caused by the service executions are described in the effects element. However, as changes are expressed by logic expressions (e.g., using KIF conditions, SWRL rules, etc.), it can be expressed that a fact can be assumed to be true after the execution. However, it remains unclear if this fact was true before the execution or not. Furthermore, it is hard to use logic formulae to state that something was updated or deleted by the execution. In order to overcome this problem, we added a type to each change description. Each change in our model can either add, update, or delete instances in the knowledge base. We furthermore describe the end state of the service execution by the expression ψ in our model of the service functionality. An end state can become relevant in the discovery phase. This is because it models the relationships between input and output parameters and relationships between outputs and other parameters that were created during the execution. End states cannot be derived completely from the OWL-S Service Profile description due to the frame problem.

In addition, OWL-S allows describing under what conditions the outputs are returned. For this, the concept process:Result is used and alternative traces of a service can be expressed. We did not consider alternative configurations, traces, and failure handling in our model of service descriptions. Still, our approach can be extended to embrace conditional outputs as proposed in OWL-S.

The Web Service Modeling Ontology WSMO provides four aspects related to Semantic Web Services: Web services, ontologies, goals, and mediators. WSMO Web service capabilities are part of the service descriptions. They describe the functionality of a service as capabilities of the provider and are modeled by the preconditions, assumptions, postconditions, and effects. Preconditions and assumptions describe the requirements for a correct service execution, which then causes that postconditions and effects result from it [ABK$^+$04]. WSMO introduces shared variables that are universally quantified over WSMO assumptions, preconditions, postconditions, and effects in order to relate preconditions and assumptions with postconditions and effects.

According to the WSMO specification in [dBBD$^+$05], the precondition corresponds to the start state description from the perspective of the service provider. The assumption describes the start state of the world in general. Although it can be problematic to describe the world, this description element can capture the conditions on the consumer side for a correct execution. While the consumer perspective can be useful for a requester, these conditions cannot be validated by the provider, because providers cannot access the consumer's knowledge. In our approach, we therefore omit the description of conditions on the consumer's side. Instead, we express such constraints in the start state description, too, if the service provider checks the correctness of the assumptions on the input parameters of a service invocation. The WSMO postconditions describe the provider knowledge after the execution. This view corresponds to our end state description ψ. Furthermore, the WSMO "effects describe the state of the world after the execution of the Web service" [dBBD$^+$05]. Apart from the possibility to describe the effects on the consumer's knowledge, which is captured by the end state description ψ in our model, the changes caused by a service execution cannot be entirely reflected by a state description. In order to highlight this distinction to our model, we used the term 'changes' and modeled changes differently. In comparison to the WSMO model, we additionally describe how the service execution led to the conditions that hold in the end state. It allows for a description and interpretation of more details of the service functionality.

Recently, WSMO-Lite has been proposed for describing Web services as the next evolutionary step after SAWSDL. WSMO-Lite fills SAWSDL annotations with concrete semantic service descriptions [VKVF08]. The WSMO-Lite ontology is on one side lightweight and provides on the other side elements for modeling the functionality of Web services. WSMO-Lite does not contain ontology elements for adding input and output parameters to service descriptions explicitly. Instead, the free variables in preconditions and effects are considered to be input and output parameters, respectively. Note that such a derivation is not possible if a formula does not have any free variables but the Web service has inputs or outputs.

Semantic Web Service Discovery

The matchmaking of DL-based service discovery approaches that were only considering input and output parameter types in service descriptions and requests is computed by subsumption reasoning. A matching technique for stateless Web services is proposed in [HZB$^+$06]. This approach is restricted to conjunctive queries, since the query containment problem is decidable for such queries. The OWL-S matchmaker [PKPS02] uses OWL-S Service Profile for describing offers as well as requests. However, these approaches have in common that a dynamic world is modeled with static languages and logics. Consequently, they cannot model the functionality of state changing services, which is required in realistic service discovery use cases. The OWL-S Matchmaker [SPS04] as well as other known Semantic Web Service discovery approaches also lack support for the specification of desired and undesired changes in requests.

The discovery approach in [MPM$^+$04] is based on OWL-S and describes service functionalities semantically by inputs, outputs, preconditions, and effects. This approach interprets preconditions as constraints that need to be satisfied for the service requester only and effects as side effects of the service execution on the world. The given perspective supports the discovery use cases in which users may search for services based on the requirements that they have to fulfill. However, it can be argued whether client-side conditions are relevant as they cannot be evaluated during service invocation and execution time (by a provider or an execution engine with limited access to the client's knowledge base). Additional conditions that have to hold on the requester's side can support users, but service providers cannot rely on them. In our approach, we model conditions that hold at the service provider's side since those conditions can be evaluated during service invocation and execution time.

The state-based service discovery approach in [SHH07, SHF11] uses a state-based formal model of service descriptions [KLS06]. The functionality of a Web service is formally described by the set of possible Web service executions while each normal execution of a Web service is determined by its start and end state. Preconditions and effects constrain the start and end states, respectively. A state describes how the world is perceived by an external observer. This implies that preconditions and effects may contain all observations that can be made. In their works, they only restrict the amount of information to those relevant for the use of the formal model.

Their discovery approach applies the WSMO description framework. In [SHF11], they substitute the shared variables in the precondition with their pre-variant. The pre-variant variables are fluents, which can have different values in different states, and are obtained by substituting the original variable name by a new unique name. It is claimed that this approach allows for reasoning with traditional model-theoretic semantics. However, the relationship between dynamic symbols in effects and their pre-variant in preconditions still needs to be managed on top of a (description) logic reasoner. Although [SHH07, SHF11] addresses changes in the knowledge bases by the introduction of dynamic symbols, the discovery approach presented in [SHH07, SHF11] fails to reason about the dynamics of Web services. The discovery algorithm relies on the assumption that the precondition ϕ logically implies the effect ψ of a Web service execution. Modeling a transition as a logical implication $\phi \Rightarrow \psi$ can be problematic, e.g., in case of a Web service that deletes a certain fact, the existence of the fact would imply nonexistence of the fact, e.g., a user subscription would imply that the user is not subscribed anymore (as discussed Section 5.1.2 and shown in Example 11).

As dynamic symbols are used in the logical formulae ϕ and ψ, the (model-theoretic) implication semantics of $\phi \Rightarrow \psi$ is not wrong for changing knowledge bases during Web service execution. However, the state changing semantics of the service description is compromised due to the use of different symbols representing (consequently different) individuals in different states. For instance, if the balance $balance(acc)$ of an account acc changes during Web service execution to $balance_{\psi}(acc)$, then a withdrawal of an amount of 150 can be expressed by the relationship between both balance values $equal(balance(acc), balance_{\psi}(acc) - 150)$. However, by the introduction of a second symbol $balance_{\psi}$ to the knowledge base, the uniqueness of the symbol $balance$ is lost.

Hybrid matchmakers like the ones presented in [KFS09, KK12] employ logic-based matchmaking techniques in conjunction with information retrieval techniques. The latter techniques promise to improve the discovery process with faster results and also increase the precision and recall of the discovery results. However, the result accuracy that we aim at is not given.

Service Request Specification

Goal-driven approaches like [KLL$^+$05, SHH07, LCC08, SHF11] do not explicitly specify inputs as parts of the goal. However, a goal needs to be mapped to a request for finding appropriate Web services. In such a request, constraints on inputs can be useful, in particular if a user wants to relate the produced outputs to the provided input parameters or wants to exclude a particular input parameter. In goal-based approaches, goals are mapped to predefined goal templates that are used to find appropriate Web services. One major difference between our approach and goal-based approaches is that we interpret inputs, outputs, start and end state descriptions differently from descriptions of changes and requests. Namely, we interpret the former as a pair of states and the latter as a pair of queries.

Non-Functional Properties

The discovery approaches that we mentioned so far did not consider non-functional properties for Web service discovery. In OWL-S, non-functional properties are considered as human-readable metadata, e.g., a service name.

The need for the inclusion of non-functional requirements expressed in a request has been discussed by the group developing the WSMO specification [ABK$^+$04, OLPL04]. WSML [dBFK$^+$08] does not include non-functional properties into the logical model. Consequently, no reasoning on them is possible. The WSMO specification defined non-functional properties. So far, however, there is no prominent implementation available that considers them. O'Sullivan et al. [O'S06] described a set of non-functional properties relevant for Web services and their modeling, which were formalized in a WSMO deliverable [TF06].

Semantic modeling [KP07] and matchmaking [KP09] of non-functional properties of services has been studied. An extension of existing service description models to provide semantics for the non-functional service parameters in order to overcome the limitations of the syntactical quality of service descriptions was suggested. The mentioned works focused on the modeling of the non-functional property structure, e.g., the entire metric with units, value types, and information about the conducted measurements, which is beyond the focus of our work. However, in [KP07] the authors give recommendations in order to extend existing discovery techniques based on the requirements on a service description model they identified. In [KP09], they introduce NFP-aware service matchmaking based on mixed integer programming, which leads to the separation of a logic-based capability matchmaking and the evaluation of non-functional requirements. Such separation is not desired in our approach, as we target rich requests that embrace functional and non-functional requirements uniformly.

5.7 Summary and Conclusions

In this chapter, we focused on the application and extension of the discovery method developed in the previous chapter to atomic service. Our aim was to support the modeling of the service functionality, which can be considered as a profile of more generic service behavior descriptions. The service functionality is a central property of atomic services that implement a single-request-response pattern. Examples for atomic services are classic Web services and Web APIs that constitute the central element of service-oriented architectures and the center of attention in the Semantic Web Services community.

We applied our property-based service model from Section 4.1.1 in order to establish a comprehensive model for service descriptions including a high flexibility for a future extension by further properties. Following the model checking driven matchmaking approach, we developed a model to express constraints on the functionality of atomic services in requests.

The developed discovery method for atomic services builds on the description and request models. We presented a formal interpretation of each sub-property of the requested functionality and presented the matchmaking technique that evaluates the requests over the state-based representation of service functionalities.

In Section 5.5, we presented details on the implementation of the discovery method. We also showed how the developed service discovery engine was integrated and aligned to existing Web technologies that were used within the SOA4All framework. Based on this implementation, the performance of the developed method was measured and its feasibility for discovery over up to 30,000 Web service descriptions was verified.

Appropriate formalisms for the description and discovery of atomic services have been provided. However, the evaluation results clearly showed that such an expressive approach might not scale up for millions of services as envisioned in [Abe09, DFD$^+$09, Zuc10]. Because scalability and efficiency are crucial to enable Semantic Web Service discovery on a large scale in the Web, we will further focus on improving the discovery performance. By the introduction of index structures and a materialization of discovery results, we expect to handle larger sets of Semantic Web Service descriptions. More specifically, we identified parameters that influence the complexity adversely and propose a classification-based technique to reduce the complexity of service discovery in the next chapter. The main idea is to assume a classification of functionalities and precompute whether a service is a match for a class. We will examine the potential to scale for a large number of services in further performance evaluations of subsequent chapters.

With respect to the existing works in the area of Semantic Web Service discovery, we have shown that we can apply a matchmaking technique that is based on model checking. We presented a discovery method for atomic services that seems more practical as it relies on a request formalism that is close to the intuitive interpretation of a search request. It further allows for a precise specification of wanted and unwanted properties. Both features are in contrast to intersection-based matchmaking approaches. Therefore, we investigated the problem of using the same formalism for service descriptions and requests, especially in the context of atomic services, in Section 5.1 thoroughly.

In order to fulfill the requirement for the accuracy of search results, our discovery method does not compute relaxed matching degrees that correspond to certain degrees of intersection of the set of service execution runs as illustrated in Figure 4.6. Instead, we provided formalisms that allow users to clearly specify a query with unambiguous interpretation. If a query does not deliver any results, then query relaxation or manual query refinement can be applied. The former method can thus simulate a subsume or intersect match since a less restrictive query will be able to return services that would match a query by subsume or intersect match.

Chapter 6

Classification of Services

Prominent service modeling frameworks such as WSDL-S, SAWSDL, OWL-S, WSMO, and WSMO-Lite have proposed to model a classification of services explicitly as an efficient mean to retrieve services and for implementing the service matchmaking, which can become very complex otherwise. However, these frameworks only provide a reference element without specifying how these classes are defined. Hence, the relation of a service classification to other explicitly modeled service properties remains unclear.

In this chapter, we present a formal underpinning of *service classes* by viewing them as a set of services that fulfill a logical combination of constraints on functional and non-functional properties. A hierarchy of service classes is automatically derived from their formal definition and can be exploited for an efficient service discovery. In addition, we show in this chapter how service classes can be used (i) to create service descriptions without specifying precise property values and (ii) to create service requests that can use service classes to easily express ranges of desired property values.

A service classification aims at improving the discovery efficiency by an early reduction of the search space. The number of properties that are verified at query time and the number of expensive DL reasoning tasks of a matchmaker are reduced. Hence, our discovery method is able to evaluate requests faster in comparison to the same method that does not exploit a classification. As a consequence, our discovery method can treat larger service repositories, while maintaining an acceptable query answering performance, which is required for the aimed use cases. At the end of this chapter, we investigate the expected performance gain of the classification-based discovery approach.

The classification of services promises a significant performance gain for the discovery task, especially when rather complex services, i.e., Web-mediated services with complex interaction patterns, are included. Here, the discovery efficiency becomes even more critical, because (i) there are a considerable number of complex services available that is typically used by end users, and (ii) evaluating the constraints of a request on services with complex behavior is computationally more complex. Therefore, we will apply the concept of meaningful service classes, which provide a formally specified semantics, not only to atomic services but also to complex services. We focus in this chapter on the behavioral properties of services and their representation in service classes. Non-functional properties can be used in a similar fashion, but as non-functional properties with a simple structure (like the price of a service) can be efficiently matched anyway, this chapter focuses on the service behavior, as the performance gain by the introduction of a classification is more decisive for this aspect.

Our approach of a service classification with meaningful classes has been presented in [AJ11, JAS12]. The former publication introduces classes of atomic services and considers the functionality as well as non-functional properties. In [JAS12], we focused on the classification of the behavior of complex services.

6.1 Motivation of Meaningful Service Classes

In the Web there are a large number of (business) services with complex behavior, such as e-commerce websites that require multiple interactions with the user, as well as an increasing number of Web automation scripts to coordinate the execution of multiple complex services. However, while there are quite a few search techniques for atomic services, search techniques for complex services are still rare and only foundational. In this section, we present *behavior classes* that have formal semantics as well as human comprehensible names in order to foster the usability of specification of constraints, and the efficiency of search for complex services and processes. Our approach enables automated methods for (i) assigning behavior classes to complex behavior descriptions, (ii) checking consistency of such a classification, and (iii) computing behavior class hierarchies. Furthermore, human comprehensible names for the behavior classes increase usability by allowing for shorter service descriptions and requests. Our evaluation results prove that a behavior class hierarchy can be exploited as an indexing structure and show a significant improvement of the achieved search performance.

A large number of services are offered in the form of websites that have complex behavior (multiple user interactions that may even depend on the user input at previous interactions of the same process run etc.). Furthermore, there are often multiple websites offering the same or similar functionalities and information. In order to gain a broader overview of the desired product, information, or functionality, a user often needs to follow several paths on various websites by providing the required inputs at appropriate time, and accepting the (intermediary) outputs.

Different websites offer different granularity of information and functionalities and have heterogeneous navigation paths. Logical dependencies between the information and functionalities provided by different websites affect the order in which they are executed by a user. This makes it difficult for end users to coordinate the execution of various websites, and aggregate the information gathered from them. Web automation scripts, originally developed for the purpose of testing websites by developers, are turning out to be promising for end users as well, since they can automate this tedious process to a large extent. However, finding and composing Web automation scripts remains very difficult, since it requires a lot of manual effort due to the huge gap between the user requirements and the functionality offered by existing search techniques. In order to find and compose complex executable models, users need to be able to search by constraining the behavior of the models as well as the information they require and deliver at various stages during their execution. The search for websites and Web automation scripts is typically based upon syntactic matching of keywords with the content at the surface Web (as opposed to the Deep Web [Ber01]), or with manually added tags of the script respectively. Even though the tags could hint at the functionality of a script, tagging requires manual effort and is often faulty [BW99]. In order to equip users with the power of finding, creatively combining, and executing websites for emerging more and more sophisticated use cases, a process-oriented view on websites is required. Even if the syntactic keyword-based approaches were extended to support the process-oriented view, they would still be restricted to support end users only. If an execution engine encounters problems like a failing service during runtime, it is often desired that it finds and composes an alternative solution automatically, and resumes the execution. However, the requirements for an alternative are known to the system in the form of structural constraints that have a formal semantics in the first place, and not as ambiguous natural language keywords. Formalisms available for modeling complex distributed processes and automated reasoning about them, including our work of the previous chapters, are neither easy to use for end users, nor do they exhibit acceptable performance and scalability for practical purposes. In this chapter, we propose a way out of this problem by introducing the notion of service classes that have human comprehensible names as well as formal definitions.

Most service description frameworks include the concept of service classifications in their model. E.g., the OWL-S Service Profile has the notion of ServiceCategory and serviceClassification, WSDL-S provides the element category, and the WSMO-Lite ontology defines a class named FunctionalClassificationRoot, which is used as a super-class of any domain-specific service class. The common approach is to link the ontology individual representing the service to one or more externally defined classes.

The service description frameworks assume that a taxonomy of service classes is given. Standardized classifications like the UNSPSC, for instance, were recommended for this purpose by UDDI, OWL-S, and WSDL-S [UDD01, MBH$^+$04, AFM$^+$05]. Services can be assigned to these classes by annotating their service descriptions accordingly. This approach is attractive because it is as easy as tagging and provides the advantage that it can be exploited for an efficient service discovery, as a request formalism composed out of desired class identifiers remains trivial.

Although the classes can be embedded formally in a taxonomy, for example defined in an RDF/S ontology that provides the formal subclass-of relationship to define class hierarchies, the classes do not provide a formal meaning regarding the behavioral, functional, or non-functional description of a service that is implied by such a classification. That is, assigning a service to a class ideally implies that certain functional or non-functional service properties fulfill the constraints, which are represented by the actual (informal) meaning of the class. However, the meaning is often not explicit.

For instance, if a service is assigned to the class of book selling services, which is named BookSales, nothing is stated about the common functionality of book selling services like the provision of a book at the end. Only users can interpret the meaning of classes by their name. Similarly, assume that the class BookSales is in a subclass-of relationship to another class named Sales. The subclass-of relationship makes explicit that BookSales is a refined class of Sales and that services of the class BookSales fulfill equal or more constraints than the services in Sales. However, as the classes are not defined, machines cannot automatically classify services or detect inconsistencies of a classification or the subclass-of relationships. The service class semantics is not machine-interpretable and human intervention is required in order to assign services to classes or to find desired classes.

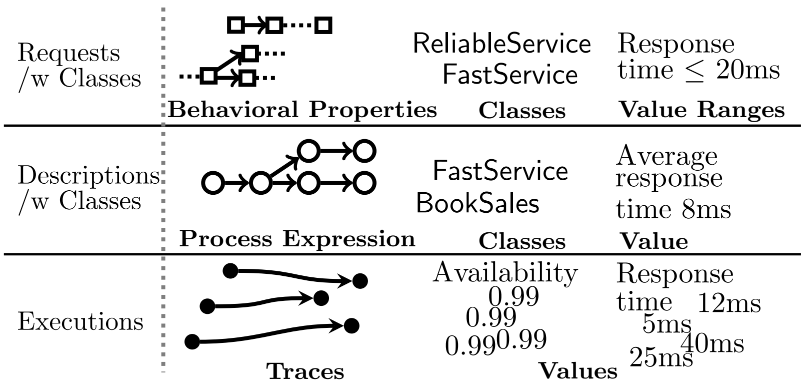


Figure 6.1: Service classes in different layers of abstraction: Discovery requests (top layer), service descriptions using explicit and implicit values (intermediate layer), and the formal interpretation of the service model (bottom layer).

6.2 Classification of Services

Although it can be more intuitive for end users and service developers to annotate services with given service classes compared to creating logical descriptions of the behavior or the functionality, service classifications cannot replace explicit descriptions in general. The accuracy of service discovery may suffer from classifications with coarse-grained class definitions, which is the case if not all specifics of a particular service are expressed by the given classes. This is due to the intrinsic nature of classes to represent commonalities of services and, so, classes of a classification have to be created for general purpose use. Otherwise, there has to be a separate class for every service. Nevertheless and as we will show, classes can effectively complement comprehensive service descriptions and requests.

6.2.1 Implicit Description of Service Properties

The property-based service model is used to formally describe services when concrete values of the functional or non-functional properties are known and the service description modeler is willing to publish them. However, in many cases the concrete value of a service property is not known or not supposed to be mentioned explicitly. However, it may still be possible or desired to specify a range of concrete values.

Furthermore, an explicit description of property values does not support negations, which means that an explicit description does not allow for an exclusion of properties. Therefore, since ontology languages have open world semantics, an ontology reasoner (e.g., HermiT) will not find any matching service descriptions if the request specifies the negation of some service property.

We introduce the notion of a service class to address the above drawbacks of explicit service descriptions. First, we show what a service class means formally by relating it to the formal model of a service. Then, we present how service classes can be described with OWL. We will then discuss how properties of services can be specified implicitly with the help of service classes.

Our contribution can be outlined with the support of Figure 6.1. Our formal model of services corresponds to the bottom layer of the figure and represents the interpretation of service descriptions with service instances, their executions runs (traces), and observed values of the non-functional properties. For example, in this figure, explicit service descriptions can be created using these models for the capability expression and the concrete value of the property "Availability". We introduce the service class formalism that is used to create implicit service descriptions that are reflected by the service class named FastService in the middle layer of Figure 6.1. A request, as depicted in Figure 6.1, describes a set of services, where each one may have different characteristics, e.g., a particular property value, if the request specifies a range of desired property values. As we will show, we apply the service class formalism to service requests. Then, we further present an implementation of our approach.

