A Semantic Approach to Approximate Service Retrieval

Eran Toch - Technion - Israel Institute of Technology

and

Avigdor Gal - Technion - Israel Institute of Technology

and

Iris Reinhartz-Berger - Haifa University

and

Dov Dori - Technion - Israel Institute of Technology

Web service discovery is one of the main applications of semantic Web services, which extend standard Web services with semantic annotations. Current discovery solutions were developed in the context of automatic service composition. Thus, the "client" of the discovery procedure is an automated computer program rather than a human, with little tolerance, if any, to inexact results. However, in the real world, services which might be semantically distanced from each other are glued together using manual coding. In this paper, we propose a new retrieval model for semantic Web services, with the objective of simplifying service discovery for human users. The model relies on simple and extensible keyword-based query language, and enables efficient retrieval of approximate results, including approximate service compositions. Since representing all possible compositions and all approximate concept references can result in an exponentially-sized index, we investigate clustering methods to provide a scalable mechanism for service indexing. Results of experiments, designed to evaluate our indexing and query methods, show that satisfactory approximate search is feasible with efficient processing time.

Categories and Subject Descriptors: H.3.5 [**Information Storage and Retrieval**]: web-based services

General Terms: Web service, semantic Web, service retrieval, ontology

---

1. INTRODUCTION

Web services are distributed software components, accessed through the World Wide Web. They are considered first class objects to be reused and combined in order to implement business processes. Semantic description of Web services (known as *semantic Web services*) was proposed in an attempt to resolve the heterogeneity at the level of Web service specifications (including naming of parameters and a description of the service behavior), and to enable automated discovery and composition of Web services. Using languages such as OWL-S [Ankolekar et al. 2001], Web services are extended with an unambiguous description by relating properties such as input and output parameters to common concepts, and by defining the execution characteristics of the service. The concepts are defined in *Web ontologies* [Bechhofer et al. 2004], which serve as the key mechanism to globally define and reference concepts.

A major portion of the research involved in semantic Web services was designed to be used in the context of automatic service composition (see [Medjahed et al. 2003; Cardoso and Sheth 2003]). In automatic composition, the user provides a description of a service requirement, and a composition engine aims at satisfying the requirements by planning a valid composition of services, resulting in an operational application. Most composition engines use logic-based proof inferencing, relying mainly on concept hierarchies as a means for providing approximate matching of services [Paolucci et al. 2002; Klusch et al. 2005].

In real-world settings, the process of service composition may be of an **exploratory** nature rather than one of planning (in the AI sense). In order to generate an executable composition, all the requirements assigned to the composer must be fulfilled. It is often the case that only partial solutions to a composer requirements exist, as Web services are created autonomously without any a-priori knowledge of their intended use. Furthermore, composer requirements may not be well-defined. Rather, they may be driven by the availability of Web services. For example, budgetary constraints may limit the scope of available services and cause the user to compromise and use only affordable Web services. This type of usage requires composition and selection of partial services, which are not well suited for handling by logic-based methods.

In an attempt to support exploratory composition, an engineering approach we

advocate in this work calls for *approximate service retrieval*, followed by "gluing" services together using some additional programming work. Such an approach has the benefit of using existing services for increased reusability, by using existing services, while limiting the number of required services, since rewriting the missing code is always a an option. Naturally, a Web service composer is intuitively interested in finding the most similar composition for her needs, thus reducing the amount of needed code. Therefore, there is a need for ranking search results according to the amount of required modifications for their utilization. Furthermore, as exploratory composition requires several iterative sessions, the response time of the query processor is crucial.

In this paper, we present an efficient method for semantic indexing and approximate retrieval of Web services. The search engine relies on graph-based indexing in which connected services can be approximately composed, while graph distance represents service relevance. Taking advantage of semantic Web services, the query interface translates a user's query into a virtual semantic Web service, which, in turn, is matched against indexed services. Semantic Web services are indexed in a *service-base* that provides fast retrieval of individual services and approximate composition of multiple services. The service-base is constructed using an algorithm [Toch et al. 2005], which classifies service properties and associated ontology concepts according to their relevance to service description. The algorithm provides service ranking that is based on the certainty of matching with a query. The contributions of this research are threefold:

—At the conceptual level we define a new exploratory model for service composition, which is aimed at human users, allowing them to query a service-base using a simple and extensible query language in an interactive way.

—At the semantic level, we extend current service composition approaches to allow *approximate compositions*, compositions that require additional manual effort. An approximate composition is accompanied by a ranking mechanism, based on the estimation of the required manual effort. Manual effort is estimated using the partiality of the composition and its overall semantic distance.

—At the computational level, we present a sub-linear service retrieval algorithm by using a two-level service index of concepts, services and compositions. We use semantic clustering techniques in order to supply a compact representation

of the index.

To demonstrate and evaluate our results, we have developed $\mathcal{OPOSSUM}$ (Object-PrOcess-SemanticS Unified Matching), a search engine for Web services which is based on the methods presented in this paper, including approximation methods based of semantic and procedural similarity. Experimental results, designed to evaluate our indexing and query methods, show that satisfactory approximate search is feasible with efficient processing time.

The article is organized as follows. Section 2 presents a case-study which will be used as a running example throughout the paper. Section 3 describes the data model for service specification. Section 4 describes the syntax and semantics of the query language. Indexing methods for efficient processing of queries are specified in Section 5. Section 6 presents the setup and results of the experimental evaluation of our work. Section 7 describes related work. Finally, Section 8 concludes the research and provides directions for future work.

2. CASE STUDY: HEALTHCARE SERVICE REPOSITORY

As a motivating example for our work, consider the following scenario. An emergency healthcare center uses a sizeable number of computer services, accessed as Web services. Web services describe computer services using some standard, *e.g.*, Web Service Description Language (WSDL) [Christensen et al. 2001], describing the input and output parameters needed for the operation. For example, the service *find nearest medical center* in Figure 1 receives as input *GPS position* and provides as output *medical center*. These services may be part of the internal IT infrastructure, or located externally, in hospitals, health insurance companies, and the public Web. In order to achieve various business tasks, a project or an IT manager requires information regarding available service resources, such as the following:

—**Linking services:** Can a new service be assembled from existing services? For example, can a patient be directed to the available hospital nearest to the patient's address?

—**Gluing services:** If a new service *cannot* be implemented by a simple composition of existing services, then what further coding would be needed and where would it be placed?

