Semantic Caching for Service Oriented Architectures

Michael Stollberg

Digital Enterprise Research Institute Innsbruck (DERI Austria),
Institute for Computer Science, University of Innsbruck,
Technikerstrasse 21a, A-6020 Innsbruck, Austria
michael.stollberg@deri.org

Abstract. This document presents an overview of a proposed PhD thesis on semantic caching for increasing the efficiency of web-based service-oriented architectures. Such systems aim at dynamically detecting and executing Web services for solving client requests. For real world applications, there will be a very large number of Web services available on the Internet. The bottleneck that hampers efficiency and scalability of such architectures is service detection, i.e. finding those Web services out of the available ones that can be used to solve a particular request. The proposed works develops a novel technique that captures service detection results for goals (formalized client requests) and uses this for performing service detection for semantically similar requests. In comparison to other approaches, this caching mechanism allows to achieve the highest possible efficiency under certain constellations between the provided Web services and requests for these. The working hypothesis is that the constellations with a better efficiency comply with the most common situations in real-world e-business applications, which is verified by empirical analysis.

Keywords: Service-oriented Architectures, Semantic Web Services, Goals, Semantic Matchmaking, Efficiency, Scalability, Goal Caching

1 Introduction

Service-oriented architectures (short: SOA) denote the most recent paradigm for IT system design, envisioning dynamic detection and execution of available services for solving client requests. While Web services are commonly understood as the base technology for SOA-based computing over the Web, so-called semantically enabled service-oriented architectures (short: SESA) aim at mechanization of service detection and execution of Web services in order to realize the SOA vision. The central requirements for successful deployment of such technologies are *efficiency* and *scalability*. The former refers to the time that a system needs to solve a given client request, denoting the critical success criterion for technology acceptance by end-users. The latter is concerned with the system's ability to handle the presumably very large number of Web services available on the Internet, which is a pre-requisite for functional stability in real-world applications.

The bottleneck for both efficiency and scalability of in SOA / SESA systems is discovery, i.e. the initial detection of those Web services out of the available ones that are potentially usable for solving a given request. As the first processing step, this needs to take all available Web services into account; all subsequent steps deal with a significantly smaller amount of Web services. The proposed thesis develops and validates a novel technique for overcoming this bottleneck. The approach is to capture service detection results for goals (formalized client requests) and, on this basis, perform service detection for new, semantically similar requests. This technique allows to (1) attain the highest possible efficiency via *discovery-by-lookup*, meaning to perform service detection without invoking a matchmaker, and (2) to increase scalability by *reducing the search space for service detection* on the basis of organizing Web services with respect goals, i.e. the problems that can be solved with them.

The achievable increase for both efficiency and scalability is dependent on the relationship and constellation of provided and requested functionalities. The working hypothesis is that those situations wherein the proposed technique achieves a better efficiency and scalability than all other known approaches correlate with the most common situation in real-world SOA applications. This will be verified by empiric analysis of existing and potential application areas.

Technically, the relevant aspects on previous service detection results is kept in a knowledge base. Discovery and composition components utilize this as an intermediate for accessing Web service repositories. For the technical solution, we focus on requested and provided functionalities that are formally described by preconditions and effects in a state-based model. While a first-order logic framework is used for illustration and demonstration throughout the thesis, the developed technique is independent of particular ontology languages and hence can be applied in several frameworks for Semantic Web services.

This document concentrates on the research methodology while technical details will be presented in other documents and the final thesis. The remainder of this section identifies the research questions and outlines the central line of argument. Section 2 introduces the problem context and discusses the motivation of the undertaken approach in detail. Section 3 provides and overview of the technical solution with respect to its design and central concepts. Section 4 explains the evaluation methodology for verifying the working hypothesis by an empirical applicability study of the developed technique, and Section 5 concludes the paper and discusses success factors of the proposed work.

1.1 Research Questions

The aim of the proposed thesis is to develop and evaluate a novel technology for enhancing the efficiency and scalability of SOA / SESA technology. The main research questions are:

1. is it possible to develop a technique that allow to perform *discovery-by-lookup* for better efficiency and *organization of Web services with respect to solvable goals* for increasing the scalability of service detection mechanisms?

2. does the most common situation in real-world SOA-applications correlate with the constellation of Web services and requests wherein this technique achieves a better efficiency and scalability than all other known approaches?

1.2 Hypotheses

In order to give a concise overview of the planned work, the following outlines the central line of argument in terms of consecutive hypotheses. We explain how each one will be corroborated below.

1. The search space (i.e. the number of available Web services) in real-world SOA / SESA applications will huge.
2. The critical bottleneck for efficiency and scalability of SOA / SESA systems is service detection (determine those Web services out of the available ones that are potentially usable for solving a given request). As the first processing step, this requires a 1:n search on the complete search space.
3. (a) Current service detection mechanisms need to search the complete search space for each request.
   (b) Existing approaches for enhancing the efficiency and scalability group Web services with respect to the provided functionalities; this has deficiencies on the accuracy and applicability in real world SOA applications.
4. Goals are formalized client objectives that allow lifting IT system usage to the knowledge level; concrete requests are specified as instantiations of goals with concrete inputs.
5. There is a significant degree of similarity among requests on the level of goals in typical SOA applications. This is a promising starting point for enhancing the efficiency and scalability of service detection mechanisms.
6. It is possible to develop a (novel) technique that captures completed service detection results and utilizes this knowledge for
   – performing discovery-by-lookup for new requests, and
   – organizing Web services with respect to solvable goals.
7. This technique is able to significantly reduce the computational costs of service detection; by overcoming the bottleneck, this allows to increase the efficiency and scalability of SOA / SESA technology.
8. The approach is superior to all known approaches with respect to the correlation of provided and requested functionalities in typical SOA-applications.

1.3 Methodology

For proper investigation of the research questions and elaboration of the scientific contributions, the thesis consists of three main parts. At first, the **concept & terminology clarification** introduces into the research context and defines

relevant concepts with respect to the state of the art (20% of the overall thesis). Secondly, the **technical solution** presents the developed technique in detail by formal specifications and a prototype implementation for demonstration purpose (50% of the thesis). Finally, the **evaluation** provides an applicability study of the developed technique in real-world SOA-applications for verifying the working hypothesis (30% of the thesis). [1]

To provide an overview of the research methods and tools applied throughout the thesis, the following explains how each of the above hypotheses will be corroborated, along with pointers to the respective sections of this document.

Hypothesis 1 verification methods (Sec.: 2.1, 4.2):
- *literature proof*: SOA vision expects billions of Web services available on the Internet
- *empirical corroboration*: first-hand data of existing Web services in (a) public registries, (b) industrial in-house solutions (e.g. SAP, Verizon), (c) potential SOA-applications (e.g. Ebay)

Hypothesis 2 verification methods (Sec.: 2.2):
- *technical analysis* of existing SOA / SESA system architectures
- *literature proof* for state of the art & related work

Hypothesis 3 verification methods (Sec.: 2.3, Sec.: 2.4):
- for (a): *technical analysis* and *literature proof* of existing service detection mechanisms (main focus on semantic matchmaking of formally described requested and provided functionalities)
- for (b): *requirements analysis* for efficiency and scalability of service detection mechanisms and *technical analysis* of existing approaches

Hypothesis 4 verification methods (Sec.: 3.1):
- *technical solution*: development / formal specification of a goal model for SOA (formal description & goal-driven Web service usage)
- *literature proof*: as extension of existing approaches in accordance to AI research

Hypothesis 5 verification methods (Sec.: 3.1):
- *conceptual analysis* on the structure and properties of typical SOA-applications
- *empirical corroboration*: first-hand data on requests for existing Web services (interviews & statistical data from SAP, Verizon, Ebay, etc.)

Hypothesis 6 verification methods (Sec.: 3):
- *technical solution*: formal specification of technical solution (Sec. 3.2)
- *technical solution*: demonstration of technique by prototype implementation and use case demonstration (Sec. 3.3)

Hypothesis 7 verified by *technical analysis* on the computational cost of the service detection algorithms in developed technique (Sec. 3.2)

Hypothesis 8 verified by empirical applicability study (Sec. 4):
- *statistical methods*: specification of the evaluation methodology (Sec: 4.1)
- *statistical methods*: evaluation of working hypothesis (Sec. 4.2)

[1] A detailed content outline is presented in: Michael Stollberg: "Semantic Goal Caching for Service-Oriented Architectures. PhD Thesis Overview." September 2006; available on request.

2 Problem Statement

This section explains the problem setting in more detail. After outlining the larger research context, we identify service detection as the bottleneck for that hampers efficiency and scalability of service-oriented architectures. We then determine the requirements for overcoming this, and finally discuss possible technical solutions in order to motivate the approach undertaken in this work.

2.1 Research Context

To position our work in the broader research context, the following examines the relevant properties of web-based SOA as well as of goal-driven architectures.

The SOA Vision and Beyond.
The first aspect of relevance is the idea of service orientation and its embedding into the context of IT system design and architectures. Figure 1 illustrates the facets relevant for the following discussion.

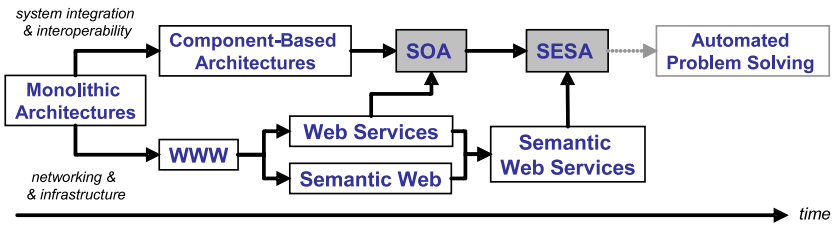


Fig. 1. Context of Service-Oriented Architectures

COMMENT: figure is extendible & very "discussable". However, a initial version fulfils its purpose

In a nutshell, service-oriented architectures (SOA) aim at overcoming the deficiencies in flexibility and cost-efficiency of monolithic system architectures by dynamically detecting and executing loosely coupled software services for solving individual client requests [25]. Web services are commonly considered as the base technology for SOA-based computing over the Internet [3], enabling world-wide access to deployed services with special attention to the integration problem [11]. This technology gains high attention by industry, resulting in several standardization activities (e.g. [45]). In contrast to previous technologies that address the same problems – in particular component architectures on basis of object-orientated programming [13] as the direct predecessor – runtime systems for SOA on the basis of Web services need to deal with newly arising challenges. The ones particulary relevant for our context are:

– To achieve the full potential of SOA with Web services, it is expected that the vast majority of commercial and non-commercial service providers offer their facilities as Web services – similar to organizations' websites as existent today. As a rough estimation of the numeric dimension, consider that each of the currently registered 60 million `.com` domains[2] offers at least one Web service. With respect to this, **efficiency** (time a system needs to solve a given request) and **scalability** (ability to handle a large number of Web services) become crucial success factors for SOA technology.

– The central characteristic of Web services is that they are invoked and consumed via an *interface* that specifies the inputs and outputs to be interchanged between the requester and the provider over the Web. This reduces the need for standardization to interface descriptions (e.g. WSDL [9]), so that Web services are not tied to a specific technology for application development – which is considered to be the central advantage as a new technology for addressing the integration problem [12].

The technical realization of the SOA vision is a massive challenge. Several research efforts are concerned with developing advanced solutions therefore. One of the most significant and promising approaches are *Semantic Web services*: based on ontologies and formal descriptions, inference-based techniques for discovery, composition, execution, and mediation shall enable automated Web service usage, or at least mechanize this to a very high extend [47,27]. The most prominent frameworks therefore are OWL-S [46], SWSF [6], WSDL-S [1], and WSMO [43] that have been submitted to the W3C as proposals for standardization in this field. While the former three are merely concerned with the semantic annotation of Web services, WSMO identifies ontologies, goals, Web services, and mediators as its top level elements and defines formal description models for each [28].