6.2.2 Service Classes

A class formally describes a set of services that have certain service properties within a common range. The classes can be based upon functional and non-functional property constraints uniformly. We now introduce the notion of service classes. We advance state of the art approaches with a hybrid description formalism for classes and highlight the benefits while discussing their use in service modeling and discovery.

A *service class* is formally defined by a constraint Φ_c and a class name (label) c. The class represents a set of services that share common properties, e.g., behavioral properties, declared by the class definition Φ_c. The name is a human comprehensible textual representation of the asserted attributes and is solely used for the purpose of increasing usability. Named classes have been widely used in taxonomies of products and services (e.g., UNSPSC) and are a fundamental concept in ontologies [SS09] in order to abstract from particular ontology concept and role definitions.

In contrast to service class taxonomies without formal class definitions, our formal service classes allow for an automated and consistent classification of service descriptions into existing classes. Also, tools are able to reason about the attributes asserted by a manual class assignment. The formalism used to define service classes corresponds to the request model that we introduced in Section 4.2. In short, it allows expressing combinations of desired functional and non-functional properties, where a set of acceptable values is attached to each property.

Definition 12 (Service Class Definition). *Based on the property-based service model, a service class c is a finite set $P$of service properties, with each property $p \in P$ associated with a set V_p that denotes the range for the values of the property p.*

Informally, a service class describes a set of services by specifying constraints that are satisfied by the values of functional and non-functional properties. More formally, a service class as defined in Definition 12 describes a set of values assigned to a property. It is interpreted in the same way as a service request, see Equation 4.4 on page 70.

Due to its complexity, we want to introduce service classes with the focus on the service behavior in the following. We use the term *behavior class* in order to highlight this restriction.

Definition 13 (Behavior Class Syntax). *The specification of behavior classes is based on the property specification language introduced in the preliminaries in Section 3.3.2. For class* *definitions, we extend the given syntax of property constraints by a choice for the inclusion of class names (C) as follows. Of course, every class name $c \in C$ has its defining formula Φ_c.*

$$\Phi ::= C \mid \Phi \wedge_\mu \Phi \mid \neg_\mu \Phi \mid \mu X.\Phi(X) \mid \langle a \rangle \Phi \mid P \mid \textbf{true} \mid \textbf{false}$$

The above definition shows the minimal syntax. Disjunction ($\vee_\mu$), greatest fixed point ($\nu X.\Phi(X)$), and universal quantification of actions ($[a]\Phi$) can be derived from the above language constructs. Of course, we keep the extension to add DL expression to the property specifications such that the variables and resources can be semantically described.

Example 12. *A class named "WebBookSelling" describes the generic behavior of Web-based book selling services. The class definition expresses that author and title or the ISBN of the desired book have to be provided by the consumer, i.e., for any traces, one of the following two input action must take place.*

$$\varphi_{initial} \overset{\mathsf{def}}{=} [\mathsf{http}[\mathsf{author} : \mathsf{ex{:}Author}, \mathsf{title} : \mathsf{ex{:}Title}]]P \vee_\mu [\mathsf{http}[\mathsf{isbn} : \mathsf{ex{:}ISBN}]]P$$

The proposition P describes input parameters and their relation to an individual representing the desired book. P may comprise the following propositions ex:Book(product), ex:hasAuthor(product, author), *and* ex:hasISBN(product, isbn). *We furthermore define in φ_{result} that it should be possible that a book is eventually returned.*

$$\varphi_{result} \overset{\mathsf{def}}{=} \textbf{eventually } \langle \mathsf{ch}\langle \mathsf{product} : \mathsf{ex{:}Product}\rangle\rangle \varphi_{final}$$

Members of the behavior class WebBookSelling must eventually return the specified book product *via a communication channel* ch. *After the output took place, the following proposition φ_{final} must hold.*

$$\varphi_{final} \overset{\mathsf{def}}{=} \Big(\mathsf{ex{:}DropShiftDelivery(delivery)} \vee \mathsf{ex{:}OnlineDelivery(delivery)}\Big) \wedge \\ \mathsf{ex{:}DeliveryItem(delivery, product)}$$

It states that the product *is either shipped via surface mail or email. In summary, we can define the overall behavior of the class by $\Phi_{c_{wbs}}$ in the following.*

$$\Phi_{c_{wbs}} \overset{\mathsf{def}}{=} \varphi_{initial} \wedge_\mu \varphi_{result} \wedge_\mu \varphi_{final}$$

Definition 14 (Behavior Class Semantics)**.** *The semantics $[\![c]\!]_\mathcal{V}$ of a class c in a constraint is defined over the LTS $L = (S, \rightarrow, A, \lambda)$ of a behavior description and corresponds to the set of states that fulfill the constraint Φ_c. That is, $[\![c]\!]_\mathcal{V} = [\![\Phi_c]\!]_\mathcal{V}$. We say that a service or process ω is member of a behavior class c, iff the start state $s_0 \in S$ of the LTS L_ω is in the set of states $[\![\Phi_c]\!]_\mathcal{V}$ that comply with the constraints of Φ_c under the valuation function $\mathcal{V}$.*

The use of formal service classes provides the following benefits to our discovery method.

(1) A formal class definition allows for automated categorization of semantically described services into given classes. The class hierarchy can be automatically derived from the definition of individual classes. Inconsistencies in existing service categorizations and contradictions between semantic service descriptions and their classification can be automatically detected and prevented.

(2) Service classes can be used to express service requests. A request either formalizes requirements from scratch or alternatively reuses given class definitions. As available service descriptions can be classified automatically due to formal service class definitions, the classification of services can be precomputed and services of a requested class can be directly retrieved, which in turn leads to a more efficient service retrieval task as the search space can be reduced.

(3) Service classes can also be used to create service descriptions when precise property values are not known or should not be revealed. For instance, if the precise response time of a service cannot be determined, a class FastService can be defined such that a service of this class features a response time of less than a second, and the service can be classified into this class to express the responsiveness implicitly.

6.2.3 Behavior Class Hierarchy

A class hierarchy $\mathcal{T} = (C, H)$ is a graph structure with a finite set of behavior classes C and the subclass relationship $H \subseteq C \times C$ over classes as the only relation. A class $c_i \in C$ is subclass of $c_j \in C$, we say $(c_i, c_j) \in H$, if all services that are member of class c_i (i.e., model of the formal definition Φ_{c_i}) are also member of class c_j (i.e., model of Φ_{c_j}). H is a partial order that is not limited to a tree structure; meshes are possible. The most generic service class is a class that poses no constraints and contains all services.

Given a set of behavior classes C, a hierarchy $\mathcal{T}$ is derived automatically based on class definitions Φ_c of each class $c \in C$. The most abstract class that describes any service represents the root node of the hierarchy. The more specific a class is, i.e., the more constraints a class definition contains, the further the node representing the specific class will be from the root. We compute the relation $H \subseteq C \times C$ by comparing pairs of class definitions Φ_{c_i}, Φ_{c_j} and determining the subclass relationship by adopting the Kozen's axiomatization technique from [Koz83]. The axiomatization provides the basis for checking whether a formula is a sub-formula of another formula. We make use of these rules in order to compute the subclass relationship H and keep the classification hierarchy consistent.

First, we replace class names in Φ_{c_i} and Φ_{c_j} with their formal definition. Then, μ-calculus expressions are transformed into the positive normal form and the relationship between both expressions is determined. In fact, a syntactical containment relationship between μ-calculus expressions is computed by this method. It was shown that this method determines whether any model of one expression Φ_{c_i} is also model of another expression Φ_{c_j}, which consequently means that class c_i is a subclass of c_j. In [Wal96], the completeness of the axiomatization of the proposition μ-calculus presented in [Koz83] has been shown. As this method is applicable to plain μ-calculus expressions only, we extended it such that the DL-based descriptions of variables and resources in the property descriptions of the class definitions are considered accordingly.

First of all, we apply the unique name assumption to the variables and resources. It allows us to refer to equivalent (ontology) individuals by their unique names. Then, for any semantic expressions about actions, their parameters, or state propositions of a class definition, we check whether it is the consequence of, i.e., is entailed by, the corresponding expressions of the other class definition. That is, we check if a proposition P_i from Φ_{c_i} is entailed by the corresponding proposition P_j from Φ_{c_j}. If so, $(c_i, c_j) \in H$ holds. As we can determine the subclass relationship between service classes, we can compute the service class hierarchy based on the relation H over classes automatically.

Updates like the insertion of new classes to or the deletion of existing classes from an existing hierarchy are automatically managed due to the formal class definitions. New classes are automatically inserted by computing the subclass relationship to existing classes in depth first search. During this process, visited classes need to be marked to avoid redundant checks due to the meshes in the hierarchy. This way, the sub- and super class relationships between the new and existing classes are determined. Then, a new class is inserted as a subclass of the most specific super classes of the hierarchy. Also, the inserted class can be super class of existing classes. These relationships are identified by comparing the new class to classes from the bottom of the hierarchy. Changes in class definitions are treated analogously to inserting new classes. When a class is removed from a hierarchy, all its subclasses become direct subclasses of its super classes.

6.3 Annotation of Behavior Descriptions

One of our contributions in this chapter is the extension of the behavior description formalism with the ability to annotate behavior descriptions with further constraints that cannot be expressed in *suprimePDL* process expressions (using π-calculus combined with DL). We start motivating the need for behavior annotations and present their semantics afterwards.

Motivation. Behavior descriptions are suitable to describe complete processes. All observable details of the modeled system need to be known in order to describe an executable process expression based on the closed world assumption. This can be done by the provider, e.g., of an enterprise system, who can derive the detailed facts from the implementation. However, it is not realistic that any implementation details are known to other users or consumers. E.g., it is not important and perhaps not possible to state which particular SSL certificate a book selling service uses for the payment process. However, the information that the service supports SSL encrypted communication is relevant, but cannot be expressed at the ABox level of process models where concrete certificate instances can be described. Rather, it needs to be expressed on a logical level by using existential quantifiers, e.g., declaratively expressed in the constraint formalism.

In contrast to behavior descriptions, the constraint language allows for the exclusion of behavioral properties. Side effects that are not observed by a user during execution can also be expressed by the constraint formalism. E.g., our π-calculus-based behavior descriptions cannot express that a service will never pass user-provided payment information to any other party.

Because it is typically impossible to observe complete service behaviors in the Web and as there are behavioral properties that cannot be captured by a process algebra, we extend the behavior description formalism by declarative constraints. Note that modeling these properties is relevant because it allows service consumers to search for services based on these properties.

Annotation. We allow for annotating behavior descriptions with a set of behavior constraints using the constraint formalism introduced in Section 3.3.2. An ordinary behavior description π_1 is interpreted by an LTS L_{π_1}. Let π_2 be a behavior description that is annotated with a behavior constraint Φ, we say $\Phi(\pi_2)$. Then π_2 is interpreted by $L_{\pi_2} \in \mathcal{L}_\Phi$, where $\mathcal{L}_\Phi$ denotes the set of LTS that fulfill the constraint Φ. That is, the set of start states of the LTS in $\mathcal{L}_\Phi$ equals the set of start states identified by $[\![\Phi]\!]$.

Using our model checking based matchmaking technique, as introduced in Chapter 4, to check whether the constraints of a class are fulfilled by a behavior description, the consistency of class annotations is automatically assured if no contradictions exist.

A behavior description π can be annotated with several behavior class constraints $\Phi_1, \ldots, \Phi_n$, where $n \geq 1$, simultaneously. The semantics is defined as follows.

$$\Phi_1(\pi), \ldots, \Phi_n(\pi) \Leftrightarrow L_\pi \in \bigcap_{1 \leq i \leq n} \mathcal{L}_{\Phi_i}$$

Example 13. *In Section 4.1.3, we presented an example description of a book selling service including its behavior. We considered six transitions during the execution, for instance, the input of a book's author and title in the beginning.*

A provider who wants to offer a similar book selling service needs to create a behavior description in one of the following ways. If she can disclose the complete behavior, then the description may be automatically derived from the implementation. Alternatively, the description is manually created from scratch. Another alternative is introduced through a given service classification. The provider searches for appropriate classes in the hierarchy by inspecting the class names and, if available, a documentation. The service class WebBookSelling as shown in Example 12 is a good candidate to annotate her service.

She annotates the service ω by adding the axiom $\mathsf{WebBookSelling(w)}$ *to the service description ontology $O_{\mathcal{D}_\omega}$, where* w *is the individual that represents the service ω in $O_{\mathcal{D}_\omega}$. The provider also aligns the variable names of the class definition to the names she uses in her own explicit behavior description. For instance, she could reuse the descriptions of the desired book, author, title, and delivery options from the class definition.*

Given the alignment of variables between explicit behavior descriptions and the class definitions, contradictions can be automatically detected by our matchmaking approach and, thus, prevented. If the properties of class definitions can be positively evaluated on the explicit behavior description, then the class annotations are consistent.

Simplification of Behavior Modeling

Modeling complex behavior as presented in our example in Section 4.1.3 can become tedious. The complexity and length of behavior descriptions as well as the modeling effort are reduced by the use of classes without compromising on the expressivity. In fact, as we described in Section 6.2.1, we gain the expressivity that allows describing properties implicitly by annotating service descriptions with service classes. As shown in the previous example, users who model the behavior can reuse existing classes in service descriptions to express that the service provides the property described by the class. Instead of a possibly very complex behavior specification, the service class annotations can be mixed with further explicit refinements for a specific behavior description.

In collaborative service modeling, only domain or modeling experts will have the skills to model the service specifics beyond the scope of behavior classes. Since users are able to identify classes from a hierarchy by their descriptive name without the need to understand its formal definition, behavior descriptions can be created without any knowledge about the underlying formalisms. Discovery of appropriate classes currently remains a manual task, including browsing of the hierarchy or keyword-based search, which is an additional but less challenging effort for users. Then, the identified classes can be further inspected in order to verify that they match the user's intention. In our prototype that we introduce below, the keyword-based search for class names is made more flexible by searching also for class names containing hypernyms and hyponyms of the given keywords. Therefore, we use WordNet as dictionary. Beyond this point, we will not further focus on matching class names since it is not related to the contribution of this work and was subject to extensive research in the domain of information retrieval, e.g., in [TGEM07, LT11].

6.4 Classification-based Service Discovery

6.4.1 Service Classes in Requests

The μ-calculus-based behavior constraint specification language constrains a desired behavior and, by this, spans a space of desired services. An offered behavior matches a request if the offer equals a member of the set of desired behaviors. A reduction of the request modeling effort and the simplification of request expressions is achieved by the reuse of behavior classes. The benefits we listed for the simplification of behavior descriptions apply to requests, too.

In order to reuse existing behavior classes for expressing requests, the requesting actor has to identify appropriate behavior classes from the hierarchy. Then, after inspecting the service class or classes, additional properties can be modeled explicitly and combined with the classes. The combined request comprising selected classes and explicit properties can be submitted to the discovery method. We use the extended behavior class syntax presented in Definition 13 to formulate requests. That is, in contrast to the discovery method presented in Chapter 4, the requests can include and combine desired service classes (referenced by their class names) just like any other desired property instance. The semantics of the request formalism remains unchanged as the class names can be replaced by their service class definition.

Example 14. *We show an example request $\mathcal{R}$ for a desired book selling behavior with an additional constraint on the acceptance of a credit card payment. Parts of the desired behavior are inherited from the class definition Φ_{wbs} of an existing WebBookSelling behavior class shown above.*

We extend the class WebBookSelling by adding the additional constraint that the shipping is free. The new class is named WebBookSellingFreeShipping and is defined by $\Phi_{wbsfree}$ as follows:

$$\Phi_{wbsfree} \overset{\mathsf{def}}{=} WebBookSelling \wedge_{\mu} \textsf{ex:hasPrice(delivery, 0)}$$

The individual delivery *was already used in the definition of the class WebBookSelling. So, we use the same identifier in the definition of the class WebBookSellingFreeShipping and in further constraints in order to refer to the same individual. In the following example request $\mathcal{R}$, it becomes evident how much more complex the request would be if no classes could be reused* *for expressing the same set of constraints (that is, the definitions of WebBookSelling and WebBookSellingFreeShipping would be needed instead of the class name in the request below). Therefore, we argue that using classes can tremendously simplify and accelerate the specification of requests. As shown, the request can be further refined if the search result set is too coarse-grained. Note, $\wedge$ denotes the logic conjunction with DL semantics while $\wedge_\mu$ was defined by the μ-calculus.*

$$\begin{aligned}\mathcal{R} \coloneqq \textbf{eventually} \Big(&\mathsf{ex{:}Seller}(\mathsf{seller}) \wedge \\ &\mathsf{acceptsPaymentMethod}(\mathsf{seller}, \mathsf{payMethod}) \wedge \\ &\mathsf{ex{:}CreditCard}(\mathsf{payMethod})\Big) \wedge_\mu \mathit{WebBookSellingFreeShipping}\end{aligned}$$

6.4.2 Discovery Based on Offline Classification

Now we exploit the behavior classes for an efficient search for complex service behaviors. Besides the advantages of the hybrid behavior class formalism we have already shown, this section presents how the search performance is increased by formal behavior classes.

A discovery engine verifies constraints for each offered behavior individually and determines whether the offer fulfills them or not. For each behavior description with or without behavior class annotations, the discovery engine creates the corresponding LTS. We assume that the engine keeps an LTS representation of each behavior description in the repository $\mathbb{D}$.

It is further assumed that a classification hierarchy $\mathcal{T} = (C, H)$ with formal class definitions and the precomputed hierarchic ordering over classes exists. During the initialization phase of the discovery engine, existing behavior descriptions are classified into existing classes automatically. In order to achieve this, the model checking technique presented in Chapter 4 determines whether behavior descriptions fulfill the constraints of a class or not and assigns them to behavior classes accordingly. Descriptions with class annotations are also directly assigned to the respective classes.

The discovery engine materializes the classification information for its later use as an indexing structure $I \subseteq C \times \mathbb{D}$. For each behavior description $\mathcal{D}_\omega \in \mathbb{D}$, we add $(c, \mathcal{D}_\omega)$ to the index I if the behavior of ω was determined to be a member of the behavior class $c \in C$. The task of building a hierarchy and classifying the behavior descriptions into the behavior classes is done offline and is not considered as a part of the query answering task. The classification index I allows to retrieve behavior descriptions of desired classes immediately. By this, it saves expensive model checking and DL reasoning at query answering time.

Let a request $\mathcal{R}$ specify constraints by means of desired behavior classes $C_\mathcal{R} \subseteq C$ of the hierarchy. In addition, $\mathcal{R}$ contains further constraints, which are explicitly modeled and not captured by the classes $C_\mathcal{R}$ (as in our example above). In a first step, the engine retrieves the behavior descriptions $I(C_\mathcal{R})$ from the index structure I. This first step returns all behavior descriptions that are member of all desired classes in $C_\mathcal{R}$.

Composed expressions over desired classes (using $\wedge$, $\vee$, $\neg$) are evaluated by following De Morgan's laws and applying the set theoretic semantics as presented in Equation 4.1 when we introduced the service request language. If a request contains no behavior classes, the first step returns all descriptions from $\mathbb{D}$. Algorithm 2 shows the procedure in which the services W of the desired classes are retrieved from the index. Members of a single class are retrieved

Algorithm 2: retrieveClassMember: Retrieves services of desired classes.

```
Require: Desired service classes expression $C_R$, Index $I$
1: if $C_R$ is a ServiceClass then
2:    $W \leftarrow getMember(C_R)$
3: end if
4: if $C_R$ is a Conjunction then
5:    $W_1 \leftarrow retrieveClassMember(C_R.leftTerm)$
6:    $W_2 \leftarrow retrieveClassMember(C_R.rightTerm)$
7:    $W \leftarrow W_1 \cap W_2$
8: end if
9: if $C_R$ is a Disjunction then
10:    $W_1 \leftarrow retrieveClassMember(C_R.leftTerm)$
11:    $W_2 \leftarrow retrieveClassMember(C_R.rightTerm)$
12:    $W \leftarrow W_1 \cup W_2$
13: end if
14: if $C_R$ is a Negation then
15:    $W_1 \leftarrow retrieveClassMember(C_R.term)$
16:    $W \leftarrow \mathbb{D} - W_1$
17: end if
18: if $C_R$ is empty then
19:    $W \leftarrow \mathbb{D}$
20: end if
21: return $W$
```

with the method *getMember* (line 2). Complex expressions are recursively decomposed. The evaluation of the individual terms of C_R return sets of services that are then combined into the final result by applying the respective set operations.

The result of the method *retrieveClassMember* then serves as input for the second step, where further explicit requirements from $\mathcal{R}$ (that were not captured by the classes) are verified. As shown in the method *discovery* in Algorithm 3, the model checking method that we described in Section 4.3 evaluates the remaining constraints of $\mathcal{R}$ without the classes (i.e., $removeClasses(\mathcal{R})$ in Algorithm 3) only on the results W_0 of the previously invoked method *retrieveClassMember*.

The model checking is only applied to the services of the desired classes. As a result of the model checking, a subset of the given set is returned. Consequently, all the services that are returned by the matchmaker are member of the desired classes, fulfill the explicit constraints, and hence are matches for the request.

By the introduction of the index structure I and the use of behavior classes, the number of behavior descriptions considered in the second step for expensive model checking operations is reduced. Further, the number of constraints evaluated at query time is reduced. Therefore, we assume that retrieving instances that are member of a class or of several classes simultaneously is faster than verifying all constraints at query time. Obviously, many factors like the complexity of class definitions, the size of the classification, and the number of behavior

Algorithm 3: discovery

```
Require: Request $\mathcal{R}$
1: $C_R \leftarrow extractClasses(\mathcal{R})$
2: $W_0 \leftarrow retrieveClassMember(C_R)$
3: $W \leftarrow modelChecking(W_0, removeClasses(\mathcal{R}))$
4: return $W$
```

descriptions have to be considered to let this assumption hold.