—**Mining services:** Which services are needed in a composition to **start** or **end**

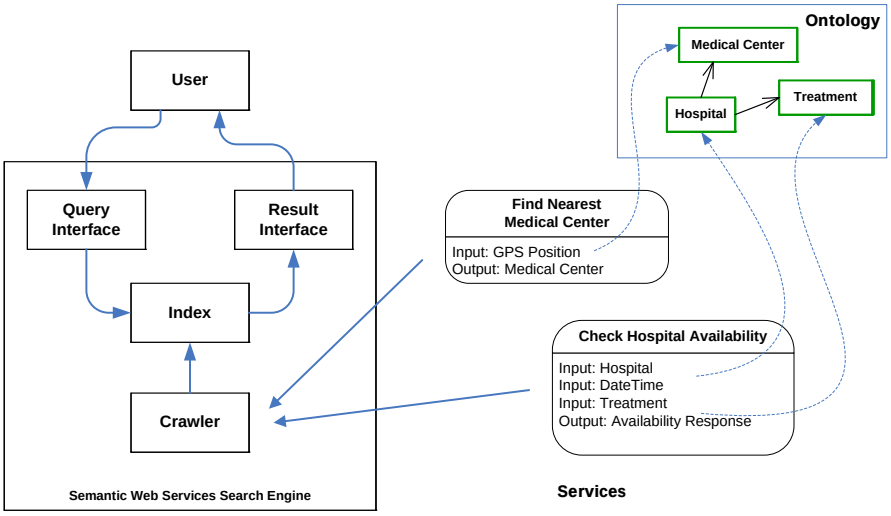

Fig. 1. The Architecture of a Search Engine for Web Services

a given service, *e.g.*, the hospital notification service is needed in a variety of complex hospitalization services.

In this work we assume that Web services are enhanced semantically, using some language of Semantic Web services (*e.g.*, OWL-S [Ankolekar et al. 2001]). Such an enhancement extends standard Web services, with the objective of providing an unambiguous description of their capabilities and properties. The additional information includes mapping of service attributes, such as input and output parameters, to concepts which are defined in a common *ontology*. For instance, in Figure 1, an output parameter of the service operation *find nearest medical center* refers to the *medical center* concept and an input parameter the service *check hospital availability* refers to the *hospital* concept.

To illustrate the architecture of a search engine that can handle these queries, consider Figure 1. The search engine contains four parts: a *Crawler*, an *Index*, a *query interface*, and a *result interface*. A query interface allows the user to set constraints on the desired services. The retrieved services are ranked and presented to the user via the result interface. The crawler discovers, analyzes and indexes semantic descriptions of Web services. The structure of the index allows the questions above to be answered, by indexing services according to the concepts they relate to, and according to their relations with other services.

3. SERVICE NETWORK MODEL

In this section we define a data model for querying Web services. Section 3.1 we provide a narrow definition of semantic Web services, to fit our needs of approximate search. For broader definitions of this concept, the interested reader is referred to [Ankolekar et al. 2001]. The model is based on a directed graph, in which nodes represent operations and edges represent procedural dependencies between operations. Dependencies can be either *inferred*, by analyzing the relations between operation properties, or *empirically derived* from wider-context sources, such as OWL-S specifications. Context classes will be presented in Section 3.2. Section 3.3 defines rules for inferring dependencies, and Section 3.4 describes methods of deriving dependencies from OWL-S models. Finally, Section 3.5 describes our implementation of aligning heterogeneous ontologies.

3.1 Basic Definitions

We define a *semantic operation* to be an atomic component of a Web service, performing an atomic task, which the service description does not further divide. Our notion of a semantic operation is a subset of an OWL-S [Ankolekar et al. 2001] *atomic* or *simple* process. Each operation receives an optional set of input messages and delivers an optional set of output messages. In order to answer queries for service represented in different levels of expressibility, our proposed definition of an *operation* does not include OWL-S's *effects* (logical expressions that define the *results* of operations) and *preconditions*. These are described using outputs and inputs parameters, respectively.

DEFINITION 1 SEMANTIC OPERATION. *A semantic operation is a quadruple* $OP = \langle In, Out, l, \gamma_C \rangle$*, such that*

—*In is a set of input parameters;*

—*Out is a set of output parameters;*

—$l : In \cup Out \cup OP \rightarrow \mathcal{O}$ *is a labeling function that associates each input and output parameter, as well as the operation, with a concept taken from an ontology* ($\mathcal{O}$).

—$\gamma_C : l \rightarrow [0, 1]$ *assigns a value that signifies the certainty of the concept mapping.*

The semantic operation definition is based on an ontology, denoted by $\mathcal{O}$, to which input parameters, output parameters and the operation itself refer.[1] We use the *Web Ontology Language (OWL)* [Bechhofer et al. 2004] as an ontology language of choice, mainly due to its solid theoretical foundations and the wide variety of tools, ontologies and applications associated with OWL. OWL comes in three flavors, each of them represent a level of language expressibility: OWL-Lite, OWL-DL, and OWL-Full. We use the simplest form, *OWL-Lite*, as it contains sufficient constructs for our task (such as class hierarchies), while being relatively simple and easy to use.

A *service network* is a graph of operations, where nodes signify operations and edges represent *relations* between operations. Each relation is associated with a certainty value, allowing relaxed relations to be introduced into the model.

Definition 2 Service Network. *A service network is a connected graph* $SN = \langle OPER, D, c, t \rangle$*, such that*

—$OPER = \{op_1, op_2, \ldots, op_n\}$ *is a finite set of operations.*

—$D : O \times O$ *is a finite set of dependencies - directed relations indicating relations between operations.*

—$c : D \rightarrow \{flow, empirical\}$*, assigning a type category to each of the dependencies.*

—$\gamma_D : D \rightarrow [0, 1]$*, assigning a certainty value to each dependency.*

A service network is depicted in Figure 2. As noted above, the directed arrows represent dependencies between operations. Input and output message parameters (respectively notated by the *in* and *out* labels) are mapped to ontological concepts, described in Figure 3. The directed arrows form dependencies, which represent procedural links between operations, connecting separate operations into service networks. Dependencies can be statically inferred by recognizing similarities between operator parameters (*flow* dependencies), or by learning about relations between operations from external sources (*empirical* dependencies). The first type is discussed in Section 3.3 and the second type in Section 3.4.

[1] In order to simplify the model, we assume that $\mathcal{O}$ is a single ontology, combined of the original ontologies that were used to annotate the semantic Web services. Not all of the concepts in $\mathcal{O}$ are connected to each other. Section 3.5 explains the construction process of $\mathcal{O}$.