These are understood as the core elements of so-called semantically enabled service-oriented architectures, short: SESA [10]. Expanding the idea of Semantic Web services, the aim is to support all aspects of SOA-technology by semantic techniques. While the efforts around WSMO are mainly centered around Web services as problem solving facilities accessible over the Web, SESA can be understood as a basis towards future technologies for automated problem solving integrate previous from Artificial Intelligence (e.g. [54,2]) with the benefits of web-based SOA. In accordance to UPML [29] as the predecessor of WSMO, it is predicable that the focus for such future developments will move from technical infrastructure towards the problem-solving layer. Therein, the main interest is the usability of formally described provided functionalities for solving formally described requests, independent of technical accessibility.

Goal-Driven SOA.
The central element in SOA / SESA frameworks that of interest in our context are goals, i.e. formalized client requests. The following outlines our the understanding and usage of goals and positions this within related approaches.

[2] daily updated statistics at `http://www.domaintools.com/internet-statistics/`.

Following realizations of intelligent systems for automated, human-like problem solving from different AI-disciplines (see [72] for an exhaustive survey), the concept of goals as the client-side element re-emerges in SOA / SESA frameworks – most obviously, WSMO defines goals as a top-level entity. The basic idea is that a client merely formulates the objective to be achieved while the system automatically detects, combines, and executes appropriate computational facilities for solving this. In the end, this shall lift IT system usage to the knowledge level so that end-users do not have to deal with technical details [49].

Goals denote the client-side element that formally describe client requests and allow automated usage of Web services (or any other problem solving facilities). Therefore, a goal needs to properly specify the client's objective in a declarative manner and carry all information that are required by the system for its automated resolution. As the WSMO goal description model appears to be not elaborated enough, [65] presents a refined model that specifies the formal description and the usage of goals in SESA / WSMO. Figure 2 gives a course-grained overview of the goal model, depicting the relevant aspects for the subsequent discussions.

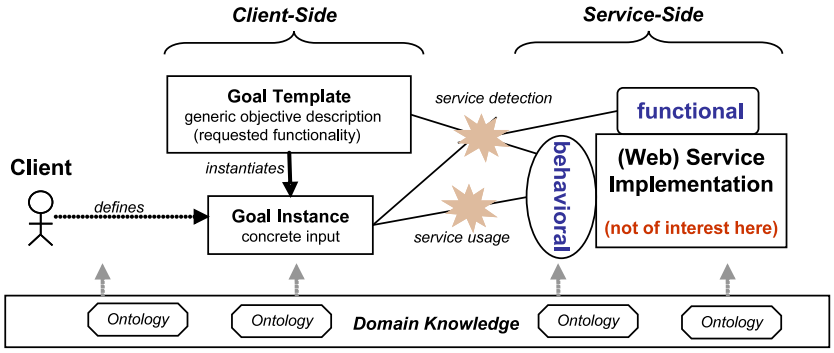


Fig. 2. Overview of Goal-Driven Service-Oriented Architecture

Following the conception of tasks in UPML [29] and their usage in IRS [14] – a broker for Semantic Web services whose initial design is based on UPML – we distinguish **Goal Templates** as the formal specification of a generic objective in terms of a *requested functionality*, and **Goal Instances** that denotes a concrete request by instantiating a goal template with specific inputs [73]. As a simple example, consider the objective of travelling from Innsbruck to Vienna. Here, the Goal Template specifies the origin and destination to be cities located in Austria, and the Goal Instance instantiates them with 'Innsbruck' and 'Vienna' as concrete knowledge items that satisfy the goal description. We explain the formal definition of goals below in more detail.

2.2 Problem Identification: Service Detection as Bottleneck

The following identifies the problem of efficiency and scalability in SOA / SESA technologies in more detail. We determine the requirements for defeating the problem and discuss possibilities for technical solutions in order to motivate the approach undertaken in the proposed research work.

Figure 3 shows an abstraction of the resolution procedure for requests that is supported by SESA realizations – in particular by WSMX [81] and IRS [14] that are based on WSMO, but by also others such as OWL-S IDE [58]. At first, potentially usable Web services are detected out of the available ones with respect to the provided functionalities. This is performed by discovery or by composition in case no directly usable Web service exists. This is followed by optional steps for refining the detection result, such as selecting the most appropriate Web service out of the usable ones [39] or contracting for determining details on the provided functionality [40]. Then, the behavioral compatibility for successful interaction between the request and the discovered or composed Web services is tested [59]. Mediation techniques for handling possibly occurring mismatches can be utilized as auxiliary facilities [62]. Finally, automated execution of the detected or composed Web services results in resolution of the request.[3]

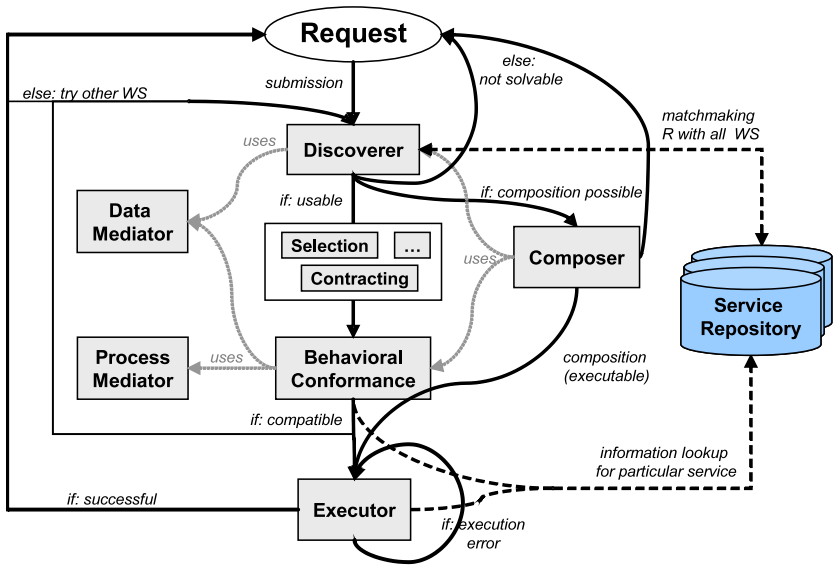


Fig. 3. Request Solving Procedure in SESA Systems

[3] With respect to the above explanations, this procedure considers formally described requested and provided functionalities as the central elements for determining the usability of a Web service for solving a request.

This identifies discovery and composition as the bottleneck for the efficiency and scalability of SOA, and especially SESA technologies. Commonly, discovery is understood as the detection of those Web services out of the available ones that are directly usable for solving the given request while composition is concerned with determining an executable combination of several Web services therefore [64]. As the first processing steps in SESA systems, these components need to take all available Web services into account while the subsequent steps only need to deal with a significantly smaller amount of candidates - in fact those Web services detected by discovery and composition.

In accordance to [36] and with respect to the context of this work, we understand discovery to be concerned with detection of usable Web services with respect to the requested and provided functionalities. Other techniques – e.g. on basis of quality-of-service information [77] – are considered to be allocated in subsequent selection steps. In particular, in this work we are interested in *precisely determining the usability of a Web service for a given request by semantic matchmaking of the formally described requested and provided functionality.* The basis therefore are the five matchmaking degrees that denote the common result of several research efforts: *exact, plugin, subsume* and *intersection* that differentiate situations under which a Web service is usable, and *disjoint* denoting that the Web service is not usable for solving a given request. In the course of this work, we provide a sophisticated and extended definition of the matchmaking degrees in a language-independent framework.[4]

Automated Web service composition is a more complex reasoning task than discovery. We can distinguish two approaches in literature: *functional-level composition* applies AI Planning techniques and works on the provided functionalities (e.g. [48,80]), while *process-level composition* takes the behavioral aspects described in interfaces of Web services into account (e.g. [7,30]); in fact, both need to be integrated in order to attain executable compositions of Web services [75]. An integral part of Web service composition is real-world settings is *candidate detection*, i.e. determining those Web services out of the available ones that serve as the candidates for the composition algorithm. Especially for functional composition this is closely related to discovery – it merely requires slightly different matchmaking criteria. However, most of the existing works do not address this explicitly but assume the composition candidates to be given *a priori*. [18] presents an approach for an interleaved integration of discovery and functional composition, meaning that candidate detection is performed for each step of the composition algorithm. Therein, the intermediate queries created during the composition process are structurally identical to client's request; hence, conventional discovery can be applied for candidate detection.

In conclusion, we observe that the bottleneck that hampers efficiency and scalability of discovery and composition mechanisms – and therewith of SOA / SESA technology in general – is the size of the search space, i.e. the number of

[4] As the most relevant works, the degrees have been presented in [44] for discovery on basis of DL-descriptions, used in [50] for matching in- and outputs of OWL-S profiles, defined for discovery in WSMO in [36] and applied in [68].

available Web services that need to be taken into account. As outlined above, this is expectably very large in real-world settings. We determine the requirements for overcoming this and discuss possible technical solutions in the following.

2.3 Requirement Analysis for Efficiency & Scalability of SOA

In order to determine the requirements for enhancing the scalability of SOA technology with special attention to SESA systems, we examine the technical procedure for solving goals in more detail. In conformance to [51] and as an extension of the discovery procedure defined in [37], Figure 4 shows this procedure that underlies our approach. The central aspect therein is the differentiation of *runtime operations*, i.e. processing steps that need to be performed for solving a concrete request, and operations that are *orthogonal to runtime*, i.e. processing steps that can be performed at any time because they are not directly related to resolving of a particular requests. In conformance to the Web service discovery framework envisioned for WSMO [36], the definition of this procedure is based on the differentiation of goal templates and instances as explained in Section 2.1.

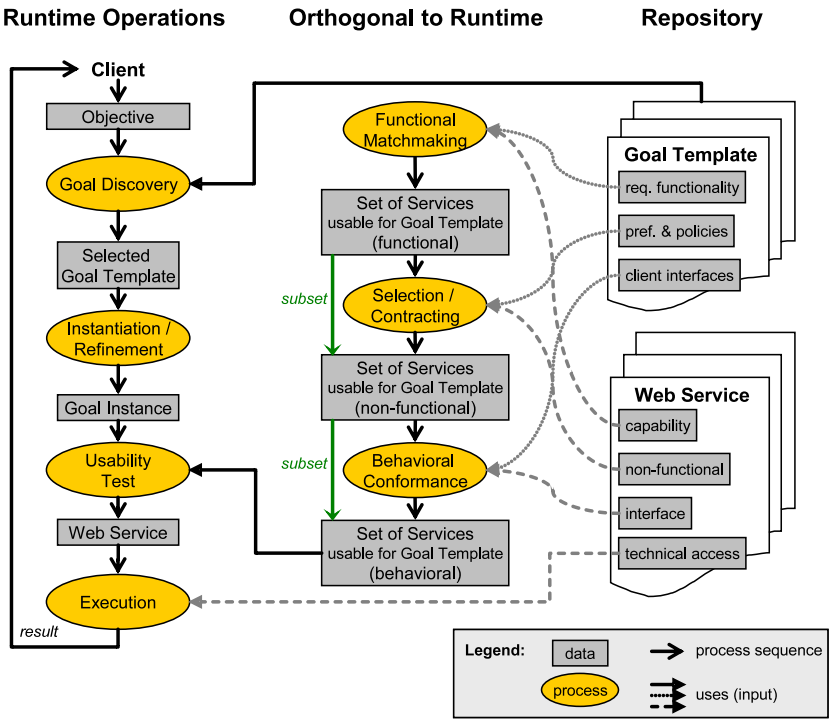


Fig. 4. Goal Resolution Procedure

The runtime branch covers the processing steps for concrete requests. At first, the client searches the repository of existing Goal Templates, i.e. generic objective descriptions. Supported by respective graphical user interfaces (e.g. [14,69]), the client creates a Goal Instance by selecting a goal template and instantiating this with concrete inputs, possibly with refinements of the declarative description. The detection of the particular Web service or the composition to be used for resolving a Goal Instance utilizes the service detection results from the orthogonal-to-runtime branch. Therein, the usability of Web services for Goal Templates is determined with respect to the different description elements of Web services and goals. In accordance to the focus of this work, we consider semantic matchmaking of formal functionality descriptions as the first step that is followed further selection and conformance tests. The set of Web services in the discovery result of semantic matchmaking on formalized requested and provided functionalities is subsequently reduced by the successive detection mechanisms; we refer to [64,65] for details on the description elements and reasoning tasks.