6.5 Implementation and Evaluation

In this section, we describe the implementation of the presented approach and report on the evaluation results. The implementation is an integral part of the suprime framework[1] for intelligent usage and management of services and processes and was also applied in the WisNetGrid project as the component to locate Grid-based services. We describe the setup of the experiments conducted in order to evaluate our claims regarding the performance gain and discuss the impact of the granularity of service class definitions.

6.5.1 Implementation

We used the aforementioned Java APIs for modeling and annotating executable behaviors and for specifying search queries for services with matching behavior. For the DL parts of the service descriptions, we use the OWL API for semantically describing the resources, and HermiT OWL reasoner for reasoning about them. We extended the APIs such that service descriptions can be annotated with service classes and requests can include desired service classes. For modeling behavior classes graphically, we provide an assisted Web form based input mask that allows the user to easily enter constraints like the existence of an input action. Existing classes are displayed in a tree-shaped structure and can be selected for reuse.

We modeled the hierarchy of service classes in an OWL repository ontology $O_{\mathbb{D}}$ that is part of the service description repository $\mathbb{D}$. For a given hierarchy and a set of service descriptions, the ontology $O_{\mathbb{D}}$ represents (i) each service class by an OWL class with the name of the service class, (ii) the hierarchical relationships between service classes by OWL subclass relationships between the respective ontology classes, and (iii) each described service as an individual, which is member of the respective service classes.

The definitions of service classes are maintained externally, as DL reasoners cannot interpret these expressions anyway. Given a service class name, the discovery engine can load the service class definition and then, for instance, evaluate the definition on given service descriptions in order to automatically classify services using the model checking based matchmaking method that we introduced in Chapter 4.

Discovery. With the initialization of the discovery engine, the repository ontology $O_{\mathbb{D}}$ is loaded and existing service descriptions are automatically classified into the service classes, if

[1]suprime framework – https://km.aifb.kit.edu/projects/suprime, retrieved 2013-06-11

Table 6.1: Different levels of behavior class granularity

Class Granularity	Fine	Medium	Coarse
Number of behavior classes	60	40	20
Simple constraints per class	3	2	1

it has not been computed before. New classification information is added to $O_{\mathbb{D}}$ by introducing ontology individuals, which represent the services, and defining them as members of the matching service classes.

The given discovery requests are passed to the discovery engine. In a first step, the search space is reduced by retrieving services from requested classes. This is done by invoking a HermiT Reasoner instance with the repository ontology $O_{\mathbb{D}}$.

Then, further explicit constraints are verified on the service descriptions, which were retrieved in the previous step. The results of the second step, i.e., services that fulfill all the constraints of the request, are displayed to the user in the implementation of our graphical search interface.

6.5.2 Evaluation

We conducted several experiments that show the benefits of the presented approach with respect to the search performance. In our experiments we examine the impact of the classification hierarchy on the search performance. As we apply logic-based model checking and classification techniques, it is not necessary to evaluate the quality (soundness and completeness) of search results. Instead, we compare the query answering times for equivalent queries with and without the use of behavior classes. Further, we differentiate between various class complexities and class hierarchy sizes. We developed a class hierarchy with formal class definitions and measured the search performance for varying behavior class granularity. This means, given a fixed number of services or processes, applying a hierarchy with coarse-grained classes corresponds in average to a small hierarchy (number of classes) and few formal constraints per class. In contrast, fine-grained classes lead to a larger hierarchy and more constraints per class. The different levels of class granularity are summarized in Table 6.1.

Test data. The test data is derived from given descriptions of end user browsing processes that coordinate existing Web-based services. We use the same approach to gain semantic service descriptions with executable behavior descriptions derived from the CoScripts of the IBM CoScripter repository. In our experiments we used 2000 synthesized descriptions. On average, our behavior descriptions describe each input/output parameter with three DL axioms, i.e., at least its data type plus its relationship to other process resources. In Table 4.2, the number of classes is related to the number of simple constraints. A *simple constraint* can be one of **eventually** ϕ or **always** ϕ, where ϕ is a simple constraint like a proposition P or the existence of an action a ($\langle a \rangle P$).

Just as in the previous experiments in Section 4.4, we created several search queries composed ($\wedge_{\mu}$, $\vee_{\mu}$, $\neg_{\mu}$) out of 9 simple constraints. Analogously to the descriptions, the complexity of each proposition and parameters of desired actions is set to an average of 3 ± 2 DL axioms (class and object property assertions, e.g., $P \equiv \mathsf{Flight(f)} \wedge \mathsf{Time(t)} \wedge \mathsf{departureTime(f,t)}$). A desired action is expressed by its type (class assertion) and the description of types and relationships between messages.

Results. In Tables 6.2 and 6.3 we show the measured query answering time for varying class granularity and proportion of behavior classes used in the queries for 1,000 and 2,000 behavior descriptions respectively. The baseline is Q_3, which represents queries without behavior classes. The opposite extreme is Q_0 which consists of a combination of classes only. The relative search performance gain is expectedly high. The query Q_2 was derived from the query Q_3 by replacing one of three explicit constraints with respective references to service classes. Q_1 was similarly derived by replacing two out of three constraint of the query with classes.

Table 6.2: Query answering time and relative gain for 1,000 behavior descriptions

Class Granularity	Fine		Medium		Coarse	
	time [s]	gain	time [s]	gain	time [s]	gain
Q_0 (classes only)	0.010	99%	0.009	99%	0.009	99%
Q_1 (66% classes)	3.32	61%	3.21	64%	3.35	63%
Q_2 (33% classes)	5.46	37%	6.10	32%	6.28	30%
Q_3 (no classes)	8.63	0	9.02	0	9.05	0

Table 6.3: Query answering time and relative gain for 2,000 behavior descriptions

Class Granularity	Fine		Medium		Coarse	
	time [s]	gain	time [s]	gain	time [s]	gain
Q_0 (classes only)	0.012	99%	0.018	99%	0.019	99%
Q_1 (66% classes)	10.47	67%	11.83	65%	11.21	68%
Q_2 (33% classes)	19.22	39%	22.06	34%	24.01	31%
Q_3 (no classes)	31.32	0	33.40	0	34.98	0

An interesting outcome of this experiment is that the class granularity is not crucial to the absolute query answering times and the relative gains. Although a slight but steady performance decrease is measured for coarse grained classes, it shows that the overhead of fine grained classes (i.e., having many classes in a hierarchy) does not add significant penalty with respect to search performance. This observation is attributed to the efficient instance retrieval provided by ontology reasoners in case of a relatively large ABox as compared to the TBox. Consequently, the experiments show that growing hierarchies, which can be expected for a Web of services, does not significantly affect the search performance. This argument is underpinned by applicable optimizations that exist for this particular reasoning task [HM08]. The offline computation of behavior class memberships further reduces the instance retrieval time. Furthermore, the query answering time increases significantly for larger numbers of available behavior descriptions. The query answering takes up to 3.9 times longer for 2000 descriptions than for 1000 descriptions if no classes are used (Q_3). If only classes are used (Q_0), the factor can be reduced to 1.2. Thus, the importance of the performance gain by classes becomes even more important when the number of behavior descriptions increases.

6.6 Related Work

The concept of classifying services in order to provide simple methods to locate appropriate services has been broadly proposed and used. In this section, we give an overview on such approaches stemming from the domains of business processes, Semantic Web Services, and services in general. Note that since we aim at developing discovery methods, we do not include a delimitation of our work with classification-based approaches that use heuristics or machine learning in this context. Such methods can provide effective means to classify services without depending on complex formalisms to model service properties. However, as they aim at different use cases, they would not fulfill the requirement for accurate discovery results.

(Semantic) Business Processes

The process query language (PQL) introduced in [KB04] is based on an interpretation of process models as entity-relationship diagrams. A query is a regular expression that allows the usage of '*' for the occurrence of sub-task relationships. Apart from the lack of temporal operators needed for reasoning about the process behavior, the pattern matching based query answering algorithm cannot find answers (process models) that use syntactically different terminology than the one used in the query.

BPMN-Q [ADW08] is a graphical query language using concepts and notations from BPMN. A query graph pattern is matched against a process graph and the control flow and activity names are considered. This approach can be compared to our model checking technique except for the ability of our model checking technique to reason about the data flow and resources as well.

The Business Process Execution Language for Web Services (BPEL4WS) [OAS07] can be compared to our language for describing executable observable behavior. In this chapter, our focus was on the usage and impact of behavior classes on process descriptions and search. Therefore, our constraint specification formalism (with or without classes) can be used for compliance checking and search for BPEL4WS process models if the latter could be interpreted as an LTS.

A process algebra for modeling process behavior using ontologies for semantically describing the process resources was proposed in [ARA08]. However, models described with process algebras need to be treated with closed world semantics, which is often an unrealistic restriction in case of Web-based processes. Therefore, we advanced this approach by the capability to add declarative constraints to process models.

Semantic Web Services

The OWL-S Service Profile aims at the description of atomic services and also implements the concept of a service classification [SPAS03]. However, the relation between service classification and the rest of the service description is not exploited in existing OWL-S based matchmakers (e.g., [SPS06]). The OWL-S Process Model supports specification of complex behavior. In contrast to the Service Profile, the Process Model does not directly support the use of classification. Even though a service classification can be introduced by adding subclasses to the class Service, it is not possible to reason about the dynamics of complex service behavior due to missing formal execution semantics of the OWL-S Process Model.

WSMO [RKL+05] supports the semantic description of both atomic and complex services. WSMO-Lite [VKVF08] is a vocabulary for annotating WSDL [KVBF07] service descriptions with semantically described service properties, e.g., a subclass of FunctionalClassificationRoot. However, as in OWL-S Profile, functional classification is unrelated from other functional properties such as preconditions and effects. Thus, service classifications contradicting the remaining service description are possible. An approach presented in [SHF11] uses functional classifications to achieve efficient discovery of atomic services. However, the formal class definitions are not allowed to be used in combination with formal query parts in requests.

Further Service Modeling Approaches

UDDI [UDD01] was the first attempt to provide users with a system for finding Web services. Each entity of the information model can be annotated with classification meta data. Thus, it allows for coarse-grained service discovery based on the classification meta data. UDDI's Yellow Pages administer a service classification that allows concluding the provided functionality of a service. Industry standards like UNSPSC provide a taxonomy for service, product, and business classification and can be used to classify a service by means of the Yellow Pages. While UDDI can deal with multiple taxonomic classification systems simultaneously, it leaves out a specification of the meaning of the classes. It suffers from underspecified classes of standardized or proprietary classifications and also from an XML-based data model which lacks explicit semantics. Thus, it is for instance not clear how to interpret multiple annotations and the system cannot prevent contradicting classifications.

These drawbacks similarly apply to other discovery approach that do not provide a semantics of classes and consequently do not allow interpreting the restrictions on the service properties implied by a classification. For instance, classification taxonomies like UNSPSC were combined with the OWL-S service matchmakers [SPS06]. Then, a matchmaker simply evaluates class assignments to determine matches without any class semantics.

In [CC06], the authors also indicate the potential of service classifications that offer an intuitive and coarse-grained service retrieval mechanism by investigating how services can be semi-automatically classified based on the similarities to previously classified services. A Bayes-based method to classify services automatically [LZLG07] also promises high accuracy but lacks support for logical consistency checking since the classes were only described by class identifiers and not formally defined.

Based on the assumption that matchmaking of a desired service classification with offered service classifications is less expensive than the DL-based matchmaking of the functionality description, a service discovery approach using formally defined service classes (specified in WSML-DL+) was presented in [LCC06]. In our approach, we can use classes in service descriptions when concrete property values are not known or cannot be revealed. Furthermore, we allow usage of classes in a request together with further constraints, whereas the approach presented in [LCC06] supports only predefined classes in a request.

Our modeling approach is similar to the state transition system based process modeling approach for BPEL4WS presented in [PTBM05], as well as to the finite state machines (FSM) based approach for modeling Web services presented in [HTN08]. [MMWvdA11] presents a linear temporal logic based approach for verifying business constraints at runtime. In contrast to our approach, in which we describe and reason about the data content of a state semantically with description logics, none of these approaches allows reasoning about the data objects involved in a process. Still, our behavior classes can be used to classify and find BPEL4WS processes and Web services as well, given that a semantics as a mapping to an LTS is provided.

A pragmatic temporal logic for reasoning about time intervals of events by proposing a set of relationships between time intervals was introduced in [All83]. Even though there are no results about the expressivity of the interval relationships to the best of our knowledge as well as the fact that μ-calculus is point-based, we believe that the interval relationships can be expressed by μ-calculus. While the interval relationships due to their pragmatic meaning could serve as a good basis for obtaining the end user query language, there are differences in the semantics that need to be taken care of while defining the translation of an interval relationship to a μ-calculus formula. An example for such a difference is “negation”. In interval logic, negation of “before” is “after”, while in μ-calculus negation of “before” is “not before”, which has a different meaning than “after”.

Our query specification formalism falls in the category of DL with modal operators [AF00]. Since we use constant domain terminologies (expressed as DL TBox), we are able to use a more expressive temporal logic than the ones discussed in [AF00] while still ensuring decidability.

6.7 Summary and Conclusion

The classification-based service discovery approach presented in this chapter was primarily motivated by our observation that despite the fact that most of the interesting services are not atomic but rather have complex behavior, there are hardly any convenient specification and search techniques available for such services. Handling a large amount of complex services quickly leads to high query answering times as the computational complexity of matchmaking hampers the development of efficient discovery methods.

In order to address the efficiency problem, we introduced an offline classification of services that has been shown to be effective in reducing the discovery complexity at query time. In contrast to most existing service classifications, we added formal definitions to service classes, which allow us to guarantee the consistent classification of services based on their descriptions.

Orthogonally, we have seen that service descriptions can be annotated with implicit property descriptions when class annotations are added manually to the service descriptions. We also extended the discovery request formalism such that existing service classes can be reused in requests. Classes can seamlessly be combined with explicit property specifications. Based on a reuse of service classes in service requests, we exploit the class hierarchy for precomputing and caching class memberships of descriptions in order to save online query time. We have implemented the presented specification and search approach and showed with our evaluation results the positive impact of behavior classes on the search performance.

While our service classification promises to increase the discovery efficiency and introduces simplifications to the process of modeling services and requests by reused service classes with human readable names instead of logical formulae, it was based on the following assumptions. A set of service classes should exist and they should be reused in requests when applicable. Therefore, users have to find and assess service classes effectively. However, we cannot verify the validity of these assumptions, because it would comprise the study of information retrieval, usability evaluations, and user studies, which is beyond the scope of the thesis. Nevertheless, in the next chapter we introduce offline and online indexes that are automatically constructed. While they do not provide human readable names like service classes, they do not depend on existing classes and potentially human effort to maintain the hierarchy of classes.

Chapter 7

Index Structures for Efficient Service Discovery

In this chapter, we present a technique that precomputes (intermediate) matchmaking results in order to accelerate the model checking phase. Model checking becomes expensive if a large number of services is considered for the matchmaking or if their behavior is rather extensive, because the model checking complexity depends on the number of states of a model (among others). Therefore, we will adapt the use case of our discovery method to a setting in which a large number of available services with rather complex behavior descriptions can be expected.

We introduce offline indexing techniques to the model checking technique from above, integrate the classification-based discovery for complex services into an online index and evaluate the achieved performance. The offline indexes allow us to apply the presented model checker to scenarios with larger numbers of available service descriptions. The online caching algorithm can further serve as a means to automatically create service classes (service class definitions) for a classification hierarchy.

Before we propose our indexing techniques, we introduce in Section 7.1 the use case and our example used throughout this chapter. With a process-oriented view on the Web as outlined in Section 2.1.3, we search for Web-mediated services that offer their functionality via Web pages typically intended for end user interaction (browsing processes). This use case obviously demands for an efficient search as a very large number of functionalities are offered by Web pages and constitute Web-mediated services with complex behavior and interaction patterns. Although other use cases as well as the one used so far benefit from the indexing techniques developed in this chapter, the new use case highlights how our service discovery method can be applied to other use cases, too.

We aim at providing the end users with a list of browsing processes that are relevant for a given information need instead of a list of links to Web pages. Each browsing process in the list of hits will lead the end user to the required information. Such a list of appropriate browsing processes is computed by searching existing end user browsing processes.

In order to be able to search appropriate browsing processes automatically, we show in Section 7.1.1 how user browsing processes (consisting of link selection, form inputs, and information extraction steps) can be formalized and automatically verified with a model checking approach. Section 7.1.2 shows how formal descriptions of browsing processes are captured. In our approach, we can leverage Web page annotations, but we do not require pages to be previously annotated. Explicit end user browsing processes provide the end users with a place for adding the semantic annotations to the Web pages they contain in a bottom up fashion. Until now, semantic annotation of Web pages by end users has been hard since the end users cannot change the Web pages. Then, we show how end user browsing processes can be efficiently searched from a collection of browsing processes so that users can save browsing time by reusing them for their complex information gathering needs. Our search technique is based on monolithic and explicit state representation model checking.

We develop various offline indexes in Section 7.2 and a randomized online index in Section 7.3 to achieve significant gains in the search performance. We present results of performance tests and complexity proofs to demonstrate their impact on search performance by presenting evaluation results. In Section 7.4 we provide implementation details and details on our experimental setup. After discussing related work in Section 7.5, we conclude in Section 7.6 by summarizing our work and giving some future directions.

The vision of our motivating use case is summarized in [Jun13]. The modeling, acquisition, and discovery of end user browsing processes have been presented in [JA13]. We refer to the end user browsing processes as *Web browsing recipes* in order to highlight that they represent instructions, which describe how to access information and functionalities. In [AJ13], we elaborated our work on this scenario by proposing how Web pages can be derived from the recipes and how recipes with similar functionalities can be identified.

7.1 Motivating Use Case

We consider Web-mediated end user browsing processes as services that provide their functionality to end users by a human-centric Web interface. These user interfaces allow for complex interactions, which reflect the complex behavior of the Web applications.

For many practical purposes end users need information that is scattered across multiple websites. Certainly, the information search in the Web can become cumbersome if the desired information is scattered across websites. For instance, even though there are pages listing track chairs of the past WWW conferences and Web accessible bibliography databases, compiling the list of recent books or journal publications of the WWW track chairs with the help of existing search engines is still a time consuming task. It is even harder to find information from the Deep Web as it requires user interactions that are hard to simulate by automatic crawlers.

The websites that provide information can be static or dynamic. Static websites can be crawled by current search engines, and their content can be indexed to provide end users with efficient search over documents. However, in many cases, end users still have to do a lot of work manually to compile the required information. For instance, consider an end user who is interested in the track chair names of previous WWW conferences. As of today, Web search engines like Google and Bing do not even deliver satisfactory results for queries like “track chairs of all WWW conferences”.

One reason for this lack of support is that the sets of links returned by document-centric Web search engines often contain similar information whereas the complex information need requires fractions of complementary information that, if combined, satisfy the information need. In order to obtain the required information, the end user has to pose multiple queries to a search engine, browse through the results, and extract and aggregate the required information fragments outside of the found Web pages.

The case of dynamic websites is even more complex. Accessing the information lying in the Deep Web [Ber01] is already an open challenge for search engines. It is not trivial for automatic crawlers to sensibly interact with the dynamic websites in order to access the underlying information. Furthermore, indexing such information is not a suitable technique since the information underlying dynamic websites can change so rapidly that the index becomes quickly outdated.

Current search engines focus on finding the most relevant Web pages for a given information need rather than providing the information. The ranking of the pages is usually based on the link structure. As a result, a user receives a list of Web pages with similar content even though the information need of the user might require pages with complementary information.

End users need help in selecting the pages that are relevant for obtaining the information scattered over multiple Web pages. Such help must contain at least the set of the pages that the end user should visit, and support for easily invoking the pages in the set. More advanced help could comprise the complete end user browsing process including support for data flow between the user and the pages as well as among the pages, and control flow if there are data dependencies among inputs and outputs of Web pages in the set. Formally, the simpler case means that for a given information need formalized as a query Q we want to compute a list of hits $H_1, \ldots, H_n$, where each hit H_i consists of a set of pages $P_1^i, \ldots, P_{m_i}^i$, and a hit H_i has higher or equal relevance[1] than another hit H_j for each $i < j$. The advanced case means that for each page P_j^i there is a path $\mathit{path}(P_j^i)$ that needs to be executed in order to reach the page P_j^i. Such a path is a sequence of user actions, which can be of three types, namely input actions, output actions, or local actions. Furthermore, a data flow among the paths of the pages of a hit is defined by connecting an input action of a path with an output action of another path. In order to compute such hits, we need to know which information need a page P_j^i satisfies and how this page P_j^i can be reached.

Semantic search based on structured data aims at efficiently answering information needs but relies on the cooperation of providers to be able to access their data. We provide an alternative solution to the information search problem. Our approach builds on goal-oriented end user browsing processes containing instructions for accessing, extracting, and merging (dynamic) information from various websites. These processes are sharable and reusable Web-based services so that users can benefit from the efforts of other users, e.g., to access and integrate information from multiple websites. Search techniques for efficiently finding browsing processes that describe the paths that lead to the requested information are the main prerequisite for effective sharing and reusing of browsing processes.

In the remainder of this chapter, we develop an efficient search technique for finding browsing recipes from large repositories. We augment explicit state representation based model checking techniques by indexing structures tailored to the requirements of information search based on the recipes. The performance evaluation of our approach reveals the impact of the indexing structures on the overall recipe search efficiency.

[1]The relevance of a hit can be determined by an additional ranking method.