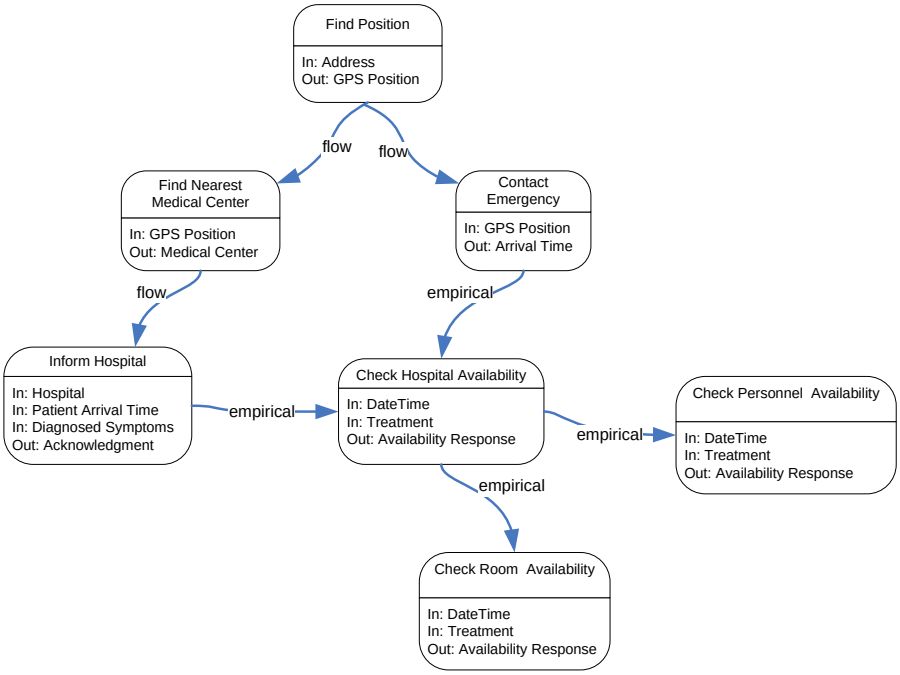

Fig. 2. An Example of Operations and Dependencies

Finally, we denote the union of all service networks by $\mathcal{BASE}$. It serves as a service-base - a repository of services, and has the form of a graph of operations (nodes) and dependencies (edges). Note that service networks are connected graphs, while the service base may **not** be connected.

3.2 Context Classes

In this section we provide a general method for calculating semantic correspondence between concepts, based on the structure of the ontology and the semantic relations between concepts. Our proposed method for analyzing relations between concepts is based on the notion of *context classes*, which form groups of concepts that allow the investigation of relations between them. We next discuss the different concept classes, followed by a definition of semantic correspondence. We shall illustrate the discussion with the use of Figure 3, which depicts a simple healthcare ontology.

For any given concept, we define a set of *context classes*, each of which defines a subset of concepts in $\mathcal{O}$, according to their relation to the concept. Given a query leaf node associated with a concept c, we define a set of concepts, $\mathsf{Exact}(c)$, as c itself

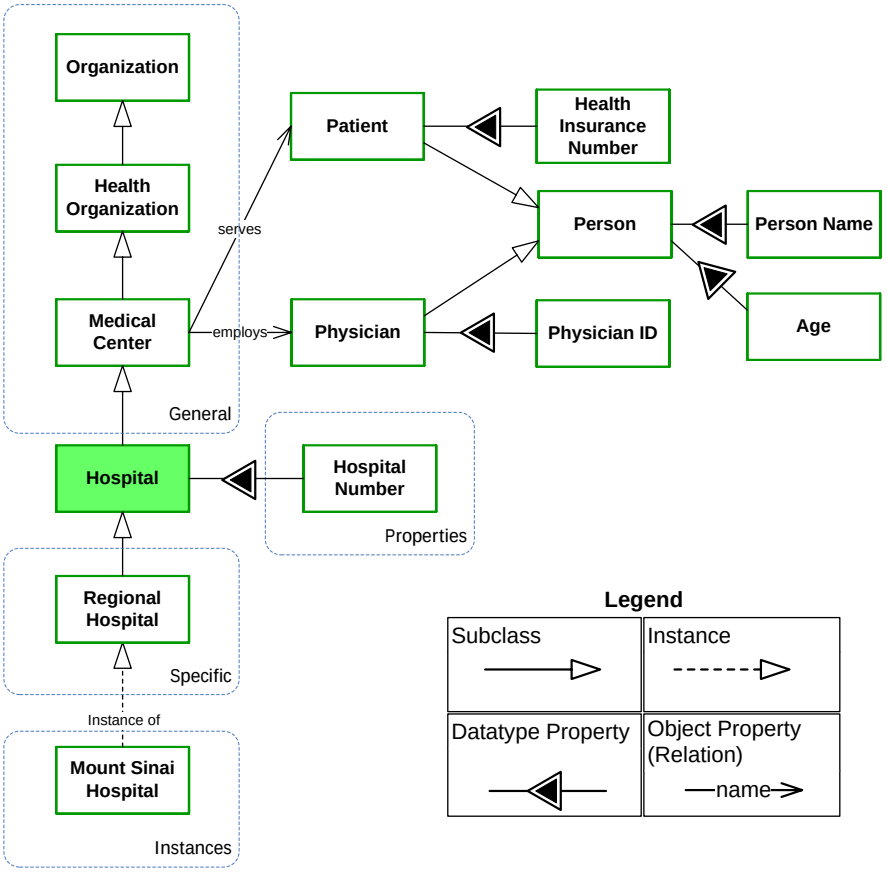

Fig. 3. An example of an ontology for the health-care domain

and concepts which are equivalent to c. The Exact class may contain concepts that have identical semantic meaning. OWL provides relations such as *equivalentClass* to define concept equivalence. As an example, consider the concept *Hospital* in Figure 3. As there are no equivalent classes in this ontology, $\mathsf{Exact}(Hospital) = \{Hospital\}$. The other context classes contain concepts with related meaning. For each concept $c \in \mathcal{O}$, we define the following set of classes:

General. Concepts that supply higher-level context; *i.e.,* belonging to the transitive closure of super classes of c. For instance, the *Medical Center*, *Health Organization*, and *Organization* concepts are super-classes of the *Hospital* concept and therefore fall under the category of General with respect to *Hospital*.

Specific. Concepts that provide a more specific context; *i.e.,* concepts that belong

to the transitive closure of sub-concepts of c. *Regional Hospital* belongs to the Specific class of *Hospital*.

Properties. Concepts which are *Datatype Properties* of c, or properties of General(c). *Object Properties*, which identify general relations in OWL, are not used for identification at this stage. We further distinguish properties into those that can serve for identifying concepts. For instance, the property *PersonName* identifies a person (to some degree) while the property *age* does not identify a *Person*. Three criteria were used in order to distinguish between identifying and non-identifying properties:

(1) **Functional properties:** properties for which each concept is associated with a single property value.
(2) **Inverse-functional properties:** properties for which each property value is associated with a single concept. Naturally, properties which are functional and inverse-functional are considered stronger candidates for identifying properties.
(3) **Property naming heuristics:** we had recognized that some naming conventions for properties can serve for identification purpose, *e.g.*, **physician ID** and **person name**.