With respect to this, the following discusses the arising requirements for enhancing the efficiency and scalability of service detection mechanisms. Thereby, we consider the following relationship on the frequency of operations that occur in typical SOA applications ($<_f$ denoting the frequency relationship) :

$$\textit{new request} \;<<_f\; \textit{new goal} \;<_f\; \textit{new / updated Web Service}$$

Requirements for Efficiency

In computational theory (e.g. [56]), efficiency is concerned with the *speed* and *space* as desirable properties of algorithms or computer systems apart from functionality and technical design. The first property – speed – refers to the time it takes for an operation to complete, which is commonly described by the *Big-O* notation as a time complexity measurement [42]. The space property refers to the memory or non-volatile storage used up by the algorithm or system, measured in terms of the amount of persistent and working memory required at compile time as well as at runtime.

Naturally, adequate optimization techniques for efficiency are highly dependent on the system design and functionality. In our context, efficiency is mainly related to the runtime branch in Figure 4. As the most critical aspect of efficiency for technology acceptance by end-users, we understand the speed of an SESA system as the *time needed for resolving a request by the usage of Web services.* While the first two processing steps – goal discovery and instantiation – require interaction of the client with the system, the critical operation is the automated determination of the specific Web service (or composition) to be used for solving a Goal Instance. While the computational costs for distinct matchmaking operations can – in theory – be optimized to a negligible extend [23,33], the size of the search space (i.e. the number of available Web services that need to be taken into consideration) denotes the most critical bottleneck for speed efficiency of service detection mechanisms. This also affects the space property of efficiency: the smaller the search space, the fewer working memory will be consumed for detecting the usable Web service or composition.

In conclusion, the central requirement for enhancing the efficiency of SOA / SESA systems is to reduce the search space for the runtime detection of the Web service or composition to be used for solving a concrete client request.

Requirements for Scalability
As another desirable property of algorithms or computer systems, scalability is concerned with the ability to handle large amounts of available resources in a graceful manner [8]. Because of the high dependence of a system's design and its usage environment, commonly accepted measurement and analysis techniques do not exists. However, scalability is considered as a pre-requisite for the applicability of a system: if it can not handle the amount of resources in its designated application area, then it is not considered to be functional for its purpose.

The scalability of SOA / SESA systems requires their ability to handle the very large number of available Web services – which expectably will be several millions (see Section 2.1). In our context, this is mainly related to the *orthogonal-to-runtime* branch in Figure 4. As the first mechanism, the functional matchmaker needs to perform needs perform semantic matchmaking between the requested functionality in a goal description with the functional description of each of the available Web services. While this is only indirectly related to the efficiency for solving a concrete request (in the runtime branch), the critical aspect is the scalability of the used reasoning infrastructure. As analyzed in [78], this is hampered by (1) the general complexity of logical reasoning in comparison to non-logical technologies, and (2) that most reasoner implementations keep all relevant knowledge in the working memory, which limits the number of processable resources tremendously.

The most promising approach for overcoming this is to reduce size of the search space, i.e. the number of Web services that need to be taken into consideration for semantic matchmaking between a requested and provided functionality. Following ordinary techniques for cost-efficient search [19], the most suitable realization is to group available Web services in a way that exhibits the properties of an efficient search graph. As we discuss below in more detail, a graph that organizes Web services with respect to the provided functionalities allows to achieve a higher scalability of service detection mechanisms the better it realizes the following properties. COMMENT: these can / should be better substantiated

1. each **node** represents an accurate generalization of the functionalities grouped in all its sub-nodes and leaves
2. the **arcs** explicate the semantic relationship between nodes such that the distance between nodes is minimal
3. the number of **leaves**, i.e. concrete Web services, is minimal for each node.

In summary, the central requirement for achieving a sophisticated scalability of SOA / SESA architectures is to reduce the search space for matchmaking of requested and provided functionalities. Most adequately, this can be achieved by organizing available Web services in an efficient search graph.

2.4 Solution Possibilities: Approach and Related Work

Completing the problem statement discussion, the following discusses possible technical solutions for enhancing the efficiency and scalability of SOA and SESA technology. On basis of the determined requirements, we investigate related work from the field of Semantic Web services and compare them with the technique proposed in this work.

Without any optimization, the computational costs for service detection are in linear time, i.e. $O(n)$ with n denoting the search space as number of available Web services. As discusses above, n is an expectably very large number in real-world SOA applications. Not reducing the search space for service detection tremendously hampers the efficiency of SOA / SESA technology, and can even functionally disable it for its designated application purpose. Most of the existing works on Web service detection are merely considered with the definition of matchmaking operations for semantic matchmaking but do not consider efficiency and scalability (e.g. [50], [44] [31], [40], [36]). Nevertheless, there are a few approaches that address this problem. We analyze these with respect to their accuracy and appropriateness for goal-driven, semantically enabled service oriented architectures as proclaimed in this work.

Web Service Categorization / Keyword Annotation. Already supported in UDDI [16], available Web services can be organized in categories on the basis of keywords that describe the provided functionality. The keywords used for annotation can be defined on basis of ontologies, which allows to enhance the semantic significance and therewith increase the rate of retrieval accuracy [34]. A recent approach that uses ontology-based keyword categorization for increasing efficiency for search in Web service repositories is the METEOR-S Web Service Discovery Infrastructure (WSDI) [76].

Obviously, this approach allows to reduce the search space for service detection because – theoretically – only those Web services need to be inspected that are allocated in the category relevant for the request to be solved. Besides, it can be argued that this is the simplest to realize because does not require detailed and correct formal descriptions – one of the central problems for broad adaptation of Semantic Web service technologies [71].

However, this approach naturally has significant deficiencies on the accuracy of service detection results. At first, keyword-based categorization by definition misses details on the functionalities provided by Web services that are essential on the usability for a concrete request. Secondly, the examination of publicly accessible UDDI reveals that the categorization schema is pre-defined by the repository administration; providers need to allocate their Web services therein, which often results in the registration of a Web service in unexpected or in multiple categories [5]. These properties result in imprecise service detection results that – in the worst case – may miss usable Web services.

Hence, we consider this technique to not be appropriate for our purposes but rather suitable as a supportive technique for human-driven browsing of Web service or goal repositories.

Repository Indexing. A more decent means for reducing the search space for service detection is to index Web service repositories with respect to the commonalities and differences of the provided functionalities. Organizing a repository index such that it exhibits the properties of an efficient search tree allows – in the best case – to reduce the computational costs of service detection to logarithmic time $O(\log n)$, with is a significant improvement in respect to n being a very large number. The work of Constaninescu et al. (most recently [17]) presents the most sophisticated realization that we examine in more detail.

The functionality provided by Web services is described by a set of description elements: inputs, outputs, preconditions, and effects – similar to functional descriptions in OWL-S and WSMO. The indexing structure is based on Generalized Search Trees, i.e. balanced search trees wherein the inner nodes are defined by a predicate with links to the sub-nodes, and the concrete data (i.e. Web services) are allocated in the leaf-nodes. Constituting the index structure, each inner node is described by two predicates: the lower bound $\underline{\Sigma}$ and the upper bound $\overline{\Sigma}$ for all the sub-nodes $\Sigma_1, \ldots, \Sigma_m$ such that $\forall \Sigma_1, \ldots, \Sigma_m.\ \underline{\Sigma} \sqsubseteq \Sigma \sqsubseteq \overline{\Sigma}$ with $\sqsubseteq$ denoting a complex entailment relation between the description sets of Web services. On the basis of this and with respect to the formal relationship between matchmaking degrees – e.g. $exact \Rightarrow plugin \wedge subsume$ – the inner nodes can be used as pruning conditions for detecting Web services. For instance, if a request Q is satisfied by $\underline{\Sigma}_N$ of an inner node N, then all Web services that are leaf nodes of N are usable for solving Q.

Obviously, this approach is more suitable for organizing Web services than categorization or keyword-based filtering as it builds upon the formally described provided functionalities. Moreover, it satisfies the requirements determined for scalability as the repository index exhibits properties of a sufficient search tree. However, this approach shows deficiencies. Regarding the runtime efficiency, the indexed repository needs to be traversed for each new request – even if the request is similar to a previous one. Also, the usability of Web services under an intersection match needs to be checked for each inner node of the index that is traversed during service detection for a request.

Semantic Caching of Service Detection Results Another solution possibility for enhancing the efficiency and scalability of service detection mechanisms is to adopt the conception of semantic caching. Working on formally described functionalities requested by goals and provided by Web services, the starting point for this technique is the expectably significant rate of similarity of requests in typical SOA applications. The main merits of this approach are:

1. Automated creation of a graph wherein the inner nodes are goals connected by arcs that denote the degree of semantic similarity between the goals, and the leaf nodes are the Web services usable for resolving the goal at the parent node. Similar to repository indexing, this denotes an efficient search graph in order to meet the requirements on scalability for the orthogonal-to-runtime branch in Figure 4. The main difference is that it is created on basis of the usability of Web services and the semantic similarity of goal descriptions.

Besides, each node has direct associated leaf nodes so that detection of usable Web services does not require a complete branch traversal, which provides a better search efficiency.

2. The formal relationship of goal templates and goal instances that allows to perform discovery-by-lookup, i.e. detecting usable Web services for a concrete request without invocation of a matchmaker. Therewith, we can achieve an efficiency of constant time $O(c)$ for the detection of usable Web services for a Goal Instance for the runtime branch in Figure 4, with c being a relatively small number in comparison to n as the number of available Web services.

A central difference to the approaches discussed above is that the achievable increase of efficiency and scalability by caching of service detection results is dependent on the application to a higher extent than techniques that only organize Web services without considering goals and requests. In particular, the attainable quality of the graph for capturing service detection results is dependent on the number of goal descriptions and their semantic similarity: the more goal exist that can be organized in proper similarity groups, the more fine grained is the graph and thus the better its indexing and filtering properties become.

However, we expect the high number of goals and requests with significantly semantic similarity as the common situation in SOA application. As an example for goal similarity, consider two goals from the 'Virtual Travel Agency" use case that is commonly used for illustrating Semantic Web service technologies [70]: G_1 request tickets for travels between Austrian cities, and G_2 requests train tickets for travels between Austrian cities. Here, G_2 is a specialization of G_1 so that the set of usable Web services for G_2 is a subset of those usable for G_1.