7.1.1 Formalization of End User Browsing Processes

In this section, we present how end user browsing processes can be described formally with *suprimePDL* combining the π-calculus process algebra with the description logic $\mathcal{ALC}$. We restrict ourselves to a less expressive description logic like $\mathcal{ALC}$, as a description of navigational paths leading to Web pages with semantic annotations of the displayed information is sufficient for the purpose of information search. More expressive description logics can be employed later if necessary. The formalization of browsing processes is based on the approach presented in [HA10]. We repeat the basic concepts of this approach in order to show how formal descriptions of browsing processes are related to individual Web pages and the Web form based interactions that they offer.

Constraints on such end user browsing processes can be specified in our request language based on the combination of the temporal logic μ-calculus with description logic. Analogue to the process descriptions, we will likewise use the $\mathcal{ALC}$ description logic in requests. In contrast to our imperative descriptions of browsing processes, declarative behavior constraints in search requests express desired properties of the behavior of a browsing process. End user browsing processes can be automatically checked against constraints with a model checking approach.

To illustrate the formalization of end user browsing processes, we use the example of collecting the track chairs of previous WWW conferences and then query their articles from a bibliography database. In the example browsing process, static conference Web pages are visited and the names of track chairs are marked as relevant outputs. Next, a Web page like DBLP is visited, author names are entered sequentially or in parallel into the input form of the page and the corresponding articles are extracted from the result pages.

A single Web page is a message sent by its hosting server to an end user. In addition to the information content, a Web page may contain the description of a choice process. The choice process consists of a set of links and a set of forms. Formally, the output action $y\langle\mathbf{v}\rangle$ of the server that produces a Web page with values $v_1, \ldots, v_l$ (denoted by $\mathbf{v}$), links $l_1, \ldots, l_m$ and forms $f_1, \ldots, f_n$ is described by:

$$\begin{aligned} y\langle v_1, \ldots, v_l\rangle.@L_1\{\mathbf{x_1}\} + \ldots + @L_m\{\mathbf{x_m}\} \\ + @F_1\{\mathbf{y_1}\} + \ldots + @F_n\{\mathbf{y_n}\}, \end{aligned}$$

where $L_1, \ldots, L_m$ denote the base URLs of the links $l_1, \ldots, l_m$, $\mathbf{x_1}, \ldots, \mathbf{x_m}$ denote their parameters if any, $F_1, \ldots, F_n$ denote the action URLs of the forms $f_1, \ldots, f_n$, and $\mathbf{y_1}, \ldots, \mathbf{y_n}$ denote their submission parameters. In our view, a URL is equivalent to an agent identifier, whereas the selection of a link, which is a usage of a URL, is equivalent to an agent invocation (denoted by an @ preceding an identifier) with concrete values for the arguments. The mapping between the major elements of Web pages to the elements of our behavior formalism is summarized in Table 7.1. Furthermore, we model the arguments of a link as concepts in the ontology associated with the *suprimePDL* process expression describing its base URL, and the values of the arguments as individuals of the ontology class corresponding to the arguments. We model the identifiers of display elements that contain output values as ontology classes. A form corresponds to a complex ontology class. The names of the input elements of the form correspond to the properties of the complex class. The name of the ontology class corresponding to the range of a property can often be derived from the label of the input field (see e.g. [MKK$^+$08]). Some types of input elements provide a set of values from which one

Table 7.1: Mapping between Web artifacts and elements of *suprimePDL*

Web Artifact	**Element of the Process Description Language**
URL	Agent identifier
Web page	Output action and a choice from a set of links and forms
Selection of a link	Invocation of an agent identifier
Submission of a form	Input action
CGI script	Execution of a local action

Table 7.2: Correspondence between page content and ontology

Element	**Maps to**
Base URL of Web page / link	Logical URI of ontology
Display element id	Ontology class
Content of a display element	Ontology instance of the class corresponding to the display element id
Variable name of link	Ontology class
Variable value of link	Ontology instance of the class corresponding to the variable
Form name	Complex ontology class
Form field id	Property of the class corresponding to the form name
Form field name	Ontology class representing the range of the property corresponding to the field id
Form field value	Ontology instance

or more values can be selected. In these cases, the provided values are modeled as ontology individuals, while the concept representing the range of an input field is modeled as an RDF/S container class instead of a normal class. Table 7.2 lists the ontology concepts used to represent the Web page elements.

An end user browsing process is equivalent to an agent identifier that is defined as a parallel composition of the invocations of the agent identifiers $P_1, \ldots, P_n$ corresponding to the websites visited in the browsing process and a coordinating process C. Such a browsing process is defined as

$$@C\{\} \parallel @P_1\{\} \parallel \ldots \parallel @P_n\{\}.$$

Example 15. *The 2013 WWW conference website at URL u_1 links among many other options to the "call for research papers" Web page denoted by the process CFP_{www13}. The process provides a Web page listing all research tracks* **tracks** *denoted by the output action $u_1\langle\mathbf{tracks}\rangle$.*

$$CFP_{www13}() \overset{\mathsf{def}}{=} u_1\langle\mathbf{tracks}\rangle.\sum_{t\in\mathbf{tracks}} @CFP_t\{\}$$

Selecting one of the provided links, say $@CFP_{bridging}\{\}$, returns the page about one specific track and lists topics, track chairs, and PC members.

$$CFP_{bridging}() \overset{\mathsf{def}}{=} u_1\langle\mathbf{topics}, \mathbf{chairs}, \mathbf{pcs}\rangle.\mathbf{0}$$

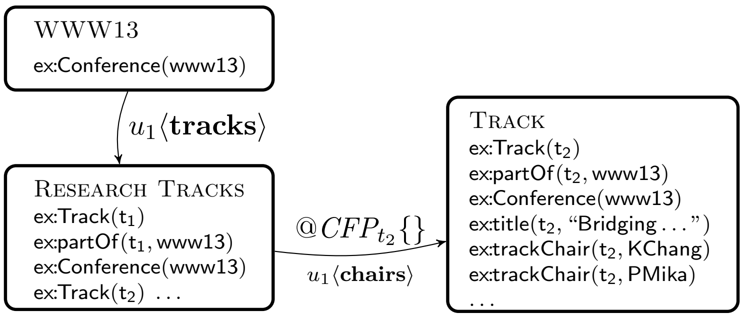


Figure 7.1: Excerpt of the LTS of an example script collecting track information

Figure 7.1 shows the corresponding LTS representation of the end user browsing process for accessing the track information of the 2013 WWW conference. The second website to be considered in this example is DBLP at URL u_2 *in order to search for publications. Among other information* $\mathbf{x}$ *on the entry Web page retrieved by the invocation of DBLP(), the Web form Search takes the author name obtained from the input action as single input value.*

$$DBLP() \overset{\text{def}}{=} u_2\langle\mathbf{x}\rangle.u_2[author].@Search\{author\}$$

The form submission generates a dynamic page listing the publications of the specified author. Although this page contains many further links, we restrict our example and terminate the process at this stage with the Null process.

$$Search(author) \overset{\text{def}}{=} u_2\langle\mathbf{articles}, \mathbf{authors}, \ldots\rangle.\mathbf{0}$$

The whole browsing process Recipe then combines both information sources by invoking both components in parallel and coordinating the flow of the track chair names from $CFP_{research}$ *to DBLP. This is achieved by means of a controlling process C that runs in parallel to both component websites, receives the outputs of the WWW conference website, provides the author names to DBLP, and receives a list of publications* **pubs** *of each track chair* $c \in$ **chairs**.

$$Recipe() \overset{\text{def}}{=} C \parallel @CFP_{www13}\{\} \parallel \prod_{c\in\mathbf{chairs}} @DBLP\{\}$$

$$C \overset{\text{def}}{=} u_1[\mathbf{chairs}]. \prod_{c\in\mathbf{chairs}} u_2\langle c\rangle.u_2\langle\mathbf{pubs}\rangle.\mathbf{0}$$

7.1.2 Capturing Browsing Processes

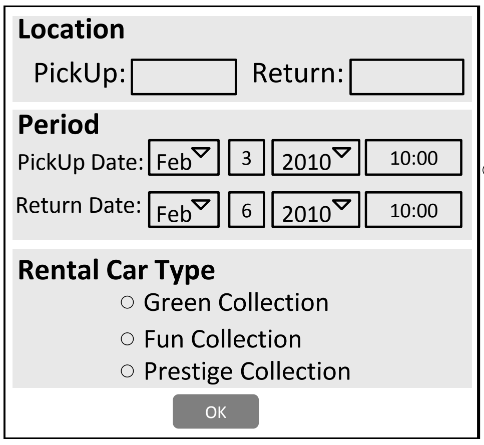
A major advantage of our approach is that it ensures that the capturing of end user browsing processes does not require unnecessary extra manual effort from the end users. There are quite a few browser plug-ins, e.g., the open source plug-in CoScripter and the commercial plug-in iMacros that can record a user's browsing actions and save them as an executable script. Furthermore, rather than the top-down approach of ontologies in the Semantic Web, we advocate a bottom-up approach beginning with websites and the end users who browse them using standard browsers. Website owners are not involved, and annotations consist of exactly those necessary for a particular application. Once a user has tediously found the right path through a sequence of websites for a particular goal, she should be able to find it much faster and easier the next time. Saving and sharing the process consisting of these steps in a reusable way is thus valuable. Therefore, such an approach not only easily captures the end user browsing processes but also incentivizes end users to extract the relevant information from the Web pages and share their browsing processes with others.

Incentives for end user information extraction. We do not have any information about what information a user has extracted from which website. One reason for the lack of such information is that currently there are hardly any tools that create added value for the users from the extracted information. Benefits of having such information are for example the following:

- End users will be able to find the previously extracted and aggregated information much more easily than searching it all over again in the Web. Currently, end users can either copy the information they want to remember in an application of their choice or create a bookmark. In our approach, the extracted information can be stored as RDF and searched with SPARQL. An example could be searching for publications relevant for a user's own research.
- End users will be able to integrate and use the extracted information more easily in other applications such as calendars and task lists.

Learning semantics of navigation paths. Having a formalism for semantically describing the navigation paths, we now turn our attention to obtaining the semantic descriptions. Of course, we can simply require end users to annotate their scripts with formal ontologies before sharing them with others. But, this would be against our bottom-up principle and is similar the early top-down Semantic Web approach that, in some aspects, has fallen short of expectations. Therefore, our aim is to demand minimum manual effort from users for those tasks that do not bring any direct added value for the end users. We developed an approach for learning the semantics of data involved in user actions. It is important to note that the actions we use for learning are the ones that these users perform anyway for their own information need. Thus, no overhead is imposed on the users. Our approach is based on the findings in [HA10].

Learning semantics of inputs. Concepts and relationships of the domain ontology of a Web page can be directly extracted from the structure of the forms and links on the page with the help of the mapping shown in Table 7.2. For doing so, we view a form as a tree with fields as nodes. While the nodes at the same level may not necessarily have a direct relationship, they are usually related with their respective parent nodes. For each field we create an ontology concept for denoting the set of values the field can take, and an ontology relation to relate it to the concept corresponding to its parent node. This method is of course a heuristic and based upon parsing HTML code which is often noisy. As a result, the ontologies learned in this manner are subject to manual review and corrections. Figure 7.2 shows an example form of a car rental Web form along with the ontology derived by our method. The semantic



Location

PickUp: [ ] Return: [ ]

Period

PickUp Date: Feb ▽ | 3 | 2010 ▽ | 10:00

Return Date: Feb ▽ | 6 | 2010 ▽ | 10:00

Rental Car Type

○ Green Collection

○ Fun Collection

○ Prestige Collection

OK

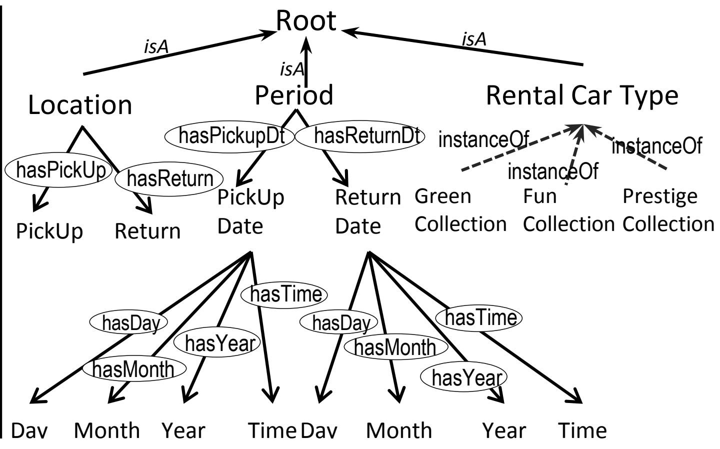


Figure 7.2: Example semantic description of the inputs of a form [HA10]

annotations of an input action describe constraints on the input values and are part of the semantic description of the input action.

When an end user enters some values in a form automatically, i.e., from her local knowledge base (*KB*), mappings between the domain ontology of the local *KB* and that of the Web page are derived. For example, if the local *KB* of a user contains a concept u:C with instances $\mathsf{u{:}c_1}, \ldots, \mathsf{u{:}c_n}$, and the user selects $\mathsf{u{:}c_1}, \ldots, \mathsf{u{:}c_n}$ as values for a form field p:D, then we derive $\mathsf{u{:}C} \sqsubseteq \mathsf{p{:}D}$.

Learning semantics of outputs. Learning semantics of outputs is a bit more complex than learning semantics of inputs. In this case, we first need to identify the outputs of navigation paths before we turn our attention to learning their semantics. In order to identify outputs, we search for the data extraction operations in a navigation path. With a data extraction operation, an end user is able to copy some useful information from a Web page to her local *KB*. Usage of data extraction operations gives us the hint that the end user is interested in the extracted information, hence the information is one of the significant outputs (information goal) of the navigation path. When an end user extracts some information from a Web page and copies it into her local *KB*, she needs to provide the information where the new information should be added to the local *KB*. The locations in the local *KB* correspond to ontology classes, and we automatically infer that the extracted information (seen as ontology individuals) are instances of the corresponding selected concepts in the local *KB*. When an end user extracts some values from a Web page, individuals are created in her local *KB*. In addition to the extraction of information and their representation as instances of appropriate ontology concepts, the user may add further structure to the newly added instances in order to be able to search for them better at a later stage. This additional structure can be added to the semantic annotation of the outputs in addition to the already inferred type information of the outputs. For example, if an end user extracts two information values a and b and adds them to her local *KB* as instances of concepts u:A and u:B respectively, we create u:a and u:b as instances of the concepts u:A and u:B respectively in the local *KB*. When the end user adds a relation u:r between u:a and u:b, the semantic annotation u:r(u:a, u:b) can be added. The ontology derived in this manner describes the types and relationships of the outputs (extracted information) of a Web page and is part of the semantic description of the extraction (output) action.

Deriving ontology mappings from script compositions. Apart from deriving semantics of input and output actions of a script, we exploit script compositions for deriving mappings between domain ontologies of different scripts. Script composition is performed by an end user in order to achieve complex tasks, parts of which can be solved by existing scripts, such as aggregation of flight offers. Technically, script composition consists of defining data flow among the scripts, i.e., wiring outputs of one script with the inputs of another script. When an end user defines such a wiring, we can automatically derive that the concepts and relations describing the outputs of a script are sub-concepts and sub-relations of those describing the inputs of the other script.

7.2 Offline Computable Indexes

After we presented the motivating use case, we introduce an efficient approach for the discovery of services based on index structures. While model checking provides a basic technique, it is not sufficient to perform model checking of each browsing process for a given query specified in the constraint specification language because the naive model checking of large sets of browsing process remains very time consuming.

7.2.1 Proposition-States Indexes

One of the main problems of naive model checking is that it requires a lot of time for evaluating the description logic (DL) propositions. This is mainly due to the following three problems: (i) The DL reasoning itself is not efficient, (ii) the naive approach checks the same proposition for the same state multiple times, and (iii) the states are checked in sequence. While optimizing description logic reasoning is out of the scope of this work, we present in the following how we address the latter two problems.

It is easy to see from Algorithm 1 in Section 4.3.3 that it takes a lot of time to iterate over all the states and check whether they satisfy a given proposition (see lines 7 to 11). This takes $O(n)$ time, where n is the number of states in the entire LTS. In order to save time for proposition evaluation, we build the proposition-states (PS) indexes.

An atomic proposition is a triple $(\mathtt{s}, \mathtt{p}, \mathtt{o})$, with subject $\mathtt{s}$, object $\mathtt{o}$, and predicate $\mathtt{p}$. A proposition-states index is a list of (φ, S) pairs, where φ is a proposition and S the set of states that satisfy φ. A PS index contains the propositions that can be directly derived from the axioms that describe the states. Therefore, it can be built offline by extracting the propositions from the states and the actions described by the transitions.

Algorithm 4 describes how the proposition-states indexes can be built. The algorithm iterates over every state of every LTS behavior description. For each proposition $\varphi \in \lambda(s)$ that describes a state s, we add the proposition φ and the label $\mathsf{ns{:}s}$ of the state s to the table I. If the proposition was already added, we add s to the list of states in the right column of the table I. Then, four versions PSO, POS, SPO, and OPS of table I are created by rearranging predicate $\mathtt{p}$, subject $\mathtt{s}$, and object $\mathtt{o}$ of each proposition in I, respectively.

Algorithm 4: Build proposition-states indexes

```
Require: an LTS $L = (S, \rightarrow, A, \lambda)$, empty table $I$
  for all states $s \in S$ do
    Let $ns$ denote the prefix of $s$
    for all propositions $\varphi \in \lambda(s)$ do
      add a new row to $I$ and let $r$ denote this row
      insert $\varphi$ into first column of $r$
      add ns:s to the entries in the second column of $r$
    end for
  end for
  PSO $\leftarrow$ table $I$ sorted by p, s, o of the first column
  POS $\leftarrow$ table $I$ sorted by p, o, s of the first column
  SPO $\leftarrow$ table $I$ sorted by s, p, o of the first column
  OPS $\leftarrow$ table $I$ sorted by o, p, s of the first column
```

Different orderings (denoted by PSO, POS, SPO, and OPS) of the serialization of propositions allow us to process queries with propositions that contain variables, e.g., ex:partOf(?x, www13) where ?x denotes the variable in the subject position. Here, the POS or OPS index may be used to syntactically match predicate and object positions and retrieve matching states efficiently. The subject position is left out as the variable ?x will syntactically not match an actual subject from the index. The orderings SOP and OSP are not indexed, as propositions with unspecified predicates (hence the trailing p) are only relevant for rare cases in search queries.

Whenever during the model checking of a browsing process we need to retrieve the set of states that satisfy a given proposition, we perform a lookup in the PS index instead of iterating over all the states of the entire model containing the LTS representations of each browsing process. This means, we replace in Algorithm 1 the lines 6 to 12 with the statement "**return** *lookupPS*(P)". Since the PS index is sorted by proposition, such a lookup in possible in $O(\log n)$ time where n denotes the number of distinct propositions derived from the states.

Example 16. *Propositions* $\lambda(s_{\text{R.Tracks}})$ *of a state* $s_{\text{R.Tracks}}$ *are derived from the axioms in the state knowledge base* $KB(s_{\text{R.Tracks}})$*. For instance, the second state in Figure 7.1 contains the axioms* ex:Track(t_1)*,* ex:Conference(www13)*, and* ex:partOf(t_1, www13) *with named instances* t_1 *and* www13*. All axioms of* $KB(s_{\text{R.Tracks}})$ *are directly added as propositions to* $\lambda(s_{\text{R.Tracks}})$*.*

The domain knowledge $O_{\mathbb{D}}$ *of the search system defines super classes and super properties by additional axioms* ex:Conference $\sqsubseteq$ ex:Event *and* ex:partOf $\sqsubseteq$ ex:colocated*. Based on* $KB(s_{\text{R.Tracks}})$ *and* $O_{\mathbb{D}}$*, we can derive further propositions* ex:Event(www13)*,* ex:colocated(t_1, www13) *and add them both to* $\lambda(s_{\text{R.Tracks}})$*.*

Table 7.3 shows the populated PS index in PSO ordering for our example. String concatenation ($\circ$) is used to serialize the propositions as a single key in the index. The propositions of each state of a browsing process are added to the index. Then, all the states of the browsing process that satisfy a proposition from the index are stored in the respective rows of the right column.

Table 7.3: Proposition-states index in the PSO ordering

Proposition ϕ (serialized to p ∘ s ∘ o)	States $\{s \mid s \in S \wedge s \models \phi\}$
rdf:type ∘ ex:Conference ∘ www2013	s_{WWW13}, $s_{\text{R.Tracks}}$, s_{Track}
rdf:type ∘ ex:Event ∘ www2013	s_{WWW13}, $s_{\text{R.Tracks}}$, s_{Track}
ex:partOf ∘ www2013 ∘ t	$s_{\text{R.Tracks}}$, s_{Track}
ex:colocated ∘ www2013 ∘ t	$s_{\text{R.Tracks}}$, s_{Track}
⋮	⋮

7.2.2 Action-States Indexes

Model checking as presented in Algorithm 1 consumes a lot of time for iterating over all the transitions and checking whether they satisfy the constraints of a requested action (see lines 19 to 23). This takes $O(n)$ time with n being the number of transition in the entire LTS. In order to save time for the evaluation of actions, we build the action-states indexes. An action-states (AS) index is a list of (A, S^2) pairs, where A is an action, and S^2 the set of states pairs that are connected via the action A. An action-states index contains all actions that are modeled as transitions of the LTS. Therefore, it can be built offline by extracting the actions from the transitions.