InvertProperties. This class holds all concepts of which c is a property, or a property of their Specific concepts. For instance, the concepts *person* and *physician* are in the InvertProperties class of *physician ID*.

Instances. Concepts that are instances of c. For example, the concept *Mount Sinai* is in the Instances class of *regional hospital* and *hospital*. In OWL, elements of finite enumerations, represented by the *oneOf* construct, can also indicate an instance.

Classifiers. Given an instance c, its class holds the concepts which classify c. For example, *regional hospital* is a member of Classifiers(*Mount Sinai*).

Siblings. Concepts that have a mutual parent concept (a general concept or a classifier). For example, *patient* and *physician* are siblings, as *person* is a mutual general concept. We define $\hat{c}$ as the mutual parent, and the set of sibling concepts as:

$$\mathsf{Siblings}(c) = \{c' \in \mathcal{O} \mid \exists\, \hat{c}\,,\ \hat{c} \in \mathsf{General}(c) \wedge \hat{c} \in \mathsf{General}(c')\}$$

Unrelated. This set holds all the concepts that do not belong to any other context

class, *i.e.*, no connection between c and c' was found. Formally:

$$\mathsf{Unrelated}(c) = \{c' \in \mathcal{O} \mid c' \notin \mathsf{Specific}(c) \cup \mathsf{General}(c) \cup \ldots \cup \mathsf{Siblings}(c)\}$$

The semantic correspondence between two concepts is based on the semantic distance between them. We based the definition of the semantic distance function on a study by Bernstein et al. [Bernstein et al. 2005], which evaluated different similarity measures in ontologies. While their findings suggest that no single semantic measure is dominant, they show that a strong relation exists between the type of the semantic measure and the structure of the ontology. Our semantic measure is founded on a combination of two approaches taken from the study by Bernstein et al.: information-theoretic approach and ontology distance approach. The approaches were chosen because they are simple to implement and are compatible with our type of ontologies, which feature clear hierarchy. We augmented the measures by (1) referring to properties of concepts in addition to the concepts themselves, and (2) handling instances.

Given an anchor concept, $c \in \mathcal{O}$, and some arbitrary concept $c' \in \mathcal{O}$, we define the semantic correspondence function $d(c, c')$ to be:

$$d(c,c') = \begin{cases} 1 & , c' \in \mathsf{Exact}(c) \\ \frac{1}{2^{log_\alpha n \cdot log_\beta (1+\delta)}} & , c' \in \mathsf{General}(c) \cup \mathsf{ClassesOf}(c) \cup \mathsf{Properties}(c) \\ \frac{1}{2^{n \cdot log_\beta 1+\delta}} & , c' \in \mathsf{Specific}(c) \cup \mathsf{Instances}(c) \cup \mathsf{InvertProperties}(c) \\ \frac{1}{2^{log_\alpha (n_1+n_2) \cdot log_\beta (1+\delta)}} & , c' \in \mathsf{Siblings}(c) \\ 0 & , c' \in \mathsf{Unrelated}(c) \end{cases}$$

Where n is the length of the shortest path between c and c' and δ is the difference between the average depth of the ontology and the depth of the upper concept. The log bases, α and β, are used as parameters in order to set the magnitude of the descent of the function. In the implementation of $\mathcal{OPOSSUM}$, we had set both parameters to 4. For each type of context class, the function $d(c, c')$ is defined differently:

(1) In the first case, the two concepts are equal or equivalent, and their similarity is set to 1, which represents the highest similarity possible.

(2) In the second case, c' has a broader semantics than c. The distance between the two concepts is calculated according to a descending function, which depends on the distance between the two concepts in the hierarchy. $\log(1 + \delta)$ is

added to reflect the notion that low-level classes are closer to each other than higher-level classes [Bernstein et al. 2005]. For example, the difference between concepts *organization* and *health organization* is more substantial than the difference between the concepts *hospital* and *medical center*. Because of multiple inheritance, a concept may have several "depth" values. In this case, we choose the maximum value. As δ can be equal to 0 (in the likely case that the concept is a leaf node in the ontology), 1 is added to the *log* calculation.

(3) In the third case, c' has a narrower semantics than c. As a result, some properties of the concept c' may not have corresponding properties. For example *medical center* $\in$ Specific*(health organization)*, therefore *medical center* has all the properties of *health organization* but the opposite is not true, *health organization* lacks properties such as *physician*. Therefore, the correspondence function has a higher descending ratio than before.

(4) In the fourth case, c and c' are not directly connected in the hierarchy, but they have a common parent, denoted as $\hat{c}$. In this case, the function is the same as before, but the distance is set to be the sum of n_1 and n_2, reflecting the distance between c and c' to $\hat{c}$.

(5) In the last case, the two concepts are unrelated, so their similarity is set to 0.

3.3 Inferring Dependencies

In order to retrieve compositions that contain operations from different sources and origins, dependencies between operations are to be *inferred*, as prior knowledge of existing relations is incomplete. The following section describes a set of rules for inferring dependencies, based on the notion of context classes.

Two operations q and p are *flow-dependent*, denoted as $q \xrightarrow{f} p$ if the output of q can be used as an input of p. In other words, all inputs of p must be satisfied by some output of q. Formally, $q \xrightarrow{f} p \iff \forall I_p \,, \exists\, O_q : I_p = O_q$. Our definition of flow dependency resembles horizontal dependency, described in [Medjahed and Bouguettaya 2005]. This dependency can be relaxed in several ways:

(1) **Parameter relaxation:** We relax the original definition of flow-dependency by allowing a matching of a subset of the parameters. Rather than requiring all input parameters to be matched with compatible output parameters, only a subset of input parameters is required to be matched. Therefore, $\exists I_p \,, \exists O_q$:

$I_p = O_q$. For example, in Figure 2, the operation *inform hospital* receives three parameters, while flow dependency can be derived on the basis of a single parameter (*i.e., hospital*).

(2) **Concept hierarchy relaxation:** Rather than requiring full compatibility from p's concepts, we relax the dependency by accepting output parameters which are subsumed by the input parameters or vice versa. $\forall I_p \,, \exists O_q \,:\, O_q \in \mathsf{General}(I_p) \vee I_p \in \mathsf{General}(O_q)$. For example, in Figure 2, the operation *find nearest medical center* can flow into *inform hospital* because *hospital* is subsumed by *medical center*.

(3) **Instance relaxation:** This relaxation is similar to the previous one, but now the dependency is satisfied with instances of q, such that $\forall I_p \,, \exists O_q \,:\, O_q \in \mathsf{Instances}(I_p)$. For example, an operation that require a *hospital* concept can flow from an operation that outputs *Mount Sinai hospital*, a specific instance of the Hospital concept.