Caching is a prominent optimization technique precisely for application areas with many similar requests – e.g. for reduce traffic on the Web by capturing often requested documents or parts thereof [79]. Semantic caching is a specialization that works on the similarity of requests with its most prominent application is the performance optimization of data base querying systems [35]: if a new query Q' that is semantically equal or subsumed by a Q existing in the cache, then the answer set can be derived from the cache. Proportional to the number of semantically similar queries, this allows to attain the best efficiency in comparison to all other known optimization techniques [15]. For more complex reasoning tasks, [4] presents a profitable approach for increasing the efficiency of automated theorem provers by semantic caching of proof results that frequently occur within different proof obligations.

In conclusion, semantic caching of service detection results appears to be a promising approach for sophisticated enhancement of the efficiency and scalability of SOA and especially SESA technologies. Hence, the proposed PhD work develops a novel technique for realizing this and evaluates its applicability for real-world SOA applications. To explicate the motivation with respect to related approaches, Table 1 summarizes the above discussion.

Table 1. Possible Solutions – Advantages and Disadvantages

Approach	Advantages	Disadvantages
Categorization / Keyword Annotation	– moderate size reduction of search space – ease of provision	– inaccurate (many irrelevant Web services in result set) – risk of wrong service detection results
Repository Indexing	– search efficiency achievable in best case: $O(\log n)$ – accurate (no irrelevant Web services in result set) – sound (i.e. correct service detection results)	– runtime efficiency achievable in best case: $O(\log n)$ – no re-use of previous service detection results
Semantic Caching	– runtime efficiency: $O(1)$ (constant time) – re-use of service detection results – accurate & sound service detection	strong dependence of achievable enhancements on the number & semantic similarity of goals and usable Web services in SOA application

3 Technical Solution – Overview

This section explains **Semantic Goal Caching** (short: SGC), which is the realization of the semantic caching technique developed as the main contribution of the thesis. The SGC technique exhibits the properties for enhancing efficiency and scalability of service detection mechanisms as outlined above, and is a language independent technique that can be adopted to several frameworks for SESA and Semantic Web services. Throughout the thesis, we apply first-order logic for specification, illustration, and demonstration purpose. This provides a sufficiently rich descriptions language that covers the Description Logic branch of ontology languages defined for the Semantic Web.

The SGC specification consists of three major parts. At first, Section 3.1 introduces the underlying model for formally describing functionalities requested in goals and provided by Web services. Then, Section 3.2 explains the design and formal specification of the SGC technology. This includes the graph for capturing service detection results for goals and its evolvement, and the service detection algorithms that work upon this. Finally, we outline the technical architecture of the SGC prototype and the demonstration use case scenario in Section 3.3. With respect to the scope of this document, we merely explain the central concepts while conceptual and formal details are presented in the following documents. [5]

[5] Written by or with main contributions from the author: [64] introduces the research field of Semantic Web services with attention towards the SESA vision, and [61]

3.1 Semantic Web Service Discovery Framework

As the framework that underlies the SGC technology, the following outlines the definitions of Web services, goals, and semantic enabled discovery by matchmaking of formalized provided and requested functionalities. Each aspect is defined with respect to the state of the art, along with deepening extensions necessary in our context.

Web Services.
In accordance to the common understanding, we understand Web services to provide a computational facility that can be accessed over the Internet via an interface. Abstracting the commonalities of different Semantic Web services frameworks, Figure 5 shows how initial WSDL-descriptions are extended towards comprehensive semantic annotations of Web services: usage of ontologies as the data model, formal descriptions of the supported communication behavior for consumption and aggregation, further non-functional aspects, and a formal description of the overall provided functionality.

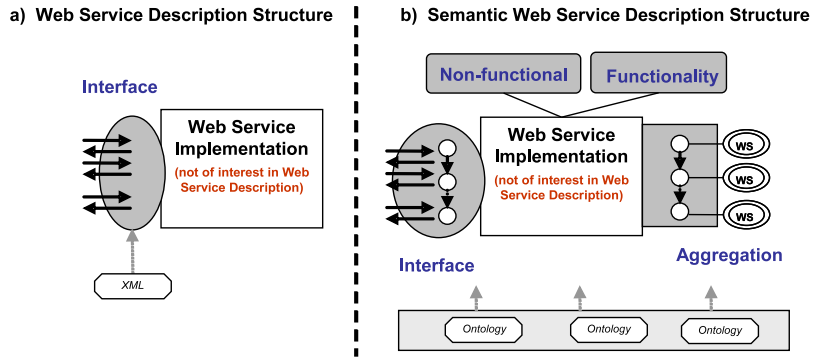


Fig. 5. Web Services and Their Semantic Annotation

The most relevant description element in our context is the *functional description* that formally specifies the overall provided functionality – which refers to WSMO capabilities [53] and is a part of the service profile in OWL-S [46].

presents the initial of the SGC technology. [65] specifies the goal model in detail, with respect to [72] as an exhaustive survey on goal-driven architectures. [38] presents the underlying model used for formally describing requested and provided functionalities. On this basis, [67] presents the formal framework for Web service discovery (submitted to IJEC Special Issue), and [60] formally specifies the SGC technology in detail (under construction). In addition, [66] in detail specifies so-called delta-relations that denote the semantic difference between formal functional descriptions and are used for optimizing the SGC graph.

As one of the essential description elements of a Web service, this needs to describe the conditions that need to hold before the Web service can be executed (preconditions), those that hold after successful execution (postconditions or effects), and the dependence between these. Besides, it should specify all inputs required for invoking and consuming the Web services as well as the outputs provided from execution. While discussing a suitable formal model for functional descriptions below, we identify the following properties of Web services:

1. Web services provide *deterministic functionalities*, meaning that all obtainable outputs and post-execution effects are completely dependent of the inputs provided for invocation and consumption. Without this restriction, a Web service could create arbitrary objects in the world that are not related to a usage request. This would disable automating and composability of Web services, which are prerequisites for service-oriented architectures [28].
2. Functionalities provided by Web services are *static* and *passive*. As the main distinction criterion to intelligent software agents [20], this means that (a) the provided functionality does not dynamically change during runtime, and (b) the proactive usage behavior is driven by the real-world actors (service requester or provider), but not by the Web service itself.

Goals.
As the client side element in SESA frameworks, goals formally describe the objectives that clients want to achieve. Extending the WSMO goal model, we define a goal model that allows to properly specify client objectives and process requests in SESA systems. Figure 6 shows its overall structure as an UML class diagram; the detailed specification is provided in [65].

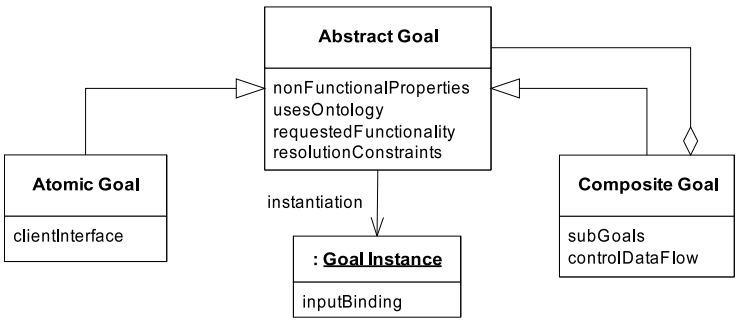


Fig. 6. Goal Model Overview

Abstract Goals formally specify generic client objectives in terms of a *requested functionality* with respect on an ontology; as determined in [72], this is sufficient for describing the type of functionalities offered by Web services in SOA

applications. Atomic Goals extend this with so-called *client interfaces* that allow automated invocation and consumption of Web services [21], and Composite Goals specify additional workflow constraints in terms of a goal decomposition. While the model is not limited to these, Atomic and Composite Goals denote the most relevant goal types for current SOA applications.

Abstract, Atomic, and Composite Goals serve as goal templates, i.e. generic objective descriptions defined as *requested functionalities* in terms of preconditions and effects. For specifying a concrete request, a client instantiates such a goal template with concrete input values.

Semantic Web Service Discovery.
This covers the semantic techniques for determining the usability of Web services for solving goals by determining logical relationships between formally described requested and provided functionalities. Without such technique, the usability of a Web service would have to be detected manually or via test runs. In context of this work, we also apply semantic matchmaking techniques for determining the similarity of goals and for managing the SGC graph (see below in Section 3.2).

The pre-requisites for specifying adequate semantic matchmaking techniques is a model of the world where Web services act in, and *formal functional descriptions* with a clearly defined. As prominent frameworks for Semantic Web services lack of precisely defined semantics for functional descriptions[6], we apply so-called *Abstract State Spaces* (short: ASS) as the underlying formal model. Presented in [38], this defines a state-based model for Web services and the world they act in with rigorous formal definitions. It allows to define semantic matchmaking on the level of executions of Web services with formal functional descriptions, and therewith is sufficiently expressive for our purposes.

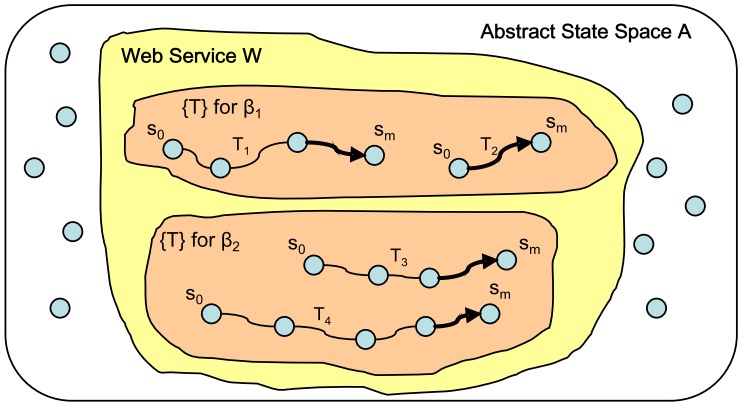


Fig. 7. Web Service, Executions, Input Bindings in the ASS model

[6] In particular, those submitted to the W3C as standardization proposals: OWL-S [46], WSMO [43], SWSF [6], and WSDL-S [1].

The ASS model understands the execution of a Web service W to denote a finite sequence of state transitions $\tau = (s_0, \ldots, s_m)$ in an Abstract State Space. As illustrated in Figure 7, this can be further differentiated into the distinct sets of possible executions of W for each valid input binding $W(\beta)$, so that $\{\tau\}_W = \{\{\tau\}_{W(\beta)}\}$.

A goal denotes the formal specification of a client's desire to change the world from its current state into a state wherein the objective is satisfied. Therefore, the requested functionality defined in a goal template $\mathcal{G}$ denotes a set of sequences of state transitions that can satisfy the described client objective, denoted by $\{\tau\}_{\mathcal{G}}$. Analogously to the invocation of a Web service, a goal instance $GI(\mathcal{G})$ defines an input binding for $\mathcal{G}$ such that the possible solutions for $GI(\mathcal{G})$ are a subset of those for $GI(\mathcal{G})$, i.e. $\{\tau\}_{GI(\mathcal{G})} \subset \{\tau\}_{\mathcal{G}}$. Hence, we define the basic matching condition for a Web service to be usable for solving a goal as:

$$match(\mathcal{G}, W) : \exists \tau.\ \tau \in (\{\tau\}_{\mathcal{G}} \cap \{\tau\}_W)$$
$$match(GI(\mathcal{G}), W) : \exists \tau.\ \tau \in (\{\tau\}_{GI(\mathcal{G})} \cap \{\tau\}_{W(\beta)})$$

It holds that $\forall W.\ match(GI(\mathcal{G}), W) \Rightarrow match(\mathcal{G}, W)$, i.e. a Web service that is usable for a goal instance is also usable for the respective goal template. In consequence, it holds that $\forall W.\ \neg match(\mathcal{G}, W) \Rightarrow \neg match(GI(\mathcal{G}), W)$, meaning that there cannot be any Web service that is usable for a goal instance but not for its template. Therewith, we can utilize knowledge on the usability of Web services on the goal template level as filter for discovery on the goal instance level [68]. This allows to define a 2-step discovery process as proposed in [36]. At first, usable Web services for goal templates $\mathcal{G}$ are determined by semantic matchmaking, and then those usable for $GI(\mathcal{G})$ that instantiates a particular goal template are determined. Thereby, the first-step is allocated in the orthogonal-to-runtime branch and the second one in the runtime branch of Figure 4.