Algorithm 5: Build action-states indexes

```
Require: an LTS $L = (S, \rightarrow, A, \lambda)$, empty table $I$
  for all transition $t = (s_1, a, s_2) \in \rightarrow$ do
    Let $ns$ denote the prefix of $s_1$
    add a new row to $I$ and let $r$ denote this row
    insert ns:a into first column of $r$
    add $(\text{ns:s}_1, \text{ns:s}_2)$ to the entries in the 2nd column of $r$
  end for
  TCP ← table $I$ sorted by t, c, p of the first column
  CTP ← table $I$ sorted by c, t, p of the first column
```

Algorithm 5 describes how the action-states indexes are built. The algorithm iterates over every transition of every LTS behavior description. Actions are characterized by their type t (input, output, or local), a communication channel c, and the parameters p. For each transition t, we add the action label ns:a and the corresponding pair of states $(\text{ns:s}_1, \text{ns:s}_2)$ to the table I. Analogue to the proposition-states indexes, no duplicate keys are added. Instead, a list of entries can be stored in the right column of I. Then, the algorithm computes two versions TCP and CTP of table I

A requested action further contains constraints (propositions) on the parameters, which are verified by means of the proposition-states indexes. An action is a match if the state s_2 satisfies the propositions of an input or a local action. Since output actions do not change the state knowledge, the original state s_1 and the subsequent state s_2 can be used likewise to verify the propositions. The action parameters are necessary to ensure that the requested propositions

Table 7.4: Action-states index in the TCP ordering

Action a (serialized to $\mathtt{t} \circ \mathtt{c} \circ \mathtt{p}$)	$\{(s_1, s_2) \mid s_1, s_2 \in S \land \exists a : s_1 \rightarrow^a s_2\}$
supProc:output $\circ$ u_1 $\circ$ **tracks**	$(s_{\text{WWW13}}, s_{\text{R.Tracks}})$
supProc:output $\circ$ u_1 $\circ$ **chairs**	$(s_{\text{R.Tracks}}, s_{\text{Track}})$
$\vdots$	$\vdots$

hold for the action parameters and not for other process resources. Hence, p is in the rightmost position of the concatenated entries in the first column (i.e., TCP, CTP).

Whenever during the model checking process we need to retrieve all the states that have a transition with a given action, we perform a look up in an action-states index instead of iterating over all the transition of the entire LTS. That is, we replace in Algorithm 1 the lines 18 to 24 by the statement "**return** *lookupAT*(a)". Since the action-states indexes are sorted by action types and channels, such a lookup in possible in $O(\log n)$ time, where n denotes the number of transitions in the LTS.

Example 17. *Table 7.4 shows the action-states index. Two output actions were depicted in our example browsing process in Figure 7.1. The first action returns a vector* ***tracks*** *of conference tracks and the second action returns a vector* ***chairs*** *of track chairs. These output parameters are part of the Web pages that are returned by a Web server and displayed in the client browser.*

7.3 A Randomized Online Index

The computation of set intersection (see line 14 of Algorithm 1), and set difference (line 16) of sets of size n takes $O(n)$ time. It is not possible to compute and index all the conjunctions and negations a priori as there can be infinitely many. However, often end users pose queries that have been posed already. So, if the system caches the answers of the queries when they are posed for the first time, it can retrieve the answers for recurring queries much faster. In Section 7.3.1, we show how a cache of previously posed search queries can be utilized. Since there can be an infinite number of queries but the cache cannot be infinitely long, we need to incorporate a caching technique that guarantees certain performance even though the maximum size of the cache is fixed. In Section 7.3.2, we apply the Marking algorithm to maintain the cache size and discuss the performance guarantees of our online index.

7.3.1 Lookup in the Online Search Index

A page in the cache corresponds to a complex formula (behavior constraint) together with the set of states that satisfy the formula. The purpose of this index is to accelerate the computation of conjunction and negation in the model checking algorithm (lines 14 and 16 of Algorithm 1).

Definition 15 (Subformula). *We say that a formula ψ is a subformula of a formula ϕ provided that ψ, when viewed as a sequence of symbols, is a substring of ϕ. A subformula ψ of ϕ is said to be proper provided that ψ is not ϕ itself. A top-level (or immediate) subformula is a maximal proper subformula. We use $SF(\phi)$ to denote the set of subformulae of ϕ.*

Algorithm 6: Online index lookup

```
Require: A formula $\phi$
  repeat
    find one of the largest subformulae $\psi$ of $\phi$
    retrieve the set $I(\psi)$ of states associated with $\psi$
    replace each occurrence $\psi$ in $\phi$ by a variable $X_\psi$
    set the value of $X_\psi$ to $I(\psi)$.
  until $\phi$ contains only variables or it is not possible to find any subformulae of $\phi$
  return $\phi$
```

Algorithm 6 illustrates the lookup procedure. It returns a formula in which all available subformulae of ϕ have been replaced with variables along with the valuations of the variables. Model checking the new formula by considering the valuations of the variables is equivalent to model checking ϕ, as (i) μ-calculus formulae can be decomposed and verified independently and (ii) the order of the composition of the individual results is irrelevant (cf. Algorithm 1).

Example 18. *Suppose the index contains formulae $\phi_1 \wedge_\mu \phi_2$ and $\phi_2 \wedge_\mu \phi_3$ along with the set of states that satisfy them. Suppose the formula that needs to be evaluated is $\phi_1 \wedge_\mu \phi_2 \wedge_\mu \phi_3$. Since it is not in the index, Algorithm 6 searches for its subformulae. Without loss of generality, let us assume that the algorithm finds $\phi_1 \wedge_\mu \phi_2$ first. Let the set of states associated with it be S_1. The algorithm then replaces the occurrence of $\phi_1 \wedge_\mu \phi_2$ in $\phi_1 \wedge_\mu \phi_2 \wedge_\mu \phi_3$ by X_1 and assigns S_1 as value of X_1. So, the original formula becomes $X_1 \wedge_\mu \phi_3$. It is not possible to find any further subformulae of $X_1 \wedge_\mu \phi_3$ in the index. So the algorithm returns $X_1 \wedge_\mu \phi_3$. Then, our online index based model checker evaluates $X_1 \wedge_\mu \phi_3$ by an online index lookup for X_1, applies model checking to ϕ_3, and computes the result as the intersection of both result sets.*

Finding an immediate subformula takes $O(\log n)$ where n denotes the size of the index. Replacing each occurrence of ψ in ϕ is linear in the length of ϕ and is negligible. Because the number of non-variable terms decreases by at least one in each iteration, the loop runs at most $m - 2$ times, where m denotes the number of terms in ϕ. This means that the complexity of the lookup algorithm is $O(m^2 \log n)$.

When during a model checking process a formula of type conjunction or negation is evaluated, at first an index lookup is performed. If the formula returned by the lookup algorithm (Algorithm 6) consists of one variable only, it means that the original formula was found in the index. Otherwise, we add the original formula to the index according to the Marking algorithm that is described in the following section.

7.3.2 Maintaining the Size of the Index

We use a randomized online algorithm called the *Marking* algorithm for building an index for complex search queries. The Marking algorithm is easy to implement and is optimal to a factor of $2H_k$. It was introduced in [FKL+91].

The *paging problem* is the problem of deciding which page from a cache of limited size is ejected into a slower memory if all pages are used. A request to a page is satisfied if it resides in the cache; otherwise a page fault occurs. It is the goal to reduce costs by reducing the number

Algorithm 7: Marking

```
if $p$ is not in the cache then
  if there are no unmarked pages in the cache then
    unmark all pages.
  end if
  if $p$ does not fit in the cache then
    evict a randomly chosen unmarked page
  end if
  bring $p$ into the cache
  mark $p$
end if
```

of page faults for a sequence of requests. An *online* paging algorithm decides which pages to eject without knowledge of future requests. The *competitiveness* of an online algorithm compares its costs to an offline algorithm that knows the entire request sequence in advance (optimum).

The Marking algorithm (see Algorithm 7) associates a bit with each page in the cache. When the corresponding bit of a page is set, we say that the page is marked, otherwise it is unmarked. Whenever a requested page is brought into the cache it is marked, that is, its bit is set. Whenever room is supposed to be made for a requested page that is not in the cache, an unmarked page is chosen uniformly at random and evicted.

Definition 16 (randomized competitiveness)**.** *Given a request sequence σ. Let $C_A(\sigma)$ and $C_B(\sigma)$ denote the cost for the sequence σ incurred by a randomized online algorithm A and an offline algorithm B respectively. A is called c-competitive against B, if there exists a constant α such that $C_A(\sigma) \leq c \cdot C_B(\sigma) + \alpha$. The constant α is independent of the number of requests σ.*

Theorem 1. *Let A be any randomized online caching algorithm for uniform page sizes. Let k denote the maximum number of pages the cache can contain at a time. The competitiveness of A is greater than or equal to H_k (where H_k is the kth harmonic number), if the slow memory contains $N \geq k+1$ pages.*

Proof. We refer to [FKL+91] for proof. □

The kth number in the harmonic series is $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$, which is roughly $ln(k)$. In comparison, deterministic online algorithms cannot be guaranteed to be in a factor smaller than k of the optimum.

Theorem 2. *The Marking algorithm is $2H_k$-competitive. This means, that the total time taken by the Marking algorithm for serving any arbitrarily long sequence of requests is at most $2H_k$ times the total time needed by an optimal offline algorithm for serving the same sequence of requests.*

Proof. We refer to [FKL+91] for proof.

The Marking algorithm divides the whole request sequence σ in rounds of size k. The first round begins with the first request σ_1. A round starting with σ_i ends with σ_j, such that

$\sigma_i, \sigma_{i+1}, \ldots, \sigma_j$ contains requests to k distinct pages and $\sigma_i, \sigma_{i+1}, \ldots, \sigma_{j+1}$ contains requests to $k+1$ distinct pages. An exception is the last round, which may contain requests to less than k distinct pages. At the beginning of a round, all the pages are unmarked.

Consider requests in any round. A page is called *clean* if it was not requested in the previous and the current round. Let l denote the number of requested clean pages during the current round. The proof establishes the claim that the expected number of faults incurred by marking during a round is at most $l \cdot H_k$.

Let S_O denote the set of items in the optimal offline algorithm's cache and S_M denote the set of items in the Marking algorithm's cache. Let d_I be the value of $|S_O \setminus S_M|$ at the beginning of the round and d_F be the value of $|S_O \setminus S_M|$ at the end of the round. The d_I and d_F terms for all rounds (except for the first and the last round) telescope, so that the *amortized* number of faults of this round is at least $\frac{l}{2}$.

Each of the l requests to clean pages adds one fault to the costs of the Marking algorithm. For the remaining $k-l$ requests, the expected costs correspond to the probability that the requested page is not in the cache. In the case when the l requests to clean pages precede the remaining $k-l$ pages, this probability is maximal. More precisely, the probability that the ith request $(1 \leq i \leq k-l)$ incurs a fault is

$$\frac{l}{k-i+1}.$$

Summing over all $i \in \{1, \ldots, k-l\}$, we obtain the expected number of total faults in this round as follows:

$$\sum_{i=1}^{k-l} \frac{l}{k-i+1} = l \cdot \sum_{i=1}^{k-l} \frac{1}{k-i+1} = l \cdot (H_k - H_l).$$

That is, the total number of faults is $l + l \cdot (H_k - H_l) \leq l \cdot H_k$. Consequently, the Marking algorithm is $2H_k$-competitive. For further details, we refer to [FKL+91, Aga01]. $\square$

The model checking complexity of a discovery system that incorporates our online index structure mainly depends from the request sequence. As for any online data structure, the actual request sequence impacts the data structure, i.e., the set of elements in the index in our system. Because it is hard to estimate the requests of the proposed system and, thus, to evaluate the performance in experiments, we discussed the theoretical complexity. The randomized algorithm is $2H_k$-competitive, which means that for any sequence of queries it requires at most $2H_k$ more time than an optimal algorithm for serving the sequence of queries.

7.4 Implementation and Evaluation

This section provides details about the evaluation settings, the origin of the test data, and the used search queries. We report on the performance results obtained in the experiments to reveal the impact of the indexes on the discovery efficiency.

7.4.1 Evaluation Setup

We developed Java APIs for modeling semantic browsing processes and constraints of search requests. Process descriptions and their LTS representations are serialized in the form of RDF/S statements based on appropriate RDF/S vocabularies we developed. LTS representations and domain ontologies used by processes are stored in the OWLIM-SE semantic repository that provides a SPARQL query interface and optimized owl:sameAs handling [BKO+11]. SPARQL[2] is the query language used to query the repository of LTS. OWLIM supports RDF/S reasoning and allows us to directly reason over heterogeneous LTS descriptions based on the domain ontologies.

Search queries are decomposed by our μ-calculus reasoner developed as a Java component that is flexible enough to provide the incremental, monolithic, and index-based model checking capabilities. Our model checking component, the μ-calculus reasoner, is placed on top of the locally installed OWLIM-based LTS repository. During the experiments we allocated 1GB of main memory for OWLIM including its internal index structures. The index structures introduced in the present chapter are implemented as Java data structures (TreeMap) within the μ-calculus reasoner.

In the evaluation experiments shown above, we examined the search performance by means of measuring the query answering time with different model checking approaches. Except for the monolithic approach without indexes, we did not experience any shortage of main memory while we conducted the experiments on commodity hardware with an Intel Core2Duo 2.6GHz CPU and 4GB main memory.

7.4.2 Test Data

As in our previous experiments, existing Web browsing processes from the IBM CoScripter repository serve as the basis of the large test collection used in our experiments. We generated semantic browsing process descriptions with the characteristics summarized in Table 4.2 (page 76). In our experiments we loaded up to 20,000 generated browsing processes into the repository. On average, they describe each input and output parameter by three axioms (i.e., at least the data type of the parameter plus its relationship to other process resources).

On average, the search queries of the experiments are conjunctive and disjunctive compositions of 4 propositions and one existential action query that eventually occurs in desired recipes. Each μ-calculus proposition describes 2 instances and further 3 DL axioms on average. For instance, the following proposition P of a search request contains two class membership axioms and an object property specifying the relation between the two instances (here, ?x denotes a variable).

$$P \equiv \mathsf{ex{:}Chair(?x), ex{:}chairs(?x, www13), ex{:}Conference(www13)}$$

An example of a tested request is Ψ, where $P_1, \ldots, P_4$ denote propositions with complexity similar to the complexity of P.

$$\Psi \overset{\text{def}}{=} (P_1 \vee_\mu P_2) \wedge_\mu (\textbf{eventually } P_3 \vee_\mu \textbf{eventually } P_4)$$

The chosen domain ontologies used to describe the process resources in requests belong to the same set of RDF/S ontologies used for the description of browsing processes. We used 4 public ontologies such as the Semantic Web Research Community ontology [SBH+05]. The largest ontology contains 70 classes and 48 object properties.

[2]SPARQL – http://www.w3.org/TR/sparql11-overview, retrieved 2013-08-15

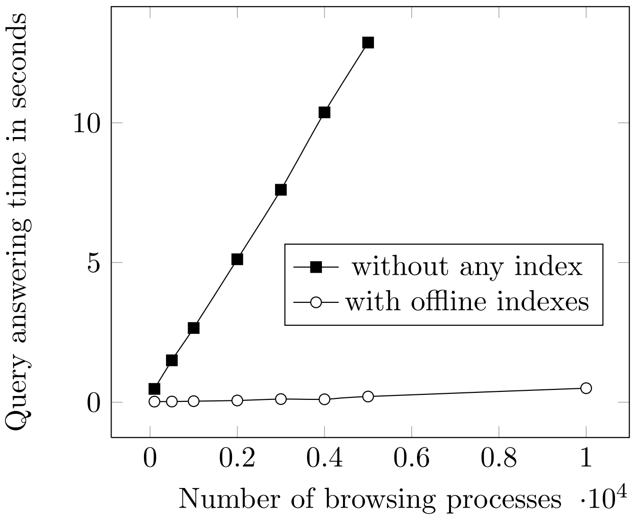


Figure 7.3: Comparison of search performance of the monolithic model checking approach with and without offline indexes

7.4.3 Performance Results

Figure 7.3 shows the performance of the so obtained search approach. The monolithic model checking approach, as described in Section 4.3.3 on page 71, is used as the baseline as it performed considerable better in comparison to the incremental approach. As we discussed earlier, the monolithic approach suffers from a high memory demand. Hence, we can introspect the performance gain of our index structures for a test repository with up to 5,000 browsing processes only.

The query answering time is tremendously reduced. E.g., the query answering time is reduced by the factor 63 for 5,000 browsing process descriptions (from 12.8 seconds to 0.205 seconds). Even for 20,000 processes, the index-based approach returns matching processes in less than 1.2 seconds. The performance difference between both approaches increases with an increasing repository size. During our evaluation with up to 20,000 processes we did not experience a sudden performance regression. Thus, we can conclude that our offline index based model checking technique scales better than the monolithic and incremental approaches.

These results show that the (computational) efforts for building and maintaining offline indexes expedites the model checking phase such that results can be computed in a reasonable time frame. Also, the memory consumption of the index based model checking approach is considerably less as we allocated only 1GB of main memory for the index structures. The main memory consumption of the monolithic approach without index was not limited, so that more than 2GB of main memory could be used by the program before an out-of-memory-exception was thrown.

7.5 Related Work

The Semantic Web [BLHL01] as well as the Linked Data community [BHBL09] addresses the issue of semantic descriptions of Web page content but not of the path to be executed in order to reach pages. Search engine crawlers have been refined such that they can index the Web pages behind forms by learning and entering suitable values into the form elements [MKK^{+}08]. However, gaining access to these pages is insufficient to collect actual user browsing paths. Therefore, some companies collect user browsing behavior with the help of browser plug-ins such as toolbars.

Information retrieval [MRS08] has focused on analyzing such browsing paths or click trails of millions of users mainly for the purpose of improving Web search results. Click trails can be used as endorsements to rank search results more effectively [SWH10], trail destination pages can themselves be used as search results [WH10], and the concept of teleportation can be used to navigate directly to the desired page [TAAK04]. The statistics-based click analysis methods typically fail to consider semantics of user queries and pages. Furthermore, the models cannot differentiate whether a frequently used path actually satisfies the information need or not.

The Semantic Web has proposed the annotation of Web pages in order to describe the information content. Apart from the fact that still most of the Web pages are not annotated, it is hard to build a server-sided semantic information search engine, since a crawler will be unable to reach and index the semantic annotations within Deep Web pages. Linked Data separates the structured data from the traditional Web (and as a result also from the end users) completely. The Linked Data approach is primarily useful for application developers since end users cannot be expected to consume RDF directly. That is, end users still require human understandable applications to interact with, and furthermore the providers would continue to protect their valuable data by controlling access, e.g., with the help of user interaction elements such as Web forms.

Semantic search over RDF and Linked Data as in [LT10] enables querying Linked Data by traversing the Web of Linked Data. The completeness of query answering over Linked Data has been studied in [HS12]. Even if Linked Data query approaches were extended to support dynamic data, e.g., by integrating so-called Linked Data Services [SH11], the suitability of the approaches will remain limited to the publicly available free data only. That is, semantic search approaches (i) heavily rely on the availability of structured data, and (ii) providers are expected to provide access to their data through APIs. But, many providers do not follow this, and (iii) even if access to data is provided, the data of one provider is often not semantically aligned with data from other providers.

Navigational plans for data integration by Friedman et al. [FLM99] are related to our approach with respect to information need driven search for Web scripts. As the navigational paths are stored and returned as search results, this approach can deal with dynamic information published within Web pages. It presents a sound and complete algorithm for computing navigational paths for a given information query and set of source descriptions. Computed navigational paths are however subsets of the Web graph only. The algorithm cannot compute paths that consist of data flow between Web pages, which are not connected in the Web graph. Consider our running example with a Web page A listing the track chairs, and another page B listing publications of a given author. Assume that A is not linking to B. For a query for publications of track chairs, the algorithm presented in [FLM99] is not able to compute the wiring of outputs of A and inputs of B. Further, the proposed use of a mediated schema is hardly applicable to the Web. It is also not possible to define temporal constraints over navigational plans that answer the information need.

Search engine providers collect user click trails as well as the data they fill in the forms, e.g., with toolbar-like browser plug-ins. There exists a plethora of work, e.g., [BW08, SWH10], on predicting next step or the target Web page by analyzing click trails. Click trails can be seen as simple browsing processes as they are sequences without variables and without data flow. The major difference is that click trail analysis aims at helping users to find the relevant pages faster mostly based on syntactic analysis of pages that the user has visited in the current session. But users still have to know which pages are relevant and have to figure out what to do with which pages once they have been found. In contrast, in our approach each browsing process has a purpose, and we aim at finding the appropriate browsing processes for the current need. Another difference is in the acceptance of the underlying technology. End users often do not see any direct added value of a toolbar plug-in, which is a reason why toolbars are often delivered as part of some other software. In our approach, end users have direct incentives for sharing their browsing processes as well as full control on whether and with whom they share them.

Existing μ-calculus model checkers such as nuSMV[3] typically support symbolic model checking based on BDD and/or SAT. To the best of our knowledge there is no ready to use μ-calculus reasoner based on explicit state representation. The $\mu\mathcal{ALCQ}$ logic introduced in [GL97] extends $\mathcal{ALC}$ by fixpoint constructs and qualified number restrictions. It has been shown in [GL97] that $\mu\mathcal{ALC}$ is equivalent to μ-calculus. However, there does not exist any implementation of a $\mu\mathcal{ALC}$ reasoner that we could have used. Our implementation of a μ-calculus model checker was also necessary in order to have a common platform for comparing different model checking approaches.

An index over partially ordered complex behavioral constraints for the classification of complex services was introduced in [JAS12]. Constraints are mapped to entire services as opposed to a mapping to states as used in the present approach. Both approaches are complementary as (i) behavior classes from [JAS12] provide another offline index for complex constraints and (ii) our indexing structures allow to compute class memberships more efficiently and are still required for the verification of any constraints that were not covered by the service classification.

A metric to quantify process similarity based on behavioral profiles [KWW11], which is grounded in the Jaccard coefficient, leverages behavioral relations between process model activities. The metric is successfully evaluated towards its approximation of human similarity assessment. So far, we did not consider similarity of browsing processes. In general, if a process p simulates another process q and it is known that p is a match for a formula ϕ, then q can be directly added to the set of matches for ϕ. Incorporating such an index may bring further search performance gains.