(4) **Property relaxation:** The original requirement is relaxed by accepting concepts that are properties of q's output parameters. This relaxation is limited to functional and inverse-functional properties only. Formally: $\forall I_p \,, \exists O_q : I_p \in \mathsf{Properties}(O_q)$, where the property is a functional and inverse functional object property relation (such that for each instance of I_p there is a single instance of O_q and vice-versa).

3.4 Empirical Dependencies

Empirical dependencies are used when prior knowledge of relations between operations exist. However, the transformation between external service models (such as OWL-S) to our service base is not straightforward. In this section we define transformation rules, in a semi-formal manner. OWL-S serves as a representative example of Web service specification language. It has been shown that WSMO [Lara et al. 2004] and BPEL4WS [Wohed et al. 2003] have adequate transformations to OWL-S, and therefore the transformation we present is applicable for these languages as well. OWL-S was chosen as the primary language of reference, for the considerable amount of research and tools associated with it.

The transformation starts with the atomic process - the basic component of the OWL-S process model. Each atomic process p, which belongs to an OWL-

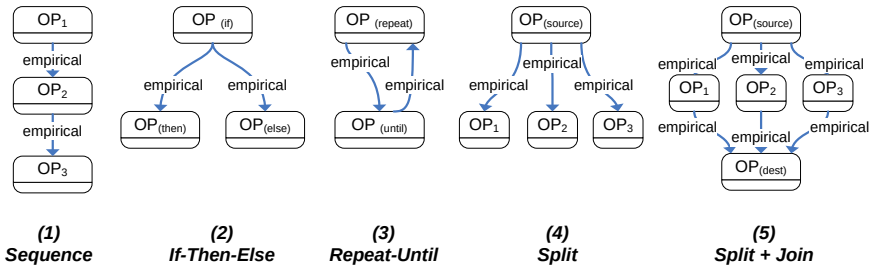

Fig. 4. Transformation Patterns for OWL-S Control Constructs

S model is transformed to an operation $OP \in \mathcal{BASE}$. The input and output properties of the atomic process are mapped to the input and output concepts of OP. Preconditions and effects of OP are abstracted and mapped to OP's input and output concepts respectively. Composite processes are represented as dependencies between operations.

OWL-S supports control constructs, such as conditional execution and parallel execution, in order to coordinate the execution of groups of operations. For instance, in OWL-S, the execution of an atomic process can be dependent on a specific result of another one. However, these constructs are not supported by the service model, as they provide over-specification which is not needed by the search engine. Therefore, complex control constructs are transformed to simple dependencies between operations. In this process, some information is lost. All conditional control constructs, such as *if-then-else* and *repeat-until*, are transformed to empirical dependencies between participating processes, without the actual condition logic. The following list specify transformation patterns for OWL-S control constructs. A visual representation of the patterns is depicted in Figure 4.

(1) **sequence**: The control construct is mapped to a set of empirical dependencies between the operations, ordered according to the original order of the atomic processes.

(2) **if-then-else**: Empirical dependencies are added between the operation that describes the condition (the *if* operation) to the conditioned operations: the *then* and the *else*.

(3) **repeat-until**: An empirical dependency will be added from the conditioned operation (repeat) to the condition operation (until) and vice versa. Note that this construct generate a cycle of dependencies, which is resolved in the

construction of the index.

(4) **split**: For each split construct, a special operation, $OP_{(source)}$ will be added to the service network, representing the beginning of the operation split. An empirical dependency will be added from $OP_{(source)}$ to each of the operations belonging to the split.

(5) **split+join**: Similarly to the transformation pattern for split, an $OP_{(source)}$ operation will be added, as well as a synchronization operation $OP_{(dest)}$. Empirical dependencies will be added to $OP_{(dest)}$ from all operations taking part in the construct (excluding $OP_{(source)}$).

3.5 Aligning Ontologies

The service model is based on the assumption that all concepts belong to a single ontology ($\mathcal{O}$). However, the original ontologies, to which the services are related, originate from different and heterogeneous sources. These ontologies may contain concepts with similar meaning which differ in labeling, content or language. In order to increase the recall of the retrieval process, the original ontologies are merged to construct $\mathcal{O}$. The process of ontology merging takes as input a set of source ontologies and returns a merged ontology, containing a union of the elements of the ontologies, such that equivalent concepts are merged.

The first step in merging ontologies is to map the relations between their concepts. We have adopted the approach of Euzenat and Valtchev [Euzenat and Valtchev 2004], which uses a combination of matching techniques in order to map concepts. These techniques include matching by string-based terminology, lexicon-based terminology, data-type comparison, properties comparison and relation comparison. The weighted contributions of all the techniques are combined to provide the final matching. We have chosen this approach as it designed for OWL-Lite and is fully automatic, making it suitable for processing large amounts of ontological data. After the ontologies were matched, they are merged by combining equivalent concepts, including their properties and relations. Foreign concepts, without any correspondence to other concepts, are copied to the merged ontology.

Ontology merging can also be used to bridge *multilingual* ontologies, which define the same concepts in different languages. As the approach we adopt for ontology matching supports lexicon-based matching, multi-lingual lexicons, such as

EuroWordNet [Vossen 1998], can be used to enable multilingualism in the merged ontology. While $\mathcal{OPOSSUM}$ supports English ontologies in its current version, we plan to augment it with multilingual ontologies in future versions.

4. QUERY INTERFACE, SYNTAX AND SEMANTICS

In this section we describe the query interface of $\mathcal{OPOSSUM}$ and the underlying query language. Section 4.1 describes the user interaction involved in query composition. Section 4.2 describes the syntax underlying the query language. Section 4.3 defines the semantics of basic query operators, while Section 4.4 defines the semantics of complex query expressions. Finally, Section 4.5 describes an extension mechanism for the query language.

4.1 Query Interface Overview

Users communicate with the $\mathcal{OPOSSUM}$ search engine either through the *basic* query interface or through the *advanced* query interface. The two interfaces share the same query language, but employ different levels of formality and expressiveness. Figure 5 displays a screenshot of the two interfaces. The basic query interface requires no more than a set of keywords. The user does not need to specify operators, which are added automatically using predefined settings. In contrast, the advanced query interface allows the user to specify exactly the types of service properties to query. It also allows the user to use logical operators in order to get more general or more specific results.

Both query interfaces generate a **declarative** specification of a virtual semantic Web service. It is declarative in the sense that: (1) it does not enforce an implementation on the query results, and (2) a query can be matched by compositions of operations, rather than by a single operation. The query is relaxed in that it contains keywords instead of formal concepts, and it may contain disjunctions and/or approximate conjunctions.

User queries are transformed into formal queries (described in Section 4.2) by automatically mapping keywords to concepts. In order to formalize a simple query, "*address hospital*", into a query expression, each of the keywords is mapped to a concept term by using content matching techniques.[2] If more than one concept is

[2]The content matching techniques are beyond the scope of this article. The interested reader is referred to [Gal et al. 2005] for an elaborated discussion.