Functional Descriptions. Evaluating these matching conditions by semantic matchmaking requires adequate formal descriptions for the functionality provided by Web services and the requested in goals. The ASS model provides a definition of functional descriptions that is precise and sufficiently rich for our purposes. Referring to [38,67], we desist from an exhaustive discussion but merely summarize the central aspects here. Defined over a signature Σ and with respect to a domain ontology Ω, the central description elements of a functional description $\mathcal{D}$ are:

(i) IF is a set of variables $i_1, \ldots, i_n$ whose scope is $\mathcal{D}$;
this denotes all required input values wherefore an input binding $\beta : (i_1, \ldots, i_n) \rightarrow \mathcal{U}_{\mathcal{A}}$ assigns objects of the universe $\mathcal{U}_{\mathcal{A}}$

(ii) ϕ^{pre} is a statement in $\mathcal{L}$ that constraints the initial state s_0
wherein $i_1, \ldots, i_n$ occur as free variables

(iii) ϕ^{eff} is a statement in $\mathcal{L}$ that constraints the final state s_m
wherein $i_1, \ldots, i_n$ occur as free variables and `out` denotes the output.

The meaning is that $\mathcal{D}$ formally describes a set of $\tau = (s_0, \ldots, s_m)$ such that if s_0, β satisfies ϕ^{pre}, then s_m will be reached such that s_m, β satisfies ϕ^{eff}.

While the ASS model and hence functional descriptions therein are language-independent, we use classical first-order logic (FOL) as the knowledge definition language throughout the thesis [57]. To reduce the complexity of dealing with formal functional descriptions, we define a FOL-structure $sim(\mathcal{D}) = (\Sigma, \Omega, IF, \phi^{\mathcal{D}})$ that simulates the semantics of a functional description. In essence, $sim(\mathcal{D})$ defines the precondition and effect as FOL formulae and represents their relationship as a logical implication. This allows to apply standard notions from model-theoretic semantics like entailment and logical equivalence for specifying logical relationships and operations on functional descriptions. We also omit details and refer to [66] for the exhaustive formal substantiation.

On this basis, we specify semantic techniques for precisely determining the usability of a Web services on the level of goal templates as well as on the level of goal instances. We only outline the general approach while referring to [67] for the detailed specification of both techniques.

Semantic Matchmaking – Goal Template Level. For the first step, we define matchmaking degrees over $\mathcal{D}_{\mathcal{G}}$ as the requested functionality specified in a goal template, and $\mathcal{D}_W$ as the functional description of a Web service. In essence, the matchmaking degrees denote different denote different relationships between $\{\tau\}_{\mathcal{G}}$ as the set of possible solutions for $\mathcal{G}$ and $\{\tau\}_W$ as the set of possible executions of W as follows:

$$\begin{aligned} \mathbf{exact}(\mathcal{D}_{\mathcal{G}}, \mathcal{D}_W) &: \{\tau\}_{\mathcal{G}} = \{\tau\}_W & \mathbf{intersect}(\mathcal{D}_{\mathcal{G}}, \mathcal{D}_W) &: \{\tau\}_{\mathcal{G}} \cap \{\tau\}_W \neq \emptyset \\ \mathbf{plugin}(\mathcal{D}_{\mathcal{G}}, \mathcal{D}_W) &: \{\tau\}_{\mathcal{G}} \subseteq \{\tau\}_W & \mathbf{disjoint}(\mathcal{D}_{\mathcal{G}}, \mathcal{D}_W) &: \{\tau\}_{\mathcal{G}} \cap \{\tau\}_W = \emptyset \\ \mathbf{subsume}(\mathcal{D}_{\mathcal{G}}, \mathcal{D}_W) &: \{\tau\}_{\mathcal{G}} \supseteq \{\tau\}_W \end{aligned}$$

Four degrees – *exact, plugin, subsume, intersect* – differentiate situations wherein the basic matching condition is satisfied, and *disjoint* denotes that this is not given. As this semantic matchmaking technique is also used for determining the similarity of goal templates in SGC, we provide a more general overview in Table 2 in Appendix A.1 of this document. Previous works for semantically enabled Web service discovery define these degrees for less expressive functional descriptions (e.g. [50,44,36]). For candidate detection in Web service composition, slightly different matchmaking definitions are required [18].

Semantic Matchmaking – Goal Instance Level. The usability of a Web service W for solving a goal instance $GI(\mathcal{G})$ is dependent on the concrete inputs defined by the client. It has to hold that the input binding β that is specified for instantiating $\mathcal{G}$ triggers an execution of W that is a solution for $GI(\mathcal{G})$, formally: $\forall \tau_{\beta}.\, \tau \in (\{\tau\}_{\mathcal{G}(\beta)} \wedge \{\tau\}_{W(\beta)})$. This can be evaluated on basis on the given description elements as follows: $match(GI(\mathcal{G}), W)$ is given if $\Omega \models [[\phi^{\mathcal{D}_{\mathcal{G}}}]_{\pi} \wedge \phi^{\mathcal{D}_W}]_{\beta_W}$. Essentially, this determines whether for the input binding defined in a goal instance $GI(\mathcal{G})$ there exists an execution of W that satisfies the final desired state of $GI(\mathcal{G})$. To integrate both levels of discovery, this general condition can be simplified for the distinct matchmaking degrees on the goal template level. We refer to the detailed specification in [67].

3.2 Semantic Goal Caching Specification

The following explains the design and formal specification of the *Semantic Goal Caching* (short: SGC) technique. This is a realization of semantic caching for Web services in order to improve the efficiency and scalability of SOA / SESA technology. With respect to the scope of this document, we here give an overview of the approach and refer to [60] for the detailed specification.

The proposed PhD work concentrates on Web service discovery, i.e. the detection of directly usable Web services. It works on Web services, goal templates and goal instances, and semantic matchmaking on rich functional descriptions as explained above. However, the technique is applicable and easy extensible to other Web service detection scenarios, such as candidate detection for Web service composition or other discovery techniques (e.g. on basis of keyword-based descriptions or for quality-of-service aspects).

This section is structured as follows. At first, we explain the approach and basic model of the SGC technique. Then, we outline the central features for efficient runtime discovery, the graph structure for capturing goal templates and Web service results from them, and the maintenance and evolution of this graph. Finally, we outline the allocation of the SGC in SOA / SESA system architectures. The prototype and use demonstrative use case are explained in Section 3.3.

Principle & Basic Model

The aim of the SGC technique is to increase the efficiency of Web service detection (with main focus on discovery) in order to improve the scalability of technologies for automated Web service usage. To reduce the number of matchmaking operations necessary to regard all potentially usable Web services as the bottleneck hampering scalability, we group goals with respect to the semantic similarity of the requested functionalities and inspect the logical relationship between their discovery results (the subset of all available Web services that can be used to solve a goal). The main merits of this approach are:

1. Web service discovery at runtime can be performed by *discovery-by-lookup*, i.e. in constant time $O(c)$ which allows to achieve a better runtime efficiency than all other known approaches
2. organization of available Web service by grouping them with respect to solvable problems on the goal template level, which allows to increase scalability by *reducing the search space for service detection*
3. the SGC technique is truly *language independent*, i.e. it is neither bound to a particular specification language nor to a particular description model for goals and Web services.

The central aspect of interest for the SGC technique is the logical relationship between the functional similarity of goals and the usability of Web services for these. Let $similar(G_i, G_j, d)$ be a ternary relation that denotes the functional similarity of two goals G_i and G_j, and $usable(G, W, d)$ denote the usability of a Web service or composition for solving a goal; in both relations, d denotes

the matchmaking degree between the functionalities: `exact`, `subsume`, `plugin`, `intersect` as positive, and `disjoint` as the negative degree. The conceptual basis for the SGC technique is that we can infer knowledge about the usability of Web services by inference rules of the form

$$usable(G_j, W, d_3) \leftarrow similar(G_i, G_j, d_1) \wedge usable(G_i, W, d_2).$$

This means that we can infer the matchmaking degree between a goal G_j and a Web service W from knowledge about the matchmaking degree between G_j and another goal G_i and the degree between G_i and W. On the basis of these rules, we can (1) use Web service discovery results on the goal template level as a filter for the goal instance level, and (2) precisely determine the set of usable Web services for semantically similar goal templates. Appendix A.2 provides a concise overview of these inference rules.

Runtime Discovery by Lookup

As the first functional aspect of the SGC technique, this is concerned with Web service discovery for the runtime branch in Figure 4. Because the requested functionality of every goal instance is a specialization of the functionality requested in the corresponding goal template, it holds that $\forall W.\ \neg match(\mathcal{G}, W) \Rightarrow \neg match(GI(\mathcal{G}), W)$ (see above). In consequence, for determining the usable Web services for a goal instance at runtime, we only need to inspect those Web services that are usable for the respective goal template.

We have specified a matchmaking technique that allows to precisely determine the usability of a Web service for a goal instance (see Section 3.1). Under the `exact` and `plugin` degree, this does not require any additional matchmaking so that we achieve computational costs of $O(1)$; for the `subsume` and `intersect` degree, an additional matchmaking step is need so that with overall computational time is $O(n_G)$ with n_G denoting the number of usable Web services for the goal template that is significantly smaller that the number of all available Web services, i.e. $n_G \ll n$.

Figure 8 illustrates the Web service discovery procedure for goal instances. With respect to the expectable frequency of operations in SOA applications (*new request* $\ll_f$ *new goal* $<_f$ *new / updated Web Service*, see Section 2.3), we therewith achieve a very high runtime efficiency.

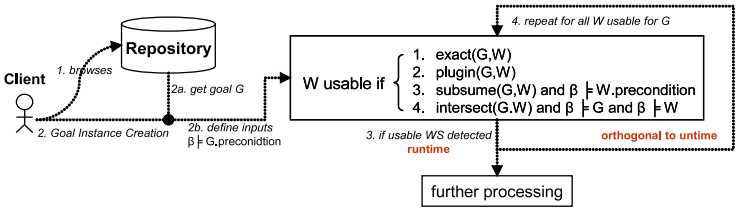


Fig. 8. Procedure of Detecting usable Web Services for a Goal Instance

Goal-driven Web Service Indexing – The SGC Graph
The second aspect is the SGC graph that groups goals on basis of their semantic similarity along with their respective discovery results. This organizes available Web services with respect to problems that they can solve, therewith reduces the search space for service detection in order to achieve a better efficiency and scalability for the orthogonal-to-runtime branch in Figure 4.