[3]nuSMV symbolic model checker – http://nusmv.fbk.eu, retrieved 2013-08-15

7.6 Summary and Conclusions

Sophisticated use cases have complex information needs. Information needs that require information from various websites are not served satisfactorily by the state of the art search engines. This leaves end users spending a lot a time for searching and compiling required information from various Web pages. In this chapter, we targeted this problem from a completely new perspective. We build on a bottom-up approach that proposes capturing and sharing of end user browsing processes [AP12], as opposed to the top-down approach that requires annotated websites in the first place.

When the number of such Web browsing processes increases, efficient techniques for finding browsing processes suitable for an information need will be required. In this chapter, we presented an efficient logic-based technique for searching end user browsing processes. We have shown how they can be modeled with *suprimePDL*, which was developed and used for modeling complex Web services in the previous chapters. One of the major advantages of such a modeling approach is that it does not require manual semantic annotation of the whole browsing process. This makes the capturing of browsing processes easier, and does not demand users to do extra manual work before sharing their browsing processes. *suprimePDL* process descriptions can be mapped to an LTS representation. Therefore, known model checking methods can be used directly to check whether a given browsing process fulfills the given constraints.

In this chapter, our main focus was to develop an efficient search technique based on the known model checking techniques. We have presented offline and online indexes on top of the underlying model checking algorithm to achieve a scalable and efficient search for browsing processes. Regarding offline computable indexes, we have shown that indexing propositions and the states in which they hold, as well as indexing which actions are possible in which states brings significant performance gains. While the offline indexes only cover atomic formulae, the online index caches complex formulae that are used in a search query posed by an end user. Since the number of such queries is potentially infinite, we have incorporated a well-known randomized algorithm for deciding which answers should be kept in the index. The randomized algorithm is $2H_k$-competitive, which means that for any sequence of queries it requires at most $2H_k$ more time than an optimal algorithm for serving the sequence of queries. We have developed a lookup procedure for efficiently searching the formula or its subformulae in the online index, which brings further performance gains, as well as presented how the index can be maintained.

The online index structure favors requests that are posed frequently to be cached over time. This allows identifying frequent requests, which represent candidates for the introduction of new service classes. These new classes are likely to be used in many requests, as they encapsulate frequently requested service properties. Introducing new service classes triggered by frequent requests might even affect the usability of the discovery method by making it easier to formulate requests with the help of classes.

Chapter 8

Conclusion

In this thesis, we presented a holistic service discovery method that identifies services that accurately match the requirements specified in a discovery request. We employed a logic-based matchmaking technique that allows us to automatically and efficiently retrieve matching services based on a comprehensive set of properties. We introduced formalisms to describe and request different kinds of service properties. These formalisms provide the ability to include any domain specific properties. We focused on the modeling and reasoning over functional service properties that are applied to uniformly describe complex behavior and Web service profiles without altering their semantics with respect to discovery.

In the following, we summarize in Section 8.1 our contributions of this thesis and discuss how they answer the research question. In Section 8.2, we present an outlook on prospective extensions of our work.

8.1 Summary of Contributions

Service discovery is a fundamental technique that is applied in the development of service-based systems. It is not only required in traditionally centralized settings of software development, but also in the more modern trend of Web-based application development. There is a shift from the shipment of software systems that can be installed locally on computers to the provision of software as a service. We studied three different scenarios that require and assume the existence of automated and efficient service discovery methods that deliver logically matching services such that they can be easily used, e.g., as an integral part of a new system. We identified the requirements for automated and efficient discovery methods so that a large amount of services can be considered effectively. Furthermore, the logically correct search results enable developers and end users to use them without extensive human intervention, e.g., in order to evaluate the applicability of a discovered service for a given use case.

We also identified that the heterogeneity of service providers in open ecosystems demands for methods to address the data heterogeneity between service providers and consumers. For this reason, we employed formal representations of descriptions and requests. This design choice is also one of the drivers for enabling automation of the service discovery method. With existing works in the areas of ontology alignment and mapping, the data heterogeneity can be resolved.

While the use of logics and ontologies is merely a design choice, we extended the state of the art in the area of discovery of semantically described services by the following aspects. We proposed the use of a property-based service model, which provides the concepts of an upper service model. It allows us to combine functional and non-functional service properties in a unified way. In particular, it specifies a key-value structure of properties in service descriptions and accommodates a broad range of property types that we do not further constrain. Analogously, we proposed a structurally similar model for requests. It allows formulating requests that combine requirements on multiple properties. In contrast to many existing works, we propose the use of a distinct request formalism, which is different to the service description formalism. That is, the requests in our approach specify a set of desired values for each requested property.

We developed a discovery method that exploits comprehensive descriptions and expressive constraints on described properties. We developed a logic-based service matchmaker that builds on the μ-calculus model checking technique. While this has been used for the discovery of behavior descriptions before, we successfully applied it to the discovery of atomic services. By this, we introduced a novel approach to the problem of Semantic Web Service discovery. In general, model checking excels in (i) its computational complexity, which is considerably lower than the complexity of subsumption-based Semantic Web Service matchmakers, and (ii) its ability to treat service profiles and complex behavior descriptions equally.

Regarding the handling of Semantic Web Services in our discovery method, we proposed a method to capture knowledge base changes caused by the service execution. We clarified how descriptions and requests can include changes properly. Based on our modeling approach, we can discover services based on desired changes.

In order to increase the efficiency of the developed discovery method, we further extended the state of the art by introducing a formal service classification and index structures to speed up the discovery process. The proposed approach of a classification is novel in the sense that we include formal class definitions that can constrain the service behavior. The automated and consistent classification of services and the consistency of the induced classification hierarchy are features that cannot be provided by existing classification-based discovery approaches without formal class definitions. We even gain a higher expressivity of the service description formalism, as we allowed for an additional annotation of descriptions with classes. This classification increases the efficiency of our discovery method. This is due to an early reduction of the search space by materializing intermediate results in the form of classes.

The offline index structures we introduced also benefit from the precomputation of intermediate matchmaking results with respect to the semantic annotations of states and actions. At query answering time, we perform index lookups instead of ontology and μ-calculus reasoning. The online index materializes more intermediate results of more complex constraints. It is related to the service classes, as both can maintain the results of arbitrary constraints on the functionality, behavior, and non-functional properties. Whereas we assumed that the classification is given, the online index is automatically populated while processing a series of discovery requests. As the frequent requests are favored for caching, it allows identifying frequent requests, which represent candidates for the introduction of new service classes.

Review of the Research Questions

The development of our discovery method was driven by the requirements we identified in the motivating scenarios. Different use cases of a service discovery method implied various requirements. In order to provide an appropriate solution to the problem, we framed the open challenges in five research questions. The proposed discovery method answers all research questions. In the following, we review these questions and explain how we addressed them in this thesis. At the end of this section, we verify the main hypothesis.

Research Question 1. Can we develop a service discovery method that considers functional and non-functional properties in a unified way as well as detailed and abstract service behavior descriptions in service descriptions and requests?

We introduced in Section 4.1.1 a property-based model for service descriptions and an analogous model for requests in Section 4.2.2. It allows expressing any service property that can be reduced to a key-value structure, without dictating the concrete property types. The latter model enables us to precisely specify desired services by expressing desired combinations of different properties. We have shown in Section 4.3, how our discovery method exploits comprehensive descriptions and requests in order to identify services that fulfill various selection criteria. Hence, our method can discover services based on functional and non-functional requirements. In Chapter 5, we applied the same property-based models of descriptions and requests to atomic services. The matchmaking technique is applicable to both atomic services and services with complex behavior descriptions.

Research Question 2. How can we support the description of the functionality of state-changing services, which cause effects during their execution, such that reasoning and discovery based on state changes in descriptions and requests is enabled?

We focused on the functionality of services in Chapter 5. We developed a profile description of the service behavior with a state-based interpretation. We explicitly modeled the effects as changes in the provider knowledge in addition to the conditions that hold at the end of the service execution. We also considered the types of changes since logical formulae cannot express changes by itself. By the introduction of changes, our approach differs from previous Web service modeling approaches that describe a profile of the service functionality.

Research Question 3. Is it possible to apply an efficient logic-based matchmaking technique (model checking) to the matchmaking of service interface (profile) descriptions without losing the ability to model and reason about state changes?

We provided in Section 5.4 a model checking based matchmaking technique for the functionality of services, which is used to discover atomic services. Hence, it is also applicable to the Semantic Web Services discovery problem. We keep the strict separation of states from the descriptions and requests in the matchmaking technique. Furthermore, we considered the explicitly modeled changes to the state knowledge during matchmaking. It allows discovering services based on desired changes, which cannot be achieved by only taking the start and end states into account.

Research Question 4. Is it possible to automatically classify services correctly? And can such a classification be used to increase the efficiency of the discovery approach while the discovery result accuracy is not compromised?

We introduced in Chapter 6 a service classification, which is founded on formally described service classes. That is, each class features a defining constraint formula, which describes the functional or non-functional properties of the class members. Due to a formal class definition, we can use our matchmaking approach to consistently classify services into given classes automatically. A consistent hierarchy of service classes is also automatically derived. Given the classification of services, we can exploit it as an index structure to retrieve discovery results more efficiently if the discovery requests reuse existing classes. The semantics of the consistent classification that is used in the extended matchmaking method preserves the required accuracy of the discovery results. We have evaluated in Section 6.5 how the discovery performance is increased by the use of service classes.

Research Question 5. Can index structures be used to reduce the computational complexity of service discovery during the matchmaking of service requests with available service descriptions?

The service classification presented in Chapter 6 already induces an offline index. Service classes are the index entries. In Chapter 7, we presented offline indexing structures that are automatically populated. That is, the index structures materialize intermediate discovery results that can be computed offline, i.e., before service requests are processed. It is possible to compute them offline as the number of propositions is finite. At query answering time, the precomputed results are reused by an index lookup. As a consequence, which was evaluated in Section 7.4, the efficiency of the discovery method was increased, because we cut down on costly reasoning tasks for the evaluation of state propositions and annotations of the action. The online index presented in Section 7.3 was introduced as an additional index structure that caches more complex constraints from the discovery requests. Only frequent requests are cached in the index structure, due to its limited size. The stream of incoming discovery requests is the main factor for the expected performance, which is therefore hard to estimate. The index structure populated online can be used to identify frequent discovery requests, which allows us to match repeated requests more efficiently by an index lookup. They can also trigger the creation of new service classes, which might even affect the usability of the discovery method.

Main Hypothesis. Services can be automatically, effectively, and efficiently discovered based on a verification of the constraints of expressive service requests over comprehensive service descriptions.

In this thesis we developed a discovery method that verifies our main hypothesis as we have argued above for the five research questions.

8.2 Future Work and Outlook

We now provide an overview on the topics and open challenges that are closely related to our work on service discovery but have not been within our scope. They provide a starting point for extending this work or for applying our work in end user ready scenarios.

End User Ready Request Language

The development of the discovery method was motivated by use cases in which service discovery is used by a machine or by an end user. It is easy to let machines, i.e., software programs, create structured service requests as we considered them in this work. End users may encounter problems in specifying service requests in such a way, because they are used to search systems, which either accept keyword-based queries or provide a supportive user interface.

So far, we did not consider how users can formulate requests. An end user ready request language should be able to abstract from the underlying formalisms while retaining the formalism's expressivity to a large extent. Otherwise, if an end user request cannot express the complete set of requirements, it is often hard to discover services that can be immediately used. In order to tackle this challenge, automated methods to elicit requirements, e.g., from the user profile or her context, can help to suggest request completions in a discovery interface. The use of forms and menus can guide users in expressing their requirements while abstracting from the concrete syntax.

Cost Benefit Analysis of the Service Classification

With a focus on developing methods and tools that allow end users to use our discovery method more easily, it becomes possible to evaluate the impact of the classification presented in Chapter 6 on the usability. The economy of time with respect to the reuse of service classes in requests can be opposed to the search for classes based on their name or a full-text description.

It is possible to increase the efficiency gains of using a service classification even when a request is specified without the use of classes. Query rewriting techniques can be developed in order to replace as many explicit constraints as possible by existing classes. Furthermore, the variable mappings between the explicit constraints and a set of replacement classes have to be integrated. Once the classes replace explicit constraints of a request, the matchmaking time for the evaluation of the request using classes is lower than for the one without classes. In Section 6.2.3, we described how the subclass relationship between classes is computed. This method can be the starting point for query rewriting techniques. The complexity of comparing explicit constraints in requests with a large set of classes can considerably increase the query answering time, if the substitution requires too much time. Then, the class hierarchy can be exploited to prune the number of class comparisons.

Integration of Ranking

In our work, we used the principle that a request is verified for all service descriptions of a repository. In a discovery system, a global ranking of the services in the repository can be computed offline. It means that an ordering of services based on objective preferences is added to the repository. The discovery method can then be extended such that the ordering from the repository determines the order in which the request is verified. The matching services can be immediately returned to the user. That is, a stream of discovery results is produced and matching services are returned while the complete set of results is continued to be computed. Thus, the user receives first results quickly. So far, we could not consider a global ranking in our discovery method, as the underlying ontology reasoners do not allow to influence the order of execution of reasoning tasks. They also do not return result streams.

It is an even more interesting aspect to combine ranking and matchmaking in a more sophisticated way to build a search system. Given global and user specific preferences that imply an ordering of services, the computational complexity of evaluating different kinds of preference terms varies. The individual steps for computing a ranking for different preferences dovetails the set of matchmaking operations. One goal can be to provide search results to a user in an interactive search interface as fast as possible. Alternatively, search results can be passed to a machine, which continues its computation every time a new search result is returned. In order to determine how ranking integrates best with matchmaking, the complexities of evaluating different preference terms and the selectivity of individual discovery constraints have to be estimated. It allows creating an execution plan that specifies the order of preference term evaluation steps and constraint verification steps. Such a plan, which is also known as query execution plan in database systems, is a workflow that, if explicitly described, can be easily deployed on external systems and stored for later reuse.

Instead of using a ranking of services to influence the order in which a request is evaluated, it is also possible to extend our discovery method to only consider top-k ranked services for the matchmaking. That is, only the most promising candidates are considered for matchmaking. In comparison to our method, such top-k based discovery methods will lose completeness of the result set, but it can be a further step towards the development of scalable discovery methods.

Knowledge Base Identification

We already mentioned in Section 4.1.1 the lack of an ability to distinguish the knowledge bases of different providers within existing service modeling frameworks. Representing the knowledge, e.g., using description logics, allows modeling only one state and only one actor at the same time. In complex service systems, where multiple actors can be involved in the service provision, the knowledge of each actor in a common state should be disconnected. Otherwise, logic inconsistencies can occur and it is not possible to reason about the knowledge of each actor.

In our approach, we either have to assume that only one actor provides the functionality or that all actors and their behavior are explicitly described. In a behavior description, the agent invocation describes the invocation of external processes, which can be provided by other actors. However, a separation of the actors' knowledge bases is required when a local action is executed by multiple actors. Adding provenance annotations to each statement in the knowledge of each state can be one approach to make the separation explicit. Ontology reasoners can then be used to reason over the provenance information.

Outlook

We observe that the initial Semantic Web vision [BLHL01] with autonomous agents browsing the Web and utilizing Semantic Web Services automatically is shifting. Currently, more data-centric trends culminating in Linked Data and Big Data related research efforts can be perceived. Nevertheless, services remain an integral part of these research efforts and emerge, e.g., as potential enablers of a read-write access to Linked Data in a layer on top of the data sets.

We believe that service orientation and especially the Web-mediated functionalities designed for end users will gain momentum in the near future. In order to empower end users with the capability to organize, customize, accelerate, reuse, and share their personal and business activities carried out in the Web, the functionalities and browsing processes need to be modeled explicitly. Based on their description, the user efforts can be reduced tremendously by applying automated methods like discovery, composition, and execution.

Service composition is one of the most appealing and, thus, prominently aimed at use cases of service discovery. Automated composition methods, e.g., based on AI planning, are not suitable in every scenario as often too many unusable plans are computed. Often it can be more efficient to manually specify templates of processes and workflows that contain abstract activities, which can be instantiated by a binding to concrete services. This approach still provides the flexibility to customize the templates later for a use case at hand or to react to a changing environment by adapting the process. For example, in the domain of scientific workflows, a workflow designer has to manually determine the overall structure of the workflows. Then, the resulting template uses service discovery to automatically bind single workflow activities to services.

In this thesis, we provided a service discovery method that treats services with complex behavior and atomic services uniformly, and produces accurate results enabling their immediate use. Our approaches increase the efficiency of discovery and allow us to utilize large service description repositories. Our method is applicable in current scenarios like the ones we introduced above and also provides features for the support of upcoming scenarios. We have shown in Chapter 7 that our discovery method can even be used to tackle open challenges of the information search problem in the Web. The presented service discovery method enables the development of service-oriented systems and also aids in tasks like composition and execution.

List of Tables

List of Figures

Bibliography

[Abe09] Sven Abels. SOA4All: Enabling a Web of billions of services. *International Journal of Interoperability in Business Information Systems (IBIS)*, 8:35–37, 2009.

[ABK$^+$04] Sinuhé Arroyo, Christoph Bussler, Jacek Kopecký, Rubén Lara, Axel Polleres, and Michael Zaremba. Web service capabilities and constraints in WSMO. Technical report, Digital Enterprise Research Institute (DERI), Aug. 2004. http://www.w3.org/2004/08/ws-cc/wsmo-20040903, retrieved 2013-08-15.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Proc. of 6th International Conference on Business Process Management (BPM), Milan, Italy, Sept. 2-4, 2008*, volume 5240 of *LNCS*, pages 326–341. Springer, 2008.

[AF00] Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):171–210, 2000.

[AFM$^+$05] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - WSDL-S. W3C member submission, W3C, Nov. 2005. http://www.w3.org/Submission/WSDL-S, retrieved 2013-08-15.

[Aga01] Sudhir Agarwal. Algorithms for general caching problems. Diploma thesis, Universität Dortmund, 2001.

[Aga07a] Sudhir Agarwal. *Formal Description of Web Services for Expressive Matchmaking*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2007.

[Aga07b] Sudhir Agarwal. A goal specification language for automated discovery and composition of Web services. In *Proc. of IEEE/WIC/ACM International Conference on Web Intelligence (WI), Silicon Valley, CA, USA, Nov. 2-5, 2007*, pages 528–534. IEEE Computer Society, 2007.

[AJ10] Sudhir Agarwal and Martin Junghans. Swapping out coordination of Web processes to the Web browser. In Antonio Brogi, Cesare Pautasso, and George Angelos Papadopoulos, editors, *Proc. of 8th IEEE European Conference on Web Services (ECOWS), Ayia Napa, Cyprus, Dec. 1-3, 2010*, pages 115–122. IEEE Computer Society, 2010.

[AJ11] Sudhir Agarwal and Martin Junghans. Meaningful service classifications for flexible service descriptions. In *Proc. of World Congress on Services (SERVICES), Washington, DC, USA, July 4-9, 2011*, pages 85–86. IEEE Computer Society, 2011.

[AJ13] Sudhir Agarwal and Martin Junghans. Towards simulation-based similarity of end user browsing processes. In Florian Daniel, Peter Dolog, and Qing Li, editors, *Proc. of 13th International Conference on Web Engineering (ICWE), Aalborg, Denmark, July 8-12, 2013*, volume 7977 of *LNCS*, pages 216–223. Springer, 2013.

[AJF$^+$09] Sudhir Agarwal, Martin Junghans, Olivier Fabre, Ioan Toma, and Jean-Pierre Lorre. First service discovery prototype. Deliverable D5.3.1, SOA4All, Sept. 2009. http://www.soa4all.eu/file-upload.html?func=fileinfo&id=123, retrieved 2013-08-15.

[AJN10a] Sudhir Agarwal, Martin Junghans, and Barry Norton. Second service discovery prototype. Deliverable D5.3.2, SOA4All, Sept. 2010. http://www.soa4all.eu/file-upload.html?func=fileinfo&id=236, retrieved 2013-08-15.

[AJN10b] Sudhir Agarwal, Martin Junghans, and Barry Norton. Second service ranking prototype. Deliverable D5.4.2, SOA4All, Aug. 2010. http://www.soa4all.eu/file-upload.html?func=fileinfo&id=237, retrieved 2013-08-15.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[ALS09] Sudhir Agarwal, Steffen Lamparter, and Rudi Studer. Making Web services tradable: A policy-based approach for specifying preferences on Web service properties. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):11–20, 2009.

[AP12] Sudhir Agarwal and Charles J. Petrie. An alternative to the top-down semantic Web of services. *IEEE Internet Computing*, 16(5):94–97, 2012.

[ARA08] Sudhir Agarwal, Sebastian Rudolph, and Andreas Abecker. Semantic description of distributed business processes. In *AI Meets Business Rules* *and Process Management, Papers from the 2008 AAAI Spring Symposium, Technical Report SS-08-01, Stanford, CA, USA, Mar. 26-28, 2008*, pages 1–11. AAAI, 2008.

[AS04] H. Peter Alesso and Craig F. Smith. *Developing Semantic Web Services.* A K Peters/CRC Press, 2004.

[AS06] Luis Araujo and Martin Spring. Services, products, and the institutional structure of production. *Industrial Marketing Management*, 35(7):797–805, 2006.

[BAT97] Pim Borst, Hans Akkermans, and Jan L. Top. Engineering ontologies. *International Journal of Human-Computer Studies*, 46(2):365–406, 1997.

[BBL11] David Beckett and Tim Berners-Lee. Turtle - terse RDF triple language. W3C team submission, W3C, Mar. 2011. http://www.w3.org/TeamSubmission/turtle/, retrieved 2013-08-15.

[BBvB$^+$01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. http://agilemanifesto.org/, retrieved 2013-08-15, 2001.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[Ber01] Michael K. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.