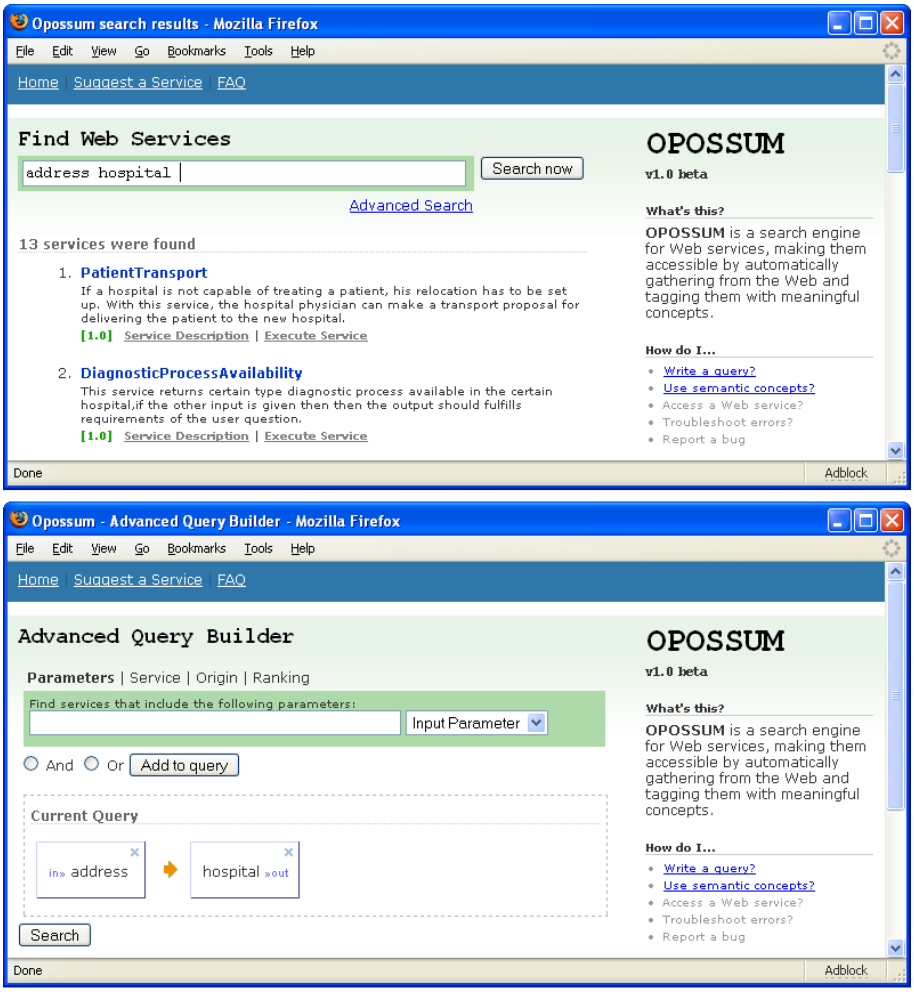

Fig. 5. Screenshots of $\mathcal{OPOSSUM}$'s simple (top) and advanced (bottom) Query Interface

matched with the keyword, the keyword with the highest matching score is used in the query evaluation. The user is also alerted, and can choose between the proposed alternative concept terms. Queries entered via the *simple* interface undergo two additional processing steps:

(1) **Adding connectors:** Query terms are connected automatically using a conjunctive connector. Applying this stage on an example query "*address hospital*" yields "$address \wedge hospital$".

(2) **Annotating terms:** Concept terms are annotated with a property category, defining which property will be matched with the term. Available categories are *input, output* and *operation*. These categories are aligned with the term labeling function (l), as defined in Section 3.1. The search engine chooses an initial default category and lets the user change these defaults after the query was evaluated.

In order to annotate the terms, a prefetching process is carried out. The index is queried regarding the existence of concepts which are either *input, output* or *operation*. Afterward, the query is expanded with the disjunction of each category with which at least one concept is related. For example, applying the annotation on "*address* $\wedge$ *hospital*" results in "*input:address* $\wedge$ *output:hospital*".

4.2 Abstract Query Syntax

The abstract query presented in this section forms a mathematical abstraction of queries presented in the previous section.

Definition 3 Abstract Query. *An abstract query is a quadruple* $Q = \langle T, l, pc, o \rangle$, *where*

—T *is a rooted, directed, binary tree;*

—$l : N \rightarrow \mathcal{O}$ *is a* ***term labeling*** *function that associates each* ***leaf*** *node in* T *with a concept* $c \in \mathcal{O}$.

—$pc : N \rightarrow \{in, out, op\}$ *is a* ***property category*** *function that associates each* ***leaf*** *node in* T *with a categorization of its property, which can be input, output, or operation.*

—$o : N \rightarrow \{\wedge, \vee\}$ *is an operator function that associates each* ***non-leaf*** *node with conjunction or disjunction connector, respectively.*

Each leaf node n_{Q_l} is annotated with a property category (*i.e.*, input, output, operation) and a concept. A non-leaf node n_Q is the application of a boolean connector over its children. If $o(n_Q) = \wedge$, then n_Q is an *and-node*, and if $o(n_Q) = \vee$, then n_Q is an *or-node*. Figure 6 depicts an abstract query, requesting services that have an input parameter aligning either with an *address* concept or with a *GPS position* concept, and should also have a *hospital* output parameter.

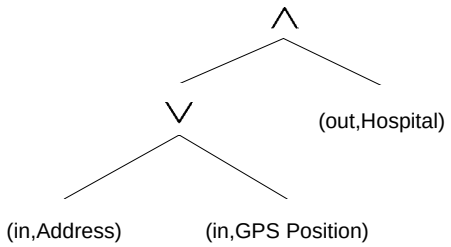

Fig. 6. An abstract query

4.3 Query Matching Semantics

In this section we provide an overview of the query semantics. Adopting a bottom-up approach, the next two sections start with a detailed element description and continue with a complete structure of the query.

The result of the query evaluation is a set of *virtual services*. A virtual service, V, is a sequence of one or more operations, ranked according to their order of execution, such that $V = \langle op_1, op_2, \ldots op_n \rangle$. Note that a virtual service may contain operations that originate from diverse sources. Each virtual service is associated with a *matching certainty*, expressing the certainty in which the virtual service answers the query. The notion of matching certainty is embodied by the μ-satisfiability relation. Let V be a virtual service, and let Q be a query. The μ-satisfiability relation, denoted as $V \models_\mu Q$, indicates that V satisfies the requirements of Q with a certainty of μ.