As exemplified in Figure 9, the SGC graph is defined as follows:

- every inner node is a goal template G
- every leaf node is a Web service W
- the arcs are mediators that connect nodes and define the matchmaking degree d between them, such that
  - goal (inner) nodes are connected by *GG Mediators* with d = similarity degree of the goal templates
  - each leaf node (Web service W) is connected by *WG Mediator* to a goal template G such that d = usability degree of W for solving G

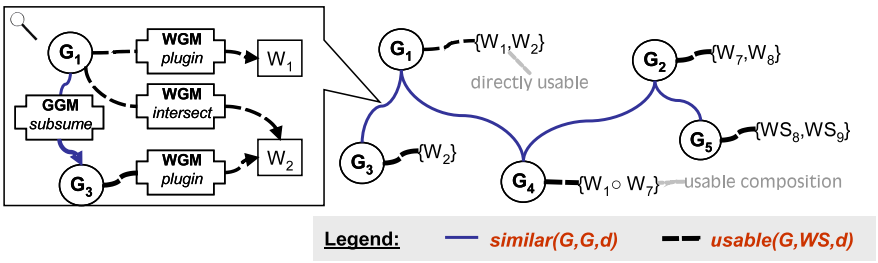


Fig. 9. Example of an SGC Graph

In the figure, G_3 is a goal template that requests a functional specialization of G_1 (e.g., G_1 requests the best restaurant in a city in Austria, and G_3 requests the best restaurant in a city in Tyrol). Hence, the GG Mediator is a directed connector with G_1 as the source and G_3 as the target, and it specifies `subsume` as the similarity degree between them. Because of the usability degree between of the Web service W_1 and W_2 for G_1, we can infer the usability degree for G_3 on basis of the above mentioned inference rules. The right hand side of the figure shows the rest of the SGC graph in a more coarse granularity.

Specifying the arcs between goal templates and Web service as mediators (logical components that connect elements and resolve possible occurring heterogeneities) allows to integrate mediation techniques. With special attention for an integration into WSMO, therewith data and process level mediation techniques can be applied for respective heterogeneities that hamper interoperability of goals and Web services [63].

SGC Graph Management
An important aspect for the operational quality and accuracy of the SGC technique is management and evolution of the SGC graph. There are two main requirements on the SGC graph in order to provide an efficient search tree.

1. Free of Redundance.
The SGC graph should only capture the minimal information that are necessary to be functional. In particular, this refers to redundant arcs between goal templates: GG Mediators that can be inferred from others are obsolete and hence are removed. In the example from Figure 9, imagine a new goal template G_6 such that $subsume(G_1, G_6)$ and $subsume(G_6, G_3)$, i.e. a functional subsumption hierarchy $G_1 < G_6 < G_3$. Here, the connection between G_1 and G_3 becomes obsolete, so that the respective GG Mediator can be removed.

2. Minimal Distance for Goal Insertion.
This is concerned with the addition of new goal templates and their allocation in the existing SGC graph. To maintain the quality of the SGC graph for representing the occurring objectives in an application based on the similarity of goals, it is critical to insert new goals in the appropriate place in the graph. This is given if the distance between the new goal and its neighbor goal nodes is minimal in comparison to all other possible places for allocating the new goal.

To ensure this, the goal insertion applies so-called Δ-relations. In a nutshell, these describe the semantic difference between the requested functionalities requested in goal templates. While we refer to [66] for the detailed definition, we here merely illustrate the approach: defined over $sim(\mathcal{D})$ as the FOL formula that simulates a functional description, a Δ-relation is a pair of formulae $\Delta = (\delta_1, \delta_2)$ with $\delta_1 = \phi^{\mathcal{D}_{G_1}} \wedge \neg\phi^{\mathcal{D}_{G_2}}$ and $\delta_2 = \neg\phi^{\mathcal{D}_{G_1}} \wedge \phi^{\mathcal{D}_{G_2}}$. Figure 10 illustrates this.

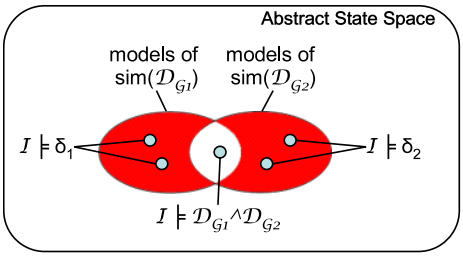


Fig. 10. Illustration of a Δ-relation

When inserting a new goal, we determine the correct branch and position in the SGC graph by determining the minimal Δ-relation between the new goals and existing ones. The addition, removal, or modification of a Web services requires to re-compute the usability degrees for affected goal templates, but this does not affect the critical qualities of the SGC graph. Therewith, it is ensured

that the SGC graph evolves towards an optimal representation of an application with respect to the requested functionalities and the available Web services for solving them.

Both goal insertion and Web service updates are expensive operations. However, (1) we expect them to be less frequent than concrete request formulation in form of goal instances, and (2) they are performed orthogonal to system runtime and hence do not hamper the efficiency of the SGC technique. While redundance-free and optimal goal insertion are the primary aspects for ensuring the appropriateness of the SGC graph, the following aspects are also relevant but will not be elaborated in detail in the proposed work.

Cache Management. This refers to the removal of objects from the cache structure that are not needed anymore. While this is critical in several caching environments that are concerned with working memory, this is not primarily relevant in our context. The reason is that the SGC graph is kept in a knowledge base (not in working memory, see below), and hence its size does not hamper the operational functionality of the SGC technique.

Goal Template Learning. Obviously, the achievable quality of the SGC technique is dependent on the number of similar goal templates. In order to increase this, one can automatically learn new goal templates out of concrete goal instances that have been formulated by clients. However, we consider this as a suitable future extension while the proposed PhD work will be focused on the primary aspects for realization of the SGC technique.

SGC in SOA / SESA Architectures

The final aspect of the SGC technology design is its allocation in SOA, and especially SESA architectures. As shown in Figure 11, the SGC technique (1) serves as an intermediate component through which the system components that need to perform Web service detection access the available Web service repositories, and (2) is used by client for browsing existing goal templates.

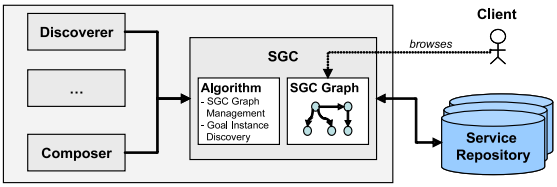


Fig. 11. SGC Allocation in SOA / SESA System Architectures

In consequence, the functional components utilize the SGC technique for Web service detection. The proposed thesis therefore specifies an algorithm for Web service discovery that works with the SGC technique. This algorithm encompasses the following features:

1. SGC Graph Generation (computing of goal similarity and Web service usability degrees)
2. Runtime Discovery (semantic matchmaking for goal instances)
3. Goal Template Insertion and Web Service Update Management

3.3 Prototype & Demonstrative Use Case

In order to demonstrate the functionality of the SGC technique, the PhD work encompasses a prototype implementation and its application in an illustrative example. The the main part of the proposed work will be the formal specification of the SGC technique as outlined above; the purpose of the prototype is merely illustration and exemplification.

The technical platform of the prototype is comprised of FLORA-2 for managing the SGC Graph, and VAMPIRE for semantic matchmaking. This technical framework allows the SGC prototype to become language independent: while FLORA-2 only deals with the SGC graph on basis of molecules that denote goal similarity and Web service usability, it remains independent of the specification language and description structure for requested and provided functionalities.

FLORA-2 is an open source reasoner (available at sourceforge) for Frame-Logic, an expressive logic programming language that allows to deal with frame-based knowledge models [41]. It has been successfully used in related efforts for Web service discovery (e.g. [40]), and is integrated into the WSML reasoning framework (see `http://tools.deri.org/wsml2reasoner/`). The reasons why FLORA-2 has been chosen as the platform for realizing the SGC technique are:

- SGC graph management as a logic programs (highly expressive and adequate for inference rules)
- the internal knowledge base can be used for storing and maintaining the SGC graph
- preserves language independence by separation from semantic matchmaking

VAMPIRE is a resolution-based theorem prover for classical first-order logic with equality [52] that has been dominating respective competitions in the recent years. It has been successfully applied in previous works on semantic Web service discovery [68]. The main benefit of using VAMPIRE is that it allows to model goal and Web service descriptions precisely as specified in our framework (see Section 3.1), and it supports the calculation and reasoning of Δ-relations as shown in [66]. However, the usability of other reasoning environments will be evaluated, especially those developed around WSML.

For illustration and demonstration purpose, the thesis exemplifies the SGC technology and the developed prototype in a real world use case. Although not definitely decided yet, the current planning is to use an ebay marketplace as the use case scenario. The reasons for this choice are as follows:

- a real world use case (not an academic invention)
- a large number of semantically similar requests and offers
- existence of a domain ontology (in form of the ebay product catalogue).

4 Evaluation – Overview

The increase of efficiency and scalability of SOA / SESA technology achievable by the SGC technique is dependent on the correlation of Web services and requests in an application. In particular, the number of goals and their degree of similarity determines the obtainable granularity of the SGC graph. The more similar goals are existing, the more fine grained the SGC graph becomes. In the worst case, there are no semantically similar goals so that the SGC graph cannot be created.

As the working hypothesis, we have claimed that the common situation of typical SOA / SEA application correlates to the constellation of goals and Web services wherein the SGC technique achieves a better increase that all other known approaches (see Hypothesis 8 in Section 1.2). The purpose of the evaluation part of the proposed thesis is to verify this hypothesis by statistical analysis of real-world SOA applications. We therefore examine existing as well as potential SOA applications with main focus on e-commerce, which is commonly considered as one of the most beneficial application areas of advanced Web technology [26,74]. The following outlines the approach for this evaluation.

4.1 Evaluation Methodology

The evaluation methodology needs to identify the aspects and relevant statistical correlations that are critical for the empiric verification of the working hypothesis. It therefore consists of two parts.

1. Computational Cost Evaluation of the SGC Discovery Algorithm.
This in detail analysis the computational costs of the SGC discovery algorithm. As the basis for evaluating the efficiency increase, we therefore identify the aspects and dependency factors relevant for efficiency and scalability of Web service discovery. In particular, this analysis will reveal that

- the runtime efficiency (Web service discovery on the goal instance level) is in $O(n_G)$. Here, n_G denotes the number of usable Web services for a goal template which is dependent on the granularity of the SGC graph
- the obtainable granularity of the SGC graph is dependent on the number of existing goals and Web services and the degree of similarity of goal templates. The optimal efficiency and scalability of the orthogonal-to-runtime branch for Web service discovery is achieved when a functional subsumption hierarchy of goal templates can be created and the most common usability degree for Web services is `exact` or `plugin`.

2. Application Evaluation Scheme.
With respect to the computational costs analysis, we define the following information to be relevant for empirical evaluation of SOA applications.

- the number and functional taxonomy of available Web services
- the number and functional taxonomy of requests for Web services.

On the basis of this scheme, we define the statistical correlations that allow to properly evaluate the increase of efficiency and scalability achievable with the SGC technology.

4.2 Evaluation Data

We concentrate on particular SOA applications from the area of e-commerce as the data basis for the evaluation. Apart from being one of the most promising application areas for advanced Web technology, e-commerce applications commonly need to process a very large amount of requests for a confined set of offered services. Moreover, the runtime efficiency and scalability is of critical importance for the success of e-commerce technology.

Ironically, there do not exists too many SOA-based e-commerce applications. The reason therefore is that the current Web service technology stack around WSDL, SOAP, and UDDI does not allow automated and dynamic usage of Web services – which is aimed to be overcome by the emerging concept of semantically enabled SOA technology (see Section 2.1). Hence, we cannot merely apply existing SOA applications as our data basis for evaluation but we need to examine emerging as well as potential ones. Covering both business-to-business (B2B) as well as business-to-consumer (B2C) applications, we therefore have identified the following examples.