[BFH$^+$09] Barry Bishop, Florian Fischer, Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, Yiorgos Trimponias, and Gulay Unel. Defining the features of the WSML-DL v2.0 language. Deliverable D3.1.3, SOA4All, Sept. 2009. http://www.soa4all.eu/file-upload.html?func=fileinfo&id=82, retrieved 2013-08-15.

[BFM11] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175(7-8):1308–1345, 2011.

[BG04] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF Schema. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/rdf-schema, retrieved 2013-08-15.

[BGG$^+$06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proc. of 8th International Conference on Coordination Models and Languages (COORDINATION), Bologna, Italy, June 14-16, 2006*, volume 4038 of *LNCS*, pages 63–81. Springer, 2006.

[BH91] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In John Mylopoulos and Raymond Reiter, editors, *Proc. of 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, Aug. 24-30, 1991*, pages 452–457. Morgan Kaufmann, 1991.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[BHL$^+$02] Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia P. Sycara. DAML-S: Web service description for the Semantic Web. In Ian Horrocks and James Hendler, editors, *The Semantic Web: Proc. of 1st International Semantic Web Conference (ISWC), Sardinia, Italy, June 9-12, 2002*, volume 2342 of *LNCS*, pages 348–363. Springer, 2002.

[BHL$^+$05] Boualem Benatallah, Mohand-Said Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. On automating Web services discovery. *VLDB Journal*, 14(1):84–96, 2005.

[BHSS09] Saartje Brockmans, Peter Haase, Luciano Serafini, and Heiner Stuckenschmidt. Formal and conceptual comparison of ontology mapping languages. In Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors, *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*, pages 267–291. Springer, 2009.

[BKCvD09] Benjamin Blau, Jan Kramer, Tobias Conte, and Clemens van Dinther. Service value networks. In Birgit Hofreiter and Hannes Werthner, editors, *Proc. of IEEE Conference on Commerce and Enterprise Computing (CEC), Vienna, Austria, July 20-23, 2009*, pages 194–201. IEEE Computer Society, 2009.

[BKO$^+$11] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web Journal*, 2(1):33–42, 2011.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 5(284):34–43, 2001.

[BM05] Michael Bolin and Robert C. Miller. Naming page elements in end-user Web automation. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[BN03] Franz Baader and Werner Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[Boc08] Jürgen Bock. Parallel computation techniques for ontology reasoning. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web: Proc. of 7th International Semantic Web Conference (ISWC), Karlsruhe, Germany, Oct. 26-30, 2008*, volume 5318 of *LNCS*, pages 901–906. Springer, 2008.

[Bör98] Egon Börger. High level system design and analysis using abstract state machines. In Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullmann, editors, *Proc. of International Workshop on Current Trends in Applied Formal Methods (FM-Trends), Boppard, Germany, Oct. 7-9, 1998*, volume 1641 of *LNCS*, pages 1–43. Springer, 1998.

[Bra98] Julian C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, 1998.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[BS01] Julian Bradfield and Colin Stirling. Modal logics and mu-calculi: an introduction. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier, 2001.

[BS06] Julian Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.

[BvHH$^+$04] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web ontology language. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/owl-ref, retrieved 2013-08-15.

[BW99] Rebecca F. Bruce and Janyce Wiebe. Recognizing subjectivity: A case study of manual tagging. *Natural Language Engineering*, 5(2):187–205, 1999.

[BW08] Mikhail Bilenko and Ryen W. White. Mining the search trails of surfing crowds: identifying relevant websites from user activity. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proc. of 17th International Conference on World Wide Web (WWW), Beijing, China, Apr. 21-25, 2008*, pages 51–60. ACM, 2008.

[BWR$^+$05] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered Web pages. In Patrick Baudisch, Mary Czerwinski, and Dan R. Olsen, editors, *Proc. of 18th Annual ACM Symposium on User Interface Software and Technology (UIST), Seattle, WA, USA, Oct. 23-26, 2005*, pages 163–172. ACM, 2005.

[BWR09] Adam Barker, Christopher D. Walton, and David Robertson. Choreographing Web services. *IEEE Transactions on Services Computing*, 2(2):152–166, 2009.

[CBF05] Ion Constantinescu, Walter Binder, and Boi Faltings. Flexible and efficient matchmaking and ranking in service directories. In *Proc. of IEEE International Conference on Web Services (ICWS), Orlando, FL, USA, July 11-15, 2005*, pages 5–12. IEEE Computer Society, 2005.

[CBMK10] Jorge Cardoso, Alistair P. Barros, Norman May, and Uwe Kylau. Towards a unified service description language for the Internet of services: Requirements and first developments. In *Proc. of IEEE International Conference on Services Computing (SCC), Miami, FL, USA, July 5-10, 2010*, pages 602–609. IEEE Computer Society, 2010.

[CC06] Miguel Ángel Corella and Pablo Castells. Semi-automatic semantic-based Web service classification. In Johann Eder and Schahram Dustdar, editors, *Proc. of Business Process Management Workshops: BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, Sept. 4-7, 2006*, volume 4103 of *LNCS*, pages 459–470. Springer, 2006.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3C note, W3C, Mar. 2001. http://www.w3.org/TR/wsdl, retrieved 2013-08-15.

[CFB04] Ion Constantinescu, Boi Faltings, and Walter Binder. Large scale, type-compatible service composition. In *Proc. of IEEE International Conference on Web Services (ICWS), San Diego, CA, USA, June 6-9, 2004*, pages 506–513. IEEE Computer Society, 2004.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking.* MIT Press, 2001.

[CHvRR04] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI spec technical committee draft, version 3.0.2. OASIS committee draft, OASIS, Oct. 2004. http://uddi.org/pubs/uddi_v3.htm, retrieved 2013-08-15.

[CSHG09] Viorica R. Chifu, Ioan Salomie, Ioana Harsa, and Marius Gherga. Semantic Web service composition method based on fluent calculus. In Stephen M. Watt, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, and Daniela Zaharie, editors, *Proc. of 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, Sept. 26-29, 2009*, pages 325–332. IEEE Computer Society, 2009.

[CVW08] Jorge Cardoso, Konrad Voigt, and Matthias Winkler. Service engineering for the internet of services. In Joaquim Filipe and José Cordeiro, editors, *Proc. of 10th International Conference on Enterprise Information Systems (ICEIS), Barcelona, Spain, June 12-16, 2008*, volume 19 of *Lecture Notes in Business Information Processing*, pages 15–27. Springer, 2008.

[dBBD$^+$05] Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kiffer, Brigitta König-Ries, Jacek Kopecký, Rubén Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dimitru Roman, James Scicluna, and Michael Stollberg. Web service modeling ontology (WSMO). W3C member submission, W3C, June 2005. http://www.w3.org/Submission/WSMO, retrieved 2013-08-15.

[dBFK$^+$08] Jos de Bruijn, Dieter Fensel, Mick Kerrigan, Uwe Keller, Holger Lausen, and James Scicluna. *Modeling Semantic Web Services: The Web Service Modeling Language*. Springer, Berlin, 2008.

[dBKZF09] Jos de Bruijn, Mick Kerrigan, Michael Zaremba, and Dieter Fensel. Semantic Web services. In Staab and Studer [SS09], pages 617–636.

[Deb04] Sumanta Deb. Designing the agile enterprise - managed migration to SOA holds the key. In Hamid R. Arabnia, Olaf Droegehorn, and S. Chatterjee, editors, *Proc. of International Conference on Internet Computing (IC), Volume 2 & Proc. of the International Symposium on Web Services & Applications (ISWS), Las Vegas, Nevada, USA, June 21-24, 2004*, pages 749–756. CSREA Press, 2004.

[DFD$^+$09] John Domingue, Dieter Fensel, John Davies, Rafael González-Cabero, and Carlos Pedrinaci. The service Web: a Web of billions of services. In Georgios Tselentis, John Domingue, Alex Galis, Anastasius Gavras, David Hausheer, Srdjan Krco, Volkmar Lotz, and Theodore Zahariadis, editors, *Towards the Future Internet - A European Research Perspective*, pages 203–216. IOS Press, 2009.

[DFGC08] John Domingue, Dieter Fensel, and Rafael González-Cabero. SOA4All, enabling the SOA revolution on a world wide scale. In *Proc. of 2th IEEE International Conference on Semantic Computing (ICSC), Aug. 4-7, 2008, Santa Clara, CA, USA*, pages 530–537. IEEE Computer Society, 2008.

[DLC$^+$07] Xinguo Deng, Ziyu Lin, Weiqing Chen, Ruliang Xiao, Lina Fang, and Ling Li. Modeling Web service choreography and orchestration with colored Petri nets. In Wenying Feng and Feng Gao, editors, *Proc. of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Qingdao, China, July 30-Aug. 1, 2007*, pages 838–843. IEEE Computer Society, 2007.

[DM00] Francesco M. Donini and Fabio Massacci. EXPTIME tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[Don03] Francesco M. Donini. Complexity of reasoning. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 96–136. Cambridge University Press, 2003.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In Jaco W. de Bakker and Jan van Leeuwen, editors, *Proc. of 7th International Colloquium on Automata, Languages and Programming (ICALP), Noordweijkerhout, The Netherlands, July 14-18, 1980*, volume 85 of *LNCS*, pages 169–181. Springer, 1980.

[ECPB13] Thomas Erl, Benjamin Carlyle, Cesare Pautasso, and Raj Balasubramanian. *SOA with REST - Principles, Patterns and Constraints for Building Enterprise Solutions with REST*. The Prentice Hall Service Technology Series. Pearson, 2013.

[Ehr06] Marc Ehrig. *Ontology Alignment: Bridging the Semantic Gap*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2006.

[EJ99] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 1999.

[EL86] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. of Symposium on Logic in Computer Science (LICS), Cambridge, MA, USA, June 16-18, 1986*, pages 267–278. IEEE Computer Society, 1986.

[ES07] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2007.

[FFH$^+$03] Christian Facciorusso, Simon Field, Rainer Hauser, Yigal Hoffner, Robert Humbel, René Pawlitzek, Walid Rjaibi, and Christine Siminitz. A Web services matchmaking engine for Web services. In Kurt Bauknecht, A. Min Tjoa, and Gerald Quirchmayr, editors, *Proc. of 4th International Conference on E-Commerce and Web Technologies (EC-Web), Prague, Czech Republic, Sept. 2-5, 2003*, volume 2738 of *LNCS*, pages 37–49. Springer, 2003.

[FFK$^+$10] Dieter Fensel, Florian Fischer, Jacek Kopecký, Reto Krummenacher, Dave Lambert, and Tomas Vitvar. WSMO-Lite: Lightweight semantic descriptions for services on the Web. W3C member submission, W3C, Aug. 2010. http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823, retrieved 2013-08-15.

[FFST11] Dieter Fensel, Federico Michele Facca, Elena Simperl, and Ioan Toma. *Semantic Web Services*. Springer, 2011.

[Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1996.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[FLM99] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In James Hendler and Devika Subramanian, editors, *Proc. of 16th AAAI National Conference on Artificial Intelligence, Orlando, FL, USA, July 18-22, 1999*, pages 67–73. AAAI Press / The MIT Press, 1999.

[Flo67] Robert W. Floyd. Assigning meanings to programs. In Jacob T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proc. of Symposia in Applied Mathematics*, volume 19, pages 19–32, 1967.

[Fra99] Piero Fraternali. Tools and approaches for developing data-intensive Web applications: A survey. *ACM Computing Surveys*, 31(3):227–263, 1999.

[GCGPP+08] Raul Garcia-Castro, Asunción Gómez-Pérez, Charles J. Petrie, Emanuele Della Valle, Ulrich Küster, Michal Zaremba, and M. Omair Shafiq, editors. *Proc. of 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC), Tenerife, Spain, June 1-2, 2008*, volume 359. CEUR, 2008.

[GCTB01] Javier González-Castillo, David Trastour, and Claudio Bartolini. Description logics for matchmaking of services. In Günther Görz, Volker Haarslev, Carsten Lutz, and Ralf Möller, editors, *Proc. of Workshop on Applications of Description Logics, Vienna, Austria, Sept. 18, 2001*, volume 44. CEUR, 2001.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part 1: Messaging framework (second edition). W3C recommendation, W3C, Apr. 2007. http://www.w3.org/TR/soap12-part1/, retrieved 2013-08-15.

[GHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In Gusztáv Hencsey, Bebo White, Yih-Farn Robin Chen, László Kovács, and Steve Lawrence, editors, *Proc. of 12th International World Wide Web Conference (WWW), Budapest, Hungary, May 20-24, 2003*, pages 48–57. ACM, 2003.

[GJR+13] José María García, Martin Junghans, David Ruiz, Sudhir Agarwal, and Antonio Ruiz Cortés. Integrating semantic Web services ranking mechanisms using a common preference model. *Knowledge-Based Systems*, 49(0):22–36, 2013.

[GL97] Giuseppe De Giacomo and Maurizio Lenzerini. A uniform framework for concept definitions in description logics. *Journal of Artificial Intelligence Research*, 6:87–110, 1997.

[GLA$^+$04] Stephan Grimm, Steffen Lamparter, Andreas Abecker, Sudhir Agarwal, and Andreas Eberhart. Ontology based specification of Web service policies. In Peter Dadam and Manfred Reichert, editors, *Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V., Band 2, Ulm, Germany, Sept. 20-24, 2004*, volume 51 of *LNI*, pages 579–583. GI, 2004.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann, San Francisco, CA, USA, 2004.

[GNT14] Malik Ghallab, Dana Nau, and Paolo Traverso. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, 208:1–17, 2014.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In Staab and Studer [SS09].

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[Gur94] Yuri Gurevich. Evolving algebras. In Bjorn Pehrson and Imre Simon, editors, *Proc. of IFIP 13th World Computer Congress on Technology and Foundations - Information Processing, Hamburg, Germany, Aug. 28-Sept. 2, 1994*, volume 1, pages 423–427. North-Holland, 1994.

[HA28] David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik.* Springer, 1928.

[HA10] Julia Hoxha and Sudhir Agarwal. Semi-automatic acquisition of semantic descriptions of processes in the Web. In Jimmy Xiangji Huang, Irwin King, Vijay V. Raghavan, and Stefan Rueger, editors, *Proc. of IEEE/WIC/ACM International Conference on Web Intelligence (WI), Toronto, Canada, Aug. 31-Sept. 3, 2010*, pages 256–263. IEEE Computer Society, 2010.

[HA12] Julia Hoxha and Sudhir Agarwal. Datenintegration und Verwaltungskomponente für Logistik-Dienste. Deliverable M4.7, InterLogGrid, Jan. 2012.

[Hal05] Alon Y. Halevy. Why your data won't mix: Semantic heterogeneity. *ACM Queue*, 3(8):50–58, 2005.

[Hay04] Patrick J. Hayes. RDF semantics. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/rdf-mt, retrieved 2013-08-15.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* Foundations of Computing. MIT Press, 1988.

[Hil77] Peter Hill. On goods and services. *Review of Income and Wealth*, 23(4):315–338, 1977.

[HJA12] Julia Hoxha, Martin Junghans, and Sudhir Agarwal. Enabling semantic analysis of user browsing patterns in the Web of data. In *Proc. of 2nd International Workshop on Usage Analysis and the Web of Data (USEWOD), Lyon, France, Apr. 17, 2012*, 2012.

[HKP$^+$09] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer.* W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-primer, retrieved 2013-08-15.

[HKR09] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies.* Chapman & Hall/CRC, 1st edition, 2009.

[HLM99] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[HM08] Volker Haarslev and Ralf Möller. On the scalability of description logic instance retrieval. *Journal of Automated Reasoning*, 41(2):99–142, 2008.

[Hoa69] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[HS12] Andreas Harth and Sebastian Speiser. On completeness classes for query evaluation on Linked Data. In Jörg Hoffmann and Bart Selman, editors, *Proc. of 26th AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, July 22-26, 2012.* AAAI Press, 2012.

[HTN08] Ramy Ragab Hassen, Farouk Toumani, and Lhouari Nourine. Web services composition is decidable. In *Proc. of 11th International Workshop on the Web and Databases (WebDB), Vancouver, BC, Canada, June 13, 2008*, 2008.

[HZB$^+$06] Duncan Hull, Evgeny Zolin, Andrey Bovykin, Ian Horrocks, Ulrike Sattler, and Robert Stevens. Deciding semantic matching of stateless services. In *Proc. of 21st National Conference on Artificial Intelligence and 18th Conference on Innovative Applications of Artificial Intelligence Conference, Boston, MA, USA, July 16-20, 2006*, pages 1319–1324. AAAI Press, 2006.

[IWA91] Neil Iscoe, Gerald B. Williams, and Guillermo Arango. Domain modeling for software engineering. In Les Belady, David R. Barstow, and Koji Torii, editors, *Proc. of 13th International Conference on Software Engineering (ICSE), Austin, TX, USA, May 13-17, 1991*, pages 340–343. IEEE Computer Society / ACM Press, 1991.

[JA10] Martin Junghans and Sudhir Agarwal. Web service discovery based on unified view on functional and non-functional properties. In *Proc. of 4th IEEE*

International Conference on Semantic Computing (ICSC), Pittsburgh, PA, USA, Sept. 22-24, 2010, pages 224–227. IEEE Computer Society, 2010.

[JA11] Martin Junghans and Sudhir Agarwal. Wissensnetzwerke im Grid - Evaluierung der Suchfunktion. Deliverable D3.2.6, WisNetGrid, June 2011. (German).

[JA12] Martin Junghans and Sudhir Agarwal. Wissensnetzwerke im Grid - Entwicklung von Methoden und Werkzeugen für Workflowmodellierung und -Ausführung. Deliverable D3.3.2, WisNetGrid, Mar. 2012. (German).

[JA13] Martin Junghans and Sudhir Agarwal. Efficient search for Web browsing recipes. In *Proc. of IEEE International Conference on Web Services (ICWS), Santa Clara, CA, USA, June 27-July 2, 2013*, page to be published. IEEE Computer Society, 2013.

[JAS10] Martin Junghans, Sudhir Agarwal, and Rudi Studer. Towards practical semantic Web service discovery. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications, Proc. of 7th Extended Semantic Web Conference (ESWC), Part II, Heraklion, Crete, Greece, May 30-June 3, 2010*, volume 6089 of *LNCS*, pages 15–29. Springer, 2010.

[JAS12] Martin Junghans, Sudhir Agarwal, and Rudi Studer. Behavior classes for specification and search of complex services and processes. In Carole A. Goble, Peter P. Chen, and Jia Zhang, editors, *Proc. of IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, June 24-29, 2012*, pages 343–350. IEEE Computer Society, 2012.

[Jaz07] Mehdi Jazayeri. Some trends in Web application development. In Lionel C. Briand and Alexander L. Wolf, editors, *Proc. of Workshop on the Future of Software Engineering (FOSE), Minneapolis, MN, USA, May 23-25, 2007*, pages 199–213, 2007.

[JJ89] Claude Jard and Thierry Jéron. On-line model checking for finite linear temporal logic specifications. In Joseph Sifakis, editor, *Proc. of International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, June 12-14, 1989*, volume 407 of *LNCS*, pages 189–196. Springer, 1989.

[Jun13] Martin Junghans. A process-oriented view of website mediated functionalities. In Craig A. Knoblock, Kai-Uwe Sattler, and Rudi Studer, editors, *Proc. of Dagstuhl Seminar 13252 on Interoperation in Complex Information Ecosystems*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 2013.

[KB04] Mark Klein and Abraham Bernstein. Toward high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36, 2004.

[KDJ06] Sofien Khemakhem, Khalil Drira, and Mohamed Jmaiel. SEC: A search engine for component based software development. In Hisham Haddad, editor, *Proc. of ACM Symposium on Applied Computing (SAC), Dijon, France, Apr. 23-27, 2006*, pages 1745–1750. ACM, 2006.

[KDPS10] Reto Krummenacher, John Domingue, Carlos Pedrinaci, and Elena Simperl. SOA4All: Towards a global service delivery platform. In Georgios Tselentis, Alex Galis, Anastasius Gavras, Srdjan Krco, Volkmar Lotz, Elena Simperl, Burkhard Stiller, and Theodore Zahariadis, editors, *Towards the Future Internet - Emerging Trends from European Research*, pages 161–172. IOS Press, 2010.

[KFS06] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. Automated semantic Web service discovery with OWLS-MX. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *Proc. of 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Hakodate, Japan, May 8-12, 2006*, pages 915–922. ACM, 2006.

[KFS09] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. OWLS-MX: A hybrid semantic Web service matchmaker for OWL-S services. *Journal of Web Semantics*, 7(2):121–133, 2009.

[KK06] Frank Kaufer and Matthias Klusch. WSMO-MX: A logic programming based hybrid service matchmaker. In *Proc. of 4th IEEE European Conference on Web Services (ECOWS), Zurich, Switzerland, Dec. 4-6, 2006*, pages 161–170. IEEE Computer Society, 2006.

[KK12] Matthias Klusch and Patrick Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.

[KKLF11] Christopher Klinkmüller, Robert Kunkel, André Ludwig, and Bogdan Franczyk. The logistics service engineering and management platform: Features, architecture, implementation. In Witold Abramowicz, editor, *Proc. of 14th International Conference on Business Information Systems (BIS), Poznan, Poland, June 15-17, 2011*, volume 87 of *LNBIP*, pages 242–253. Springer, 2011.

[KLL$^+$05] Uwe Keller, Rubén Lara, Holger Lausen, Axel Polleres, and Dieter Fensel. Automatic location of services. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Proc. of 2nd European Semantic Web Conference (ESWC), Heraklion, Crete, Greece, May 29-June 1, 2005*, volume 3532 of *LNCS*, pages 1–16. Springer, 2005.

[KLS05] Pal Krogdahl, Gottfried Luef, and Christoph Steindl. Service-oriented agility: An initial analysis for the use of agile methods for SOA development. In *Proc.*

of IEEE International Conference on Services Computing (SCC), Orlando, FL, USA, July 11-15, 2005, pages 93–100. IEEE Computer Society, 2005.