We define the levels of matching in a recursive manner. The basic unit of matching is related to a single operation, which is matched with a query **leaf** node (n_{Q_l}). In this case, the matching certainty is determined according to the semantic correspondence between the node's concept and the operation's concepts. The matching certainty of virtual services is computed based on the certainty of each of the operations and the certainty of the relation(s) between them.

In order to formally define the μ-satisfiability of an operation, we first define semantic correspondence. The function $\mu : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$ defines the semantic correspondence that maps query leaf node concept (c) and operation parameter

concept (c') to a value between 0 and 1, where 0 implies no compatibility and 1 implies full compatibility. We can now define the operation satisfiability of a query leaf node as follows.

DEFINITION 4 OPERATION SATISFIABILITY. *An operation OP satisfies n_{Q_l} if the following jointly applies:*

(*1*) *OP contains a parameter p with the same property category of the leaf query node, $pc(n_{Q_l})$.*

(*2*) *$\mu(l(n_{Q_l}), l(p)) > \hat{\mu}$ - the semantic correspondence between the two concepts is higher than a threshold $\hat{\mu}$.*

The method for calculating μ, the semantic correspondence function, is identical to the context-classes based method described in Section 3.2.

4.4 Complex Queries Semantics

In order to define the semantics of complex queries, the notion of μ-satisfiability is broadened from operation matching to the matching of complete queries, including conjunctive and disjunctive operators. We say that $V \models_\mu Q$, when a query can be satisfied by a virtual service, in a given μ level of certainty.

In order to define the semantics of disjunction, the query is transformed into a disjunctive normal form. For instance, the example query (depicted in Figure 6), which has the original form of $((in, Address) \vee (in, GPSPosition)) \wedge (out, Hospital))$ will be transformed into the following form:

$$((in, Address) \wedge (out, Hospital))$$
$$\vee$$
$$((in, GPSPosition) \wedge (out, Hospital))$$

A virtual service satisfies an *or-node* if it satisfies one of its child nodes. Let n_{Q_1} and n_{Q_2} be the child nodes of the *or-node*, n_Q. The μ-satisfiability specification of *or-node* is defined as follows:

DEFINITION 5 DISJUNCTION MATCHING. *$V \models_\mu (n_{Q_1} \vee n_{Q_2}) \Leftrightarrow V \models_\mu n_{Q_1} \vee V \models_\mu n_{Q_2}$. The certainty is defined as $\mu = max\{\mu_1, \mu_2\}$. The certainty values of matching n_{Q_1} and n_{Q_2} are μ_1 and μ_2, respectively.*

While matching an *or-node* is straightforward, matching an *and-node* is more

complex. An *and-node* can be satisfied by an ordered pair of virtual services. The basic assumptions underlying the semantics of *and-nodes* are the following:

—In order to allow relaxed service retrieval, an *and-node* can be satisfied by a composition of operations. For instance, the query $(in, GPSPosition) \wedge (out, Hospital)$ might be satisfied by a single service (*find nearest medical center*) or by a composition of two services (*contact emergency* and *find nearest medical center*).

—If two services satisfy an *and-node* with equal certainty (the *ceteris paribus* - "all other things being equal" - of our model), then the shortest composition of operations will be chosen. In the context of the previous example, the service *find nearest medical center* will be chosen, as its composition length is 0. The rationale of this assumption is that any operation added to an existing composition reduced the overall certainty of the composition.

—The order of the elements in the query is important. If an *and-node* is satisfied by a composition, the left child of the *and-node* $(In, GPSPosition)$ should precede the right child $(Out, Hospital)$. As users search for procedural artifacts, we assume that there is a direct link between the location of elements within the query and the location of operations within the procedure.

In conjunction matching, the two query child nodes form a simple pattern, starting from the leftmost node, and ending with the rightmost node. The pattern is matched against the service network, resulting in a correspondence value that depends on the correspondence of the nodes and the certainty of the composition. The formal μ-satisfiability specification of *and-node* is as follows:

Definition 6 Conjunction Matching. *We say that* $V \models_\mu (n_{Q_1} \wedge n_{Q_2})$ *if the following conditions hold:*

(*1*) *V contains two sub services,* V_1 *and* V_2*, such that* $V_1 \models_\mu n_{Q_1} \wedge V_2 \models_\mu n_{Q_2}$*, and* $\mathcal{BASE}$ *contains a path which starts with* V_1 *and ends with* V_2*. Since the query was transformed into disjunctive normal form, any node can be either an* and-node *or a leaf node.*

  (*a*) *If* n_{Q_i} *is a leaf node* V_i *holds a single operation, and path matching is based on the operation as a starting or ending point.*

  (*b*) *If* n_{Q_i} *is an* and-node V_i *is a sequence of operations. The path matching starts with the first operation of the sequence (if* n_{Q_i} *is the left node), or*

the last operation of the sequence (if n_{Q_i} is a right node).

(*2*) *The overall composition certainty of the path is higher than a given threshold.*

The composition certainty reflects the certainty of the *dependencies* between operations. Recall from Section 3.1 that each dependency is associated with a certainty value, denoted as γ_D. We define $path(op_1, op_2)$ as the set of edges belonging to the shortest path between two operations, op_1 and op_2. The composition certainty function γ_{cc} is calculated as the product of edge certainty of the path, as is common in the literature [Do and Rahm 2002]:

$$\gamma_{cc(l,k)} = \prod_{(i,j)\in path(op_1,op_2)} \gamma_{D(i,j)}$$

Finally, γ_{cc} is bounded by a threshold, $\hat{\gamma_{cc}}$:

$$\gamma_{cc} = \begin{cases} \gamma_{cc} & , \gamma_{cc} > \hat{\gamma_{cc}} \\ 0 & , \gamma_{cc} \leq \hat{\gamma_{cc}} \end{cases}$$

Note that Definition 6 accepts situations in which the *and-node* is satisfied with a single operation, *i.e.*, $op_1 = op_2$, and the path has a length of 0. Moreover, it is likely that single-operation results will receive high certainty value, as their composition certainty is maximal.

In $\mathcal{OPOSSUM}$, partial results are allowed to be retrieved by relaxing condition (1) of Definition 6. We redefine $\models_\mu$ to accept partial services that do not necessarily satisfy the full *conjunctive chain*. We define $V^p \subset V$ as a partial virtual services, which is contained in V.

Definition 7 Partial Conjunction Matching. *We say that* $V^p \models_\mu (n_{Q_1} \wedge n_{Q_2})$ *if the following conditions hold:*

(*1*) *The partial service satisfies at least one of the child nodes:* $V^p \models_\mu n_{Q_1} \vee V^p \models_\mu n_{Q_2}$.