- for B2B, the SOA system established at Verizon for internal software management (statistical data provision has been offered)
- for B2C, the analysis of ebay-marketplaces as a potential SOA application (the provision of statistical data is problematic)

These applications appear to be sufficient for providing a statistically significant evaluation of the working hypothesis. However, it might be changed or extended with other SOA applications from real-world scenarios wherefore the relevant statistical data are available.

4.3 Expected Results

The expected result of the evaluation is the empirical evidence for the working hypothesis stated in the introduction. In particular, we expect a verification that the SGC technique allows to significantly increase the efficiency and scalability of SOA / SESA technology (Hypothesis 7), and that it is superior to all known approaches (as discussed in Section 2.4) with respect to the common situation in typical SOA-applications (Hypothesis 8).

Although the evaluation as outlined here has not yet been carried out, we can observe that the working hypothesis is consistent with the design and purpose of emerging SOA-applications. Regardless whether used within B2B and B2C e-commerce [24,32], e-government [55], or Enterprise Application Integration [22], we observe the following properties of SOA systems: (1) provided Web services provide generic functionalities designed to be (re)-usable for different

tasks, and (2) systems are designed to handle very many goals with a similar semantic structure that differ in the detailed objective specifications. This precisely correlates with the above identified settings wherein the SGC technique can achieve its optimal increase rates for efficiency and scalability.

5 Conclusions, Success Factors, Future Work

This paper has presented an overview of a PhD thesis that develops a semantic caching technique in order to enhance the efficiency and scalability of SOA and SESA technology.

We have shown the importance of the addressed problem with respect to the applicability and operational functionality of SOA / SESA technology, and identified Web service detection as the critical bottleneck. On the basis of a detailed requirements analysis, we have motivated the technical solution; in this paper, we have provided a conceptional overview while the detailed specifications are available in related documents. Finally, we have outlined the approach for evaluating the working hypothesis, stating that the developed technique achieves a better increase of efficiency and scalability in typical SOA applications that all other known approaches. Summarizing, the central research contributions of the proposed work are:

1. the differentiation of goal templates and goal instances that allows a two-phase discovery procedure with a runtime and a orthogonal-to-runtime branch
2. the formal specification of semantically enabled Web service discovery with accurate matchmaking techniques for both branches that work on sufficiently rich descriptions of requested and provided functionalities
3. the formal specification of the Semantic Goal Caching technique (short: SGC) that improves efficiency and scalability for Web service discovery by grouping goal templates with respect to the semantic similarity of the requested functionalities, and enables efficient runtime Web service discovery by capturing discovery results for goal templates
4. an applicability evaluation of the SGC technique that proofs the working hypothesis by empirical evidence, and provides insights on the challenges for future SOA / SESA technology developments.

The developed approaches and technologies are specified in terms of generic formal definitions. Therewith, the presented work is independent of specific specification languages and hence can be applied within several frameworks for SOA and Web service technology. For illustration and demonstration, we apply classical first-order logic through the work. Thus, the proposed PhD work presents a generic solution for an important problem that is critical to the success of SOA and especially of SESA technology. We have shown that the technical realization of a suitable semantic caching technique is possible. Moreover, we have shown that the critical success factor – that is whether the SGC technique allows to achieve an sophisticated efficiency increase in typical SOA applications – can be considered to be minimal and is explicitly addressed in the evaluation part.

Regarding the focus and scope of the proposed work, we concentrate on Web service discovery whereof other service detection mechanisms can be derived. We aim at a broader applicability and hence provide a generic, formal specification of the developed techniques instead of a complete implementation in a particular framework. Therewith, the presented work can be adopted to several SOA technology developments. Finally, we focus on formal descriptions of requested and provided functionalities while neglecting technical details of Web services. This, potential impact of the proposed work ranges beyond the area of Web services but may serve as a generic solution for future technology developments for automated problem solving.

Concluding, this paper as presented a comprehensive overview of the proposed thesis with major attention to the overall approach and line of argument. Detailed technical specifications are available in related documents that will become part of the final thesis.

Acknowledgements

This material is mainly based upon works supported by the EU funding under the DIP project (FP6 - 507483). I would like to thank the following people for fruitful discussions, input, and review of the proposed work: Martin Hepp, Uwe Keller, Michael Genesereth, Stijn Heymans, and Dieter Fensel.

References

1. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S. W3C Member Submission 7 November, 2005. online: http://www.w3.org/Submission/WSDL-S/.
2. J. Allen, T. Austin, and J. Hendler. *Readings in Planning*. Morgan Kaufmann Publishers, 1990.
3. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, Berlin, Heidelberg, 2004.
4. O. L. Astrachan and M. E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *In Proc. of the 11th International Conference on Automated Deduction (CADE-11)*, 1992.
5. D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma. Web Service Discovery - A Reality Check. Technical Report DERI-TR-2006-01-17, DERI, 2006.
6. S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, Martin. D., McIlraith. S., D. McGuinness, J. Su, and S. Tabet. Semantic Web Services Framework (SWSF). W3C Member Submission 9 September, 2005. online: http://www.w3.org/Submission/SWSF/.
7. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of e-Services that Export their Behavior. In *In Proc. of First Int. Conference on Service Oriented Computing (ICSOC)*, 2003.
8. A. B. Bondi. Characteristics of Scalability and their Impact on Performance. In *Proceedings of the 2nd International Workshop on Software and Performance, Ottawa, Ontario, Canada*, pages 195–203, 2000.

9. D. Booth and C. K. Liu. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. Candidate recommendation 6 january 2006, W3C, 2006. online: http://www.w3.org/TR/wsdl20-primer.
10. M. Brodie, C. Bussler, J. de Brujin, T. Fahringer, D. Fensel, M. Hepp, H. Lausen, D. Roman, T. Strang, H. Werthner, and M. Zaremba. Semantically Enabled ServiceOriented Architectures: A Manifesto and a Paradigm Shift in Computer Science. Technical Report TR-2005-12-26, DERI, 2005.
11. M. Brodie and M. R. Stonebraker. *Legacy Information Systems Migration: The Incremental Strategy*. Morgan Kaufmann, San Francisco, 1995.
12. C. Bussler. *B2B Integration: Concepts and Architecture*. Springer, Berlin, Heidelberg, 2003.
13. Szyperski. C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, Boston, 2 edition, 2002.
14. L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, B. Norton, V. Tanasescu, and C. Pedrinaci. IRS-III – A Broker for Semantic Web Services based Applications. In *In Proc. of the 5th International Semantic Web Conference (ISWC 2006), Athens(GA), USA*, 2006.
15. B. Chidlovskii, C. Roncancio, and M.-L. Schneider. Semantic Cache Mechanism for Heterogeneous Web Querying. *Computer Networks*, 31(11-16):1347–1360, 1999.
16. L. Clement, A. Hately, C. von Riegen, and T. (eds) Rogers. UDDI Version 3. "uddi spec technical committee draft", OASIS, 2004. Available from http://uddi.org/pubs/uddi_v3.htm.
17. I. Constantinescu, W. Binder, and B. Faltings. Flexible and Efficient Matchmaking and Ranking in Service Directories. In *Proc. of the 3rd International Conference on Web Services (ICWS 2005), sFlorida, USA*, 2005.
18. I. Constantinescu, B. Faltings, and W. Binder. Large Scale, Type-Compatible Service Composition. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04), San Diego, California, USA*, 2004.
19. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2 edition, 2001.
20. I. Dickinson and M. Wooldridge. Agents are not (just) Web Services: Considering BDI Agents and Web Services. In *Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005), Utrecht, The Netherlands*, 2005.
21. J. Domingue, S. Galizia, and L. Cabral. The Choreography Model for IRS-III. In *Proc. of the 39th Hawaiian International Conference On System Sciences (HICSS-39), Kauai Island, Hawaii*, 2006.
22. A. Duke, J. Davies, M. Richardson, and N. Kings. A Semantic Service Orientated Architecture for the TelecommunicationsIndustry. In *Proc. of the IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM) 2004, Bangkok, Thailand*, pages 236–245, 2004.
23. O. M. Duschka and M. R. Genesereth. Query planning in Infomaster. In *In Proc. of the ACM Symposium on Applied Computing*, 1997.
24. T. Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, 2004.
25. T. Erl. *Service-Oriented Architecture (SOA). Concepts, Technology, and Design*. Prentice Hall PTR, 2005.
26. D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and E-Commerce*. Springer, Berlin, Heidelberg, 2 edition, 2003.
27. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

28. D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domigue. *Enabling Semantic Web Services. The Web Service Modeling Ontology*. Springer, 2006.
29. D. Fensel et al. The Unified Problem Solving Method Development Language UPML. *Knowledge and Information Systems Journal (KAIS)*, 5(1), 2003.
30. C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated Composition of E-services: Lookaheads. In *Proc. of International Conference on Service Oriented Computing (ICSOC 2004), NY*, 2004.
31. S. Grimm, B. Motik, and C. Preist. Variance in e-Business Service Discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004*, 2004.
32. H. He, H. Haas, and D. Orchard. Web Services Architecture Usage Scenarios. W3C Working Group Note 11 February 2004November 2004, 2004. online: http://www.w3.org/TR/ws-arch-scenarios/.
33. I. Horrocks and S. Tobies. Optimisation of Terminological Reasoning. In *Proceedings of the International Workshop in Description Logics 2000 (DL2000)*, 2000.
34. S. Kabel, R. de Hoog, B. J. Wielinga, and A. Anjewierden. The added value of Task and Ontology-based Markup for Information Retrieval. *Journal of the American Society for Information Science and Technology (JASIST)*, 55(4):348–362, 2004.
35. A. M. Keller and J. Basu. A Predicate-based Caching Scheme for Client-Server Database Architectures. *The VLDB Journal*, 5:35–47, 1996.
36. U. Keller, R. Lara, H. Lausen, and D. Fensel. Semantic Web Service Discovery in the WSMO Framework. In J. Cardoses, editor, *Semantic Web: Theory, Tools and Applications*. Idea Publishing Group, 2006. (to appear).
37. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Crete, Greece*, 2005.
38. U. Keller, H. Lausen, and M. Stollberg. On the Semantics of Funtional Descriptions of Web Services. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Montenegro*, 2006.
39. M. Kerrigan. Web Service Selection Mechanisms in the Web Service Execution Environment (WSMX). In *Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC)*, 2006.
40. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004*, 2004.
41. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *JACM*, 42(4):741–843, 1995.
42. D. Knuth. Big Omicron and big Omega and big Theta. *ACM SIGACT News*, 8(2), 1976.
43. H. Lausen, A. Polleres, and D. Roman (eds.). Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June, 2005. online: http://www.w3.org/Submission/WSMO/.
44. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary*, 2003.
45. C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0. Committee Specification, OASIS, 2006.