[KLS06] Uwe Keller, Holger Lausen, and Michael Stollberg. On the semantics of functional descriptions of Web services. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, Proc. of 3rd European Semantic Web Conference (ESWC), Budva, Montenegro, June 11-14, 2006*, volume 4011 of *LNCS*, pages 605–619. Springer, 2006.

[KLS13] Paul Karänke, Jörg Leukel, and Vijayan Sugumaran. Ontology-based QoS aggregation for composite Web services. In *11. Internationale Tagung Wirtschaftsinformatik, Leipzig, Germany, Feb. 27-Mar. 1, 2013*, pages 1343–1357, 2013.

[KMP06] László Kovács, András Micsik, and Peter Pallinger. Two-phase semantic Web service discovery method for finding intersection matches using logic programming. In M. Omair Shafiq, editor, *Proc. of Workshop on Semantics for Web Services (SemWS), Zurich, Switzerland, Dec. 4-6, 2006*, volume 316. CEUR, 2006.

[KNSP09] Reto Krummenacher, Barry Norton, Elena Simperl, and Carlos Pedrinaci. SOA4All: Enabling web-scale service economies. In *Proc. of 3rd IEEE International Conference on Semantic Computing (ICSC), Berkeley, CA, USA, Sept. 14-16, 2009*, pages 535–542. IEEE Computer Society, 2009.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP06] Kyriakos Kritikos and Dimitris Plexousakis. Semantic QoS metric matching. In *Proc. of 4th IEEE European Conference on Web Services (ECOWS), Zurich, Switzerland, Dec. 4-6, 2006*, pages 265–274. IEEE Computer Society, 2006.

[KP07] Kyriakos Kritikos and Dimitris Plexousakis. Requirements for QoS-based Web service description and discovery. In *Proc. of 31st Annual International Computer Software and Applications Conference (COMPSAC), Beijing, China, July 24-27, 2007*, pages 467–472. IEEE Computer Society, 2007.

[KP09] Kyriakos Kritikos and Dimitris Plexousakis. Mixed-integer programming for QoS-based Web service matchmaking. *IEEE Transactions on Services Computing*, 2(2):122–139, 2009.

[KP12] Kyriakos Kritikos and Dimitris Plexousakis. Towards optimal and scalable non-functional service matchmaking techniques. In Carole A. Goble, Peter P. Chen, and Jia Zhang, editors, *Proc. of IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, June 24-29, 2012*, pages 327–335. IEEE Computer Society, 2012.

[KVBF07] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[KWW11] Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral similarity - a proper metric. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Proc. of 9th International Conference on Business Process Management (BPM), Clermont-Ferrand, France, Aug. 30-Sept. 2, 2011*, volume 6896 of *LNCS*, pages 166–181. Springer, 2011.

[Lar06] Rubén Lara. Two-phased Web service discovery. In Prashant Doshi, Richard Goodwin, and Amit Sheth, editors, *Proceedings of AI-Driven Technologies for Services-Oriented Computing Workshop, Boston, MA, USA, July 16, 2006.* AAAI Press, 2006.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[LCC06] Rubén Lara, Miguel Ángel Corella, and Pablo Castells. A flexible model for Web service discovery. In *Proc. of International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives, Seoul, Korea, Sept. 11, 2006*, 2006.

[LCC08] Rubén Lara, Miguel Corella, and Pablo Castells. A flexible model for locating services on the Web. *International Journal of Electronic Commerce*, 12(2):11–40, 2008.

[LdBPF05] Holger Lausen, Jos de Bruijn, Axel Polleres, and Dieter Fensel. WSML - a language framework for semantic Web services. In *Proc. of W3C Workshop on Rule Languages for Interoperability, Washington, DC, USA, Apr. 27-28, 2005.* W3C, 2005.

[Lef07] Dean Leffingwell. *Scaling Software Agility: Best Practices for Large Enterprises.* The Agile Software Development. Addison-Wesley, 2007.

[LH03] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic Web technology. In Gusztáv Hencsey, Bebo White, Yih-Farn Robin Chen, László Kovács, and Steve Lawrence, editors, *Proc. of 12th International World Wide Web Conference (WWW), Budapest, Hungary, May 20-24, 2003*, pages 331–339. ACM, 2003.

[LKS11] Jörg Leukel, Stefan Kirn, and Thomas Schlegel. Supply chain as a service: A cloud perspective on supply chain systems. *IEEE Systems*, 5(1):16–27, 2011.

[LMP07] Vladimir Lifschitz, Leora Morgenstern, and David Plaisted. Knowledge representation and classical logic. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 1, pages 3–88. Elsevier Science, 2007.

[LP06] Heiko Ludwig and Charles J. Petrie. Session summary - “cross cutting concerns”. In Francisco Cubera, Bernd J. Krämer, and Michael P. Papazoglou, editors, *Proc. of Dagstuhl Seminar 05462 on Service Oriented Computing*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 2006.

[LRPF04] Rubén Lara, Dumitru Roman, Axel Polleres, and Dieter Fensel. A conceptual comparison of WSMO and OWL-S. In Liang-Jie Zhang, editor, *Proc. of European Conference on Web Services (ECOWS), Erfurt, Germany, Sept. 27-30, 2004*, volume 3250 of *LNCS*, pages 254–269. Springer, 2004.

[LT10] Günter Ladwig and Thanh Tran. Linked Data query processing strategies. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web, Proc. of 9th International Semantic Web Conference (ISWC), Part I, Shanghai, China, Nov. 7-11, 2010*, volume 6496 of *LNCS*, pages 453–469. Springer, 2010.

[LT11] Günter Ladwig and Thanh Tran. Index structures and top-k join algorithms for native keyword search databases. In Craig Macdonald, Iadh Ounis, and Ian Ruthven, editors, *Proc. of 20th ACM Conference on Information and Knowledge Management (CIKM), Glasgow, United Kingdom, Oct. 24-28, 2011*, pages 1505–1514. ACM, 2011.

[LZLG07] Gexin Li, Wenjie Zhang, Huxiong Li, and Junfang Guo. An efficient way to accelerate service discovery and invocation. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics (SMC), Montréal, QC, Canada, Oct. 7-10, 2007*, pages 1304–1309. IEEE Computer Society, 2007.

[MA10] Carolin Michels and Sudhir Agarwal. Elicitation of preferences for Web service compositions. In Klaus-Peter Fähnrich and Bogdan Franczyk, editors, *Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V.: Service Science - Neue Perspektiven für die Informatik, Band 2, Leipzig, Germany, Sept. 27-Oct. 1, 2010*, volume 176 of *LNI*, pages 103–108. GI, 2010.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic markup for Web services. W3C member submission, W3C, Nov. 2004. http://www.w3.org/Submission/OWL-S/ (accesses Aug. 15, 2013).

[McC02] John McCarthy. Actions and other events in situation calculus. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *Proc. of 8th International Conference on Principles and Knowledge Representation and Reasoning (KR), Toulouse, France, Apr. 22-25, 2002*, pages 615–628. Morgan Kaufmann, 2002.

[Mey91] Bertrand Meyer. *Eiffel: The Language.* Prentice-Hall, 1991.

[Mey92] Bertrand Meyer. Applying “design by contract”. *IEEE Computer*, 25(10):40–51, 1992.

[MGH$^+$98] Drew V. McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus.* Cambridge University Press, 1999.

[MKK$^+$08] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Y. Halevy. Google’s Deep Web crawl. *Proc. of VLDB Endowment (PVLDB)*, 1(2):1241–1252, 2008.

[MM04] Frank Manola and Eric Miller. RDF primer. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/rdf-primer, retrieved 2013-08-15.

[MMWvdA11] Fabrizio M. Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Proc. of 9th International Conference on Business Process Management (BPM), Clermont-Ferrand, France, Aug. 30-Sept. 2, 2011*, volume 6896 of *LNCS*, pages 132–147. Springer, 2011.

[MPM$^+$04] David L. Martin, Massimo Paolucci, Sheila A. McIlraith, Mark H. Burstein, Drew V. McDermott, Deborah L. McGuinness, Bijan Parsia, Terry R. Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia P. Sycara. Bringing semantics to Web services: The OWL-S approach. In Jorge Cardoso and Amit Sheth, editors, *Proc. of 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, CA, USA, July 6, 2004*, volume 3387 of *LNCS*, pages 26–42. Springer, 2004.

[MPPP02] Massimo Mecella, Francesco Parisi-Presicce, and Barbara Pernici. Modeling e-service orchestration through Petri nets. In Alejandro P. Buchmann, Fabio Casati, Ludger Fiege, Meichun Hsu, and Ming-Chien Shan, editors, *Proc. of 3rd International Workshop on Technologies for E-Services (TES), Hong Kong, China, Aug. 23-24, 2002*, volume 2444 of *LNCS*, pages 38–47. Springer, 2002.

[MPW92a] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, 1992.

[MPW92b] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Information and Computation*, 100(1):41–77, 1992.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2008.

[MW11] Francisco Martín-Recuerda Moyano and Dirk Walther. Towards understanding reasoning complexity in practice. In Patrick De Causmaecker, Joris Maervoet, Tommy Messelis, Katja Verbeeck, and Tim Vermeulen, editors, *Proc. of 23rd Benelux Conference on Artificial Intelligence (BNAIC), Ghent, Belgium, Nov. 3-4, 2011*, pages 144–151, 2011.

[Neb88] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[New82] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[NL04] Eric Newcomer and Greg Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley, 2004.

[NM02] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of Web services. In David Lassner, Dave De Roure, and Arun Iyengar, editors, *Proc. of 11th International World Wide Web Conference (WWW), Honolulu, HI, USA, May 7-11, 2002*, pages 77–88. ACM, 2002.

[NVSM07] Meenakshi Nagarajan, Kunal Verma, Amit Sheth, and John A. Miller. Ontology driven data mediation in Web services. *International Journal of Web Services Research*, 4(4):104–126, 2007.

[OAS07] OASIS WSBPEL TC. Web services business process execution language version 2.0. OASIS standard, OASIS, Apr. 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, retrieved 2013-08-15.

[OBS06] Eric Overby, Anandhi S. Bharadwaj, and V. Sambamurthy. Enterprise agility and the enabling role of information technology. *European Journal of Information Systems*, 15(2):120–131, 2006.

[OLPL04] Daniel Olmedilla, Rubén Lara, Axel Polleres, and Holger Lausen. Trust negotiation for semantic Web services. In Jorge Cardoso and Amit Sheth, editors, *Proc. of 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, CA, USA, July 6, 2004*, volume 3387 of *LNCS*, pages 81–95. Springer, 2004.

[O'S06] Justin O'Sullivan. *Towards a Precise Understanding of Service Properties.* PhD thesis, Queensland University of Technology, Faculty of Information Technology, 2006.

[OWL09] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview.* W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-overview, retrieved 2013-08-15.

[Pan09] Jeff Z. Pan. Resource description framework. In Staab and Studer [SS09].

[Pap08] Michael P. Papazoglou. *Web Services - Principles and Technology.* Prentice Hall, 2008.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of Web services capabilities. In Ian Horrocks and James Hendler, editors, *The Semantic Web: Proc. of 1st International Semantic Web Conference (ISWC), Sardinia, Italy, June 9-12, 2002*, volume 2342 of *LNCS*, pages 333–347. Springer, 2002.

[PS03] Massimo Paolucci and Katia P. Sycara. Autonomous semantic Web services. *IEEE Internet Computing*, 7(5):34–41, 2003.

[PTBM05] Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and Annapaola Marconi. Automated synthesis of composite BPEL4WS Web services. In *Proc. of IEEE International Conference on Web Services (ICWS), Orlando, FL, USA, July 11-15, 2005*, pages 293–301. IEEE Computer Society, 2005.

[PvdH07] Michael P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB Journal*, 16(3):389–415, 2007.

[Ran01] Francesco Ranzato. On the completeness of model checking. In David Sands, editor, *Programming Languages and Systems, Proc. of 10th European Symposium on Programming (ESOP), Genova, Italy, Apr. 2-6, 2001*, volume 2028 of *LNCS*, pages 137–154. Springer, 2001.

[RKL$^+$05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[Roy87] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In William E. Riddle, Robert M. Balzer, and Kouichi Kishida, editors, *Proc. of 9th International Conference on Software Engineering, Monterey, CA, USA, Mar. 30-Apr. 2, 1987*, pages 328–339. ACM Press, 1987.

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated Web service composition methods. In Jorge Cardoso and Amit Sheth, editors, *Proc. of 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, CA, USA, July 6, 2004*, volume 3387 of *LNCS*, pages 43–54. Springer, 2004.

[Rud11] Sebastian Rudolph. Foundations of description logics. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F. Patel-Schneider, editors, *Reasoning Web, Tutorial Lectures of 7th International Summer School on Semantic Technologies for the Web of Data, Galway, Ireland, Aug. 23-27, 2011*, volume 6848 of *LNCS*, pages 76–136. Springer, 2011.

[RVNLE09] Marcos Martínez Romero, José Manuel Vázquez-Naya, Javier Pereira Loureiro, and Norberto Ezquerra. Ontology alignment techniques. In Juan R. Rabuñal, Julian Dorado, and Alejandro Pazos, editors, *Encyclopedia of Artificial Intelligence*, pages 1290–1295. IGI Global, 2009.

[SAM12] Majlesi Shahrbanoo, Mehrpour Ali, and Mohsenzadeh Mehran. An approach for agile SOA development using agile principals. *International Journal of Computer Science and Information Technology*, 4(1):237–244, 2012.

[SBF98] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998.

[SBH$^+$05] York Sure, Stephan Bloehdorn, Peter Haase, Jens Hartmann, and Daniel Oberle. The SWRC ontology - semantic Web for research communities. In Carlos Bento, Amílcar Cardoso, and Gaël Dias, editors, *Progress in Artificial Intelligence, Proc. of 12th Portuguese Conference on Artificial Intelligence (EPIA), Covilhã, Portugal, Dec. 5-8, 2005*, volume 3808 of *LNCS*, pages 218–231. Springer, 2005.

[SGA07] Rudi Studer, Stephan Grimm, and Andreas Abecker. *Semantic Web Services.* Springer, 2007.

[SGT$^+$00] Craig Schlenoff, Michael Gruninger, Florence Tissot, John Valois, Joshua Lubell, and Jintae Lee. *The Process Specification Language (PSL): Overview and Version 1.0 Specification.* U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2000.

[SH07] Michael Stollberg and Martin Hepp. Semantic discovery caching: Prototype and use case evaluation. Technical Report DERI-2007-03-27, Digital Enterprise Research Insitute (DERI), Apr. 2007.

[SH11] Sebastian Speiser and Andreas Harth. Integrating Linked Data and services with Linked Data services. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *The Semantic Web: Research and Applications, Proc. of 8th Extended Semantic Web Conference (ESWC), Part I, Heraklion, Crete, Greece, May 29-June 2, 2011*, volume 6643 of *LNCS*, pages 170–184. Springer, 2011.

[SHF11] Michael Stollberg, Jörg Hoffmann, and Dieter Fensel. A caching technique for optimizing automated service discovery. *International Journal of Semantic Computing*, 5(1):1–31, 2011.

[SHH07] Michael Stollberg, Martin Hepp, and Jörg Hoffmann. A caching mechanism for semantic Web service discovery. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, Proc. of 6th International Semantic Web Conference (ISWC) and 2nd Asian Semantic Web Conference (ASWC), Busan, Korea, Nov. 11-15, 2007*, volume 4825 of *LNCS*, pages 480–493. Springer, 2007.

[Sie00] Jon Siegel. *CORBA 3 Fundamentals and Programming.* Wiley Computer Publishing. Wiley Press, 2nd edition, 2000.

[SLB09] Nathalie Steinmetz, Holger Lausen, and Manuel Brunner. Web service search on large scale. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *Proc. of 7th International Joint Conference on Service-Oriented Computing (ICSOC/ServiceWave), Stockholm, Sweden, Nov. 24-27, 2009*, volume 5900 of *LNCS*, pages 437–444, 2009.

[SPAS03] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic Web services. *Journal of Web Semantics*, 1(1):27–46, 2003.

[Spe12] Sebastian Speiser. *Usage Policies for Decentralised Information Processing.* PhD thesis, Karlsruhe Institute of Technology (KIT), 2012.

[SPM06] Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith. Web service composition via generic procedures and customizing user preferences. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web, Proc. of 5th International Semantic Web Conference (ISWC), Athens, GA, USA, Nov. 5-9, 2006*, volume 4273 of *LNCS*, pages 597–611. Springer, 2006.

[SPS04] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding OWL-S to UDDI, implementation and throughput. In Jorge Cardoso and Amit Sheth, editors, *Proc. of 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), San Diego, CA, USA, July 6, 2004*, volume 3387 of *LNCS*. Springer, 2004.

[SPS06] Naveen Srinivasan, Massimo Paolucci, and Katia P. Sycara. Semantic Web service discovery in the OWL-S IDE. In *Proc. of 39th Hawaii International International Conference on Systems Science (HICSS), Kauai, HI, USA, Jan. 4-7, 2006.* IEEE Computer Society, 2006.

[SS97] Ian Sommerville and Peter Sawyer. Viewpoints: Principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3:101–130, 1997.

[SS09] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies.* International Handbooks on Information Systems. Springer, 2nd edition, 2009.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Sti01] Colin Stirling. *Modal and Temporal Properties of Processes.* Springer, New York, NY, USA, 2001.

[SW04] Murray Shanahan and Mark Witkowski. Event calculus planning through satisfiability. *Journal of Logic and Computation*, 14(5):731–745, 2004.

[SWH10] Adish Singla, Ryen White, and Jeff Huang. Studying trailfinding algorithms for enhanced Web search. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *Proc. of 33rd International ACM Conference on Research and Development in Information Retrieval (SIGIR), Geneva, Switzerland, July 19-23, 2010*, pages 443–450. ACM, 2010.

[TAAK04] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *Proc. of ACM Conference on Human Factors in Computing Systems (CHI), Vienna, Austria, Apr. 24-29, 2004*, pages 415–422. ACM, 2004.

[TF06] Ioan Toma and Douglas Foxvog. Non-functional properties in Web services. WSMO Deliverable D28.4v0.1, Digital Enterprise Research Insitute (DERI), Oct. 2006. http://www.wsmo.org/TR/d28/d28.4/v0.1/, retrieved 2013-08-15.

[TGEM07] Raquel Trillo, Jorge Gracia, Mauricio Espinoza, and Eduardo Mena. Discovering the semantics of user keywords. *Journal of Universal Computer Science*, 13(12):1908–1935, 2007.

[Thi98] Michael Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2:179–192, 1998.

[UDD01] UDDI.org. UDDI executive white paper. Technical report, UDDI.org, Nov. 2001. http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf, retrieved 2013-08-15.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[VHA05] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. QoS-based service selection and ranking with trust and reputation management. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *On the Move to Meaningful Internet Systems, Proc. of OTM Confederated International Conferences CoopIS, DOA, and ODBASE, Part I, Agia Napa, Cyprus, Oct. 31-Nov. 4, 2005*, volume 3760 of *LNCS*, pages 466–483. Springer, 2005.

[VKVF08] Tomas Vitvar, Jacek Kopecký, Jana Viskova, and Dieter Fensel. WSMO-Lite annotations for Web services. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, Proc. of 5th European Semantic Web Conference (ESWC), Tenerife, Canary Islands, Spain, June 1-5, 2008*, volume 5021 of *LNCS*, pages 674–689. Springer, 2008.

[W3C13] W3C SPARQL Working Group. SPARQL 1.1 overview. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/sparql11-overview/, retrieved 2013-08-15.

[Wal96] Igor Walukiewicz. A note on the completeness of Kozen's axiomatisation of the propositional μ-calculus. *Bulletin of Symbolic Logic*, 2(3):349–366, 1996.

[WH10] Ryen W. White and Jeff Huang. Assessing the scenic route: measuring the value of search trails in Web logs. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *Proc. of 33rd International ACM Conference on Research and Development in Information Retrieval (SIGIR), Geneva, Switzerland, July 19-23, 2010*, pages 587–594. ACM, 2010.

[WJH11] Andreas Wagner, Martin Junghans, and Andreas Harth. Simulation des dynamischen Abgleichs von Dienstbeschreibungen. Deliverable AP7.3.2, MeRegioMobil, Sept. 2011. (German).

[WJSH11] Andreas Wagner, Martin Junghans, Sebastian Speiser, and Andreas Harth. Privacy-aware semantic service discovery for the smart energy grid. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *The Semantic Web: Research and Applications, Proc. of 8th Extended Semantic Web Conference (ESWC), Part I, Heraklion, Crete, Greece, May 29-June 2, 2011*, volume 6643 of *LNCS*. Springer, 2011.

[Woo75] William A. Woods. What's in a link: Foundations for semantic networks. In Daniel Bobrow and Allan Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press, 1975.

[WP10] Daniel Winkler and Matthias Pressnig. Second prototype for description logic reasoner for WSML DL v2.0. Deliverable D3.2.7, SOA4All, Aug. 2010.

[ZSS94] Shipei Zhang, Oleg Sokolsky, and Scott A. Smolka. On the parallel complexity of model checking in the modal mu-calculus. In *Proc. of 9th Annual Symposium on Logic in Computer Science (LICS), Paris, France, July 4-7, 1994*, pages 154–163. IEEE Computer Society, 1994.

[Zuc10] Maurilio Zuccalà. SOA4All in action: Enabling a Web of billions of services. In Elisabetta Di Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet, Proc. of 3rd European Conference ServiceWave, Ghent, Belgium, Dec. 13-15, 2010*, volume 6481 of *LNCS*, pages 227–228. Springer, 2010.

[ZW95] A. Moormann Zaremski and Jeannette M. Wing. Signature matching: A tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, 1995.

[ZW97] A. Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.