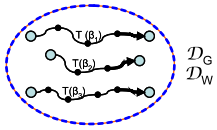
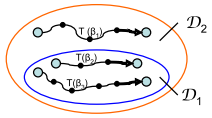
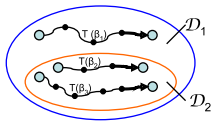
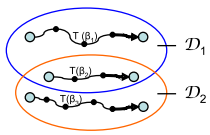
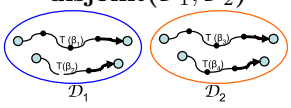
46. D. Martin. OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November, 2004. online: http://www.w3.org/Submission/OWL-S/.
47. S. McIlraith, T. Cao Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.
48. S. McIlraith and T. C. Son. Adapting Golog for Composition of Semantic Web Services. In *roc. of the 8th International Conference on Knowledge Representation and Reasoning (KR '02), Toulouse, France*, 2002.
49. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–122, 1982.
50. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference, Springer*, 2002.
51. C. Preist. A Conceptual Architecture for Semantic Web Services. In *Proc. of the Int. Semantic Web Conf. (ISWC 2004)*, 2004.
52. A. Riazanov and A. Voronkov. The Design and Implementation of VAMPIRE. *AI Communications*, 15(2):91–110, 2002. Special Issue on CASC.
53. D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). Working Draft D2, WSMO Working Group, 2005. final version v1.2, 13 April 2005, online at: http://www.wsmo.org/TR/d2/v1.2/.
54. S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach.* Prentice Hall, 2 edition, 2003.
55. S. K. Sharma and J. N.D. Gupta. Web Services Architecture for M-Government: Issues and Challenges. *International Journal of Electronic Government*, 1(4):462–474, 2004.
56. M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 2 edition, 2005.
57. R. M. Smullyan. *First Order Logic.* Springer, 1968.
58. N. Srinivasan, M. Paolucci, and K. Sycara. CODE: A Development Environment for OWL-S Web services. In *Demonstration at 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan.*, 2004.
59. M. Stollberg. Reasoning Tasks and Mediation on Choreography and Orchestration in WSMO. In *Proceedings of the 2nd International WSMO Implementation Workshop (WIW 2005), Innsbruck, Austria*, 2005.
60. M. Stollberg. Semantic Goal Caching. Technical Report, DERI, 2006.
61. M. Stollberg, E. Cimpian, and D. Fensel. Mediating Capabilities with Delta-Relations. In *Proceedings of the First International Workshop on Mediation in Semantic Web Services, co-located with the Third International Conference on Service Oriented Computing (ICSOC 2005), Amsterdam, the Netherlands*, 2005.
62. M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel. A Semantic Web Mediation Architecture. In *Proceedings of the 1st Canadian Semantic Web Working Symposium (CSWWS 2006), Quebec, Canada*, 2006.
63. M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel. A Semantic Web Mediation Architecture. In M. T. Kon and D. Lemire, editors, *Canadian Semantic Web*, Semantic Web and Beyond: Computing for Human Exprerience. Springer, 2006.
64. M. Stollberg, C. Feier, D. Roman, and D. Fensel. Semantic Web Services - Concepts and Technology. In N. Ide, D. Cristea, and D. Tufis, editors, *Language Technology, Ontologies, and the Semantic Web*. Kluwer Publishers (to appear), 2006.
65. M. Stollberg and M. Hepp. Goal Description Ontology. Deliverable D3.10, DIP, 2006.
66. M. Stollberg and U. Keller. Delta Relations – Semantic Difference of Functional Descriptions. Technical Report DERI-TR-2006-08-18, DERI, 2006.

67. M. Stollberg and U. Keller. Semantic Web Service Discovery with Rich Functional Descriptions. *International Journal of Electronic Commerce (IJEC), Special Issue on: Semantic Matchmaking and Resource Retrieval*, 2007 (to appear). submitted manuscript.
68. M. Stollberg, U. Keller, and D. Fensel. Partner and Service Discovery for Collaboration Establishment on the Semantic Web. In *Proceedings of the Third International Conference on Web Services, Orlando, Florida*, 2005.
69. M. Stollberg, U. Keller, P. Zugmann, and R. Herzog. Semantic Web Fred - Agent Cooperation on the Semantic Web. Demontration at the 3rd International Semantic Web Conference, Hiroshima, Japan, 2004.
70. M. Stollberg and R. Lara. WSMO Use Case "Virtual Travel Agency". Deliverable D3.3, WSMO, 2004.
71. M. Stollberg, M. Moran, L. Cabral, B. Norton, and J. Domingue. Experiences from Semantic Web Service Tutorials. In *Proc. of the Semantic Web Education and Training Workshop (SWET'06) at the First Asian Semantic Web Conference (ASWC 2006), Beijing, China.*, 2006.
72. M. Stollberg and F. Rhomberg. Survey on Goal-driven Architectures. Technical Report DERI-TR-2006-06-04, DERI, 2006.
73. M. Stollberg, D. Roman, I. Toma, U. Keller, R. Herzog, P. Zugmann, and D. Fensel. Semantic Web Fred – Automated Goal Resolution on the Semantic Web. In *Proc. of the 38th Hawaii International Conference on System Science*, 2005.
74. R. Thome, H. Schinzer, and M. Hepp. *Electronic Commerce und Electronic Business. Mehrwert durch Integration und Automation.* Vahlen, Munich, 2005.
75. P. Traverso and M. Pistore. Automatic Composition of Semantic Web Services into Executable Processes. In *Proc. 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan*, 2004.
76. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 6(1):17–39, 2005. Special Issue on Universal Global Integration.
77. L.-H. Vu, M. Hauswirth, and K. Aberer. QoS-Based Service Selection and Ranking with Trust and Reputation Management. In *Proc. of the OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus*, pages 466–483, 2005.
78. H. Wache, L. Serafini, and R. García-Castro. Survey of Scalability Techniques for Reasoning with Ontologies. Deliverable D2.1.1, Knowledge Web, 2004.
79. D. Wessels. *Web Caching.* O'Reilly & Associates Inc, 2001.
80. D. Wu, B. Parsia, Sirin E., J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC 2003), Sanibel Island, Florida*, 2003.
81. M. Zaremba and C. Bussler. Towards Dynamic Execution Semantics in Semantic Web Services. In *Proc. of the Workshop on Web Service Semantics: Towards Dynamic Business Integration, International Conference on the World Wide Web (WWW2005)*, Chiba, Japan, 2005.

A Appendix

A.1 Matchmaking Degrees

Table 2. Definition of Matchmaking Degrees for $\mathcal{D}_1$, $\mathcal{D}_2$

Denotation & Visualization for $\mathcal{D}_1, \mathcal{D}_2$	Definition for: $sim(\mathcal{D}_1) \simeq \mathcal{D}_1$ $sim(\mathcal{D}_2) \simeq \mathcal{D}_2$	Meaning for $\{\tau\}_1$, $\{\tau\}_2$
exact$(\mathcal{D}_1, \mathcal{D}_2)$ 	$sim(\mathcal{D}_1) \equiv sim(\mathcal{D}_2)$, i.e. $\forall\beta.\ \phi^{\mathcal{D}_1} \Leftrightarrow \phi^{\mathcal{D}_2}$.	$\{\tau\}_1 = \{\tau\}_2$
plugin$(\mathcal{D}_1, \mathcal{D}_2)$ 	$sim(\mathcal{D}_1) \models sim(\mathcal{D}_2)$, i.e. $\forall\beta.\ \phi^{\mathcal{D}_1} \Rightarrow \phi^{\mathcal{D}_2}$.	$\{\tau\}_1 \subseteq \{\tau\}_2$
subsume$(\mathcal{D}_1, \mathcal{D}_2)$ 	$sim(\mathcal{D}_2) \models sim(\mathcal{D}_1)$, i.e. $\forall\beta.\ \phi^{\mathcal{D}_1} \Leftarrow \phi^{\mathcal{D}_2}$.	$\{\tau\}_1 \supseteq \{\tau\}_2$
intersect$(\mathcal{D}_1, \mathcal{D}_2)$ 	$\exists\beta.\ \phi^{\mathcal{D}_1} \wedge \phi^{\mathcal{D}_2}$ $\wedge\neg(\forall\beta.\ (\phi^{\mathcal{D}_1} \Rightarrow \phi^{\mathcal{D}_2})$ $\vee\ (\phi^{\mathcal{D}_1} \Leftarrow \phi^{\mathcal{D}_2}))$.	$\{\tau\}_1 \cap \{\tau\}_2 \neq \emptyset$
disjoint$(\mathcal{D}_1, \mathcal{D}_2)$ 	$\neg\exists\beta.\ \phi^{\mathcal{D}_1} \wedge \phi^{\mathcal{D}_2}$.	$\{\tau\}_1 \cap \{\tau\}_2 = \emptyset$

A.2 Inference Rules

Definition 1 (Inference Rules for Web Service Usability). *Let W be a Web service with a functional description $\mathcal{D}_W$, and let G_1 and G_2 be goal templates with functional descriptions $\mathcal{D}_{G_1}$ and $\mathcal{D}_{G_2}$. Let $d(G, W)$ denote the matchmaking degree between a goal template and a Web service, and $d(G_i, G_j)$ denote the similarity degree between two goal templates.*

Given $d(G_1, W)$ and $d(G_1, G_2)$, we can infer knowledge about $d(G_2, W)$ by the following rules.

1. $exact(G_1, G_2)$: $d(G_1, W) = d(G_2, W)$.
2. $plugin(G_1, G_2)$:
   - *(i)* $exact(G_1, W) \Rightarrow subsume(G_2, W)$.
   - *(ii)* $subsume(G_1, W) \Rightarrow subsume(G_2, W)$.
   - *(iii)* $plugin(G_1, W) \Rightarrow subsume(G_2, W)$ *or*
   - *(iv)* $plugin(G_1, W) \Rightarrow plugin(G_2, W)$ *or*
   - *(v)* $plugin(G_1, W) \Rightarrow intersect(G_2, W)$.
   - *(vi)* $intersect(G_1, W) \Rightarrow subsume(G_2, W)$ *or*
   - *(vii)* $intersect(G_1, W) \Rightarrow intersect(G_2, W)$.
   - *(viii)* $disjoint(G_1, W)$*: no statement possible.*
3. $subsume(G_1, G_2)$:
   - *(i)* $exact(G_1, W) \Rightarrow plugin(G_2, W)$.
   - *(ii)* $plugin(G_1, W) \Rightarrow plugin(G_2, W)$.
   - *(iii)* $subsume(G_1, W) \Rightarrow subsume(G_2, W)$ *or*
   - *(iv)* $subsume(G_1, W) \Rightarrow plugin(G_2, W)$ *or*
   - *(v)* $subsume(G_1, W) \Rightarrow intersect(G_2, W)$ *or*
   - *(vi)* $subsume(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(vii)* $intersect(G_1, W) \Rightarrow plugin(G_2, W)$ *or*
   - *(viii)* $intersect(G_1, W) \Rightarrow intersect(G_2, W)$ *or*
   - *(ix)* $intersect(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(x)* $disjoint(G_1, W) \Rightarrow disjoint(G_2, W)$.
4. $intersect(G_1, G_2)$:
   - *(i)* $exact(G_1, W) \Rightarrow intersect(G_2, W)$.
   - *(ii)* $plugin(G_1, W) \Rightarrow plugin(G_2, W)$ *or*
   - *(iii)* $plugin(G_1, W) \Rightarrow intersect(G_2, W)$.
   - *(iv)* $subsume(G_1, W) \Rightarrow subsume(G_2, W)$ *or*
   - *(v)* $subsume(G_1, W) \Rightarrow intersect(G_2, W)$ *or*
   - *(vi)* $subsume(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(vii)* $intersect(G_1, W) \Rightarrow subsume(G_2, W)$ *or*
   - *(viii)* $intersect(G_1, W) \Rightarrow intersect(G_2, W)$ *or*
   - *(ix)* $intersect(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(x)* $disjoint(G_1, W)$*: no statement possible.*
5. $disjoint(G_1, G_2)$:
   - *(i)* $exact(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(ii)* $subsume(G_1, W) \Rightarrow disjoint(G_2, W)$.
   - *(iii)* $d(G_1, W)$ *and* $d \neq subsume$*: no statement possible.*