(*2*) *The partial certainty is higher than the conjunction threshold.*

The partial certainty takes into account the proportion of the partial service, with respect to the complete service, and is defines as:

$$\mu(n_{Q_1} \wedge n_{Q_2}, V^p) = \frac{|V^p|}{|V|} \min\{\mu(n_{Q_1}, V^p), \mu(n_{Q_2}, V^p)\}$$

The certainty function ranks the conjunction subsets according to their size, giving higher score to larger subsets. The highest certainty will be given to V itself -

the service that answers the complete intersection. The remaining subsets will receive monotonically non-increasing scores. To demonstrate the relaxed conjunction semantics, consider a simple query $(in, \text{Address}) \wedge (out, \text{Hospital})$. Let us evaluate the query against the subset of $\mathcal{BASE}$ depicted in Figure 2. The leaf node $(in, Address)$ is matched with a single operation, *Find Position*, with a certainty score of 1. The leaf node $(out, Hospital)$ is matched with a single operation, *find nearest medical center*, with a certainty score of 0.85 (due to the inexact matching of *emergency medical center* with *hospital*). The relaxed conjunction between the nodes is the set:

(find position, find nearest medical center), *(find position)*, *(find nearest medical center)*

The first virtual service is a path starting with *find position* and ending with *find nearest medical center*. The path has an initial certainty score of 0.93, based on the composition certainty. The approximate conjunction recalculates the score, assigning 0.93 to the first virtual service, 0.5 to the second virtual service and 0.43 to the third service.

4.5 Query Language Extensions

We now present an extension mechanism for the query language. Its aim is to allow users to write advanced queries without compromising the simple syntax of the query language. There are two types of syntax extensions and property extensions.

Syntax extensions extend the query language by adding syntactic sugar. In order to demonstrate our approach, we define two syntax extensions: the **optional** expression and the **any** expression. Unlike the default configuration, which mandates that all the query parts be retrieved, the *optional* extension allows users to define optional query phrases. For example, in the query "*address hospital optional(availability)*", the last token is optional, and therefore, results which contain the *availability* property will be assigned the same ranking as results which do not contain it.

The implementation of the extension is simple. It is based on rewriting the query using disjunctions in the preprocessing phase. Each query of the type "$x \wedge$ *optional(y)*", will be transformed to a query of the type "$(x \wedge y) \vee x$". Thus, results satisfying x and results satisfying both x and y will be ranked equally. In order

to avoid illegal queries, queries which contain solely optional expressions, such as "*optional(y)*", are not allowed.

The **any** expression allows users to define sets of options. The user can specify different options for a single property. For example, if the user wish to select services with an output which is either hospital, clinic or doctor, the following query pattern can be used: "*address any(hospital,clinic,doctor)*".

Property extensions allow definitions of new property categories for concepts. The basic definitions of the query language include three types: *in* (for input), *out* (for output) and *op* (for a concept which is assigned with the operations). However, services may have more specific properties that can be used in retrieval. Examples for interesting properties include the following:

—**Price:** The price of using the operation.

—**Availability**: The times in which the service is available.

—**Provider:** The organization which provides the service.

—**Location:** The geographical location in which the service is carried out.

—**Language:** The interface language used by the service (*e.g.* English, Hebrew, Arabic).

Extension properties are defined by the users, simply by assigning a label to a specific property of all, or some, of the semantic Web services. Thereafter, the user can use this name for restricting the results according to a certain value of the property, writing queries such as "*flight provider:singapore location:new york*". The query evaluator maps the value following the property name to a value assigned with the original concept, before continuing with the retrieval process.

5. INDEXING AND QUERY EVALUATION

In this section, we discuss the indexing method for the service model. The objective of the index is to enable efficient evaluation of queries with respect to processing time and storage space. The index is composed of two data structures: $I_{concepts}$ and $I_{services}$. $I_{concepts}$ is a hash-based index that maps concepts to their associated operations, allowing efficient evaluation of query concepts. $I_{services}$ is a graph-based index that represents the structural summary of the service network, and is used to answer queries that require several atomic operations. This section is organized as follows. Section 5.1 discusses the structure of the $I_{concepts}$ index and describes its

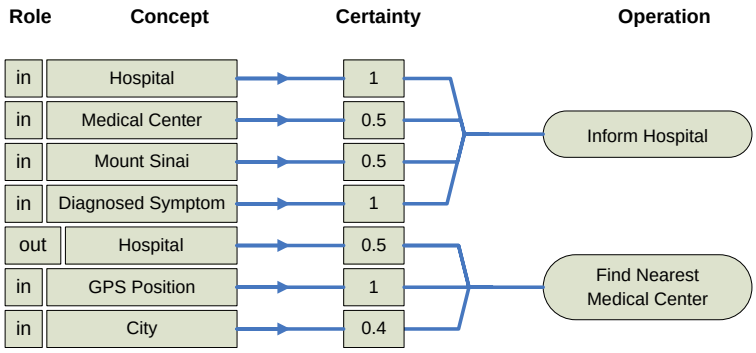

Fig. 7. An example of $I_{concepts}$ - the concepts index

construction process through concept expansion. Section 5.2 describes the structure and construction process of $I_{services}$. Section 5.3 describes and analyzes the query evaluation algorithm, operating on the index.

5.1 Concept Index

$I_{concepts}$ is based on a hash table, where each entry represents a concept, pointing to a node in $I_{services}$. Formally, $I_{concepts}$ immerse a mapping function, defined as:

$$I_{concepts} : C \times \{in, out, op\} \rightarrow G_N$$

C is defined as a set of concepts, $\{in, out, op\}$ is a property type (for input / output / operation), G_N is a set of operations index keys in $I_{services}$. Each mapping is associated with a certainty function, $\gamma_I(I_{concepts}) \rightarrow [0, 1]$, reflecting the semantic affinity between the concept and the concepts of the operation. Figure 7 represents an instance of $I_{concepts}$, which partially reflects the healthcare services running example (Figure 2). Concepts that serve as keys of $I_{concepts}$ are derived from the service model. For instance, *GPS position* is associated with an input parameter of the *find nearest medical center* operation, with a certainty of $\gamma_I = 1$. *hospital* is associated with an output parameter of *find nearest medical center*, with $\gamma_I = 0.5$. In this case, γ_I reflects a lower certainty, originating from the distance between the *hospital* concept and the *medical center* concept - the actual concept related to *find nearest medical center*.

$I_{concepts}$ is expanded with additional concepts that convey a broader meaning, in order to retrieve approximate services. Expanding the index is carried out through the index construction process. Constructing $I_{concepts}$ is a multi-phase process, in

Algorithm 1 Operation indexing in $I_{concepts}$

Input: OP_i, $\mathcal{O}$

Output: $I_{concepts}(OP_i) \subseteq I_{concepts}$

$I_{concepts}(OP_i) \leftarrow \phi$
for all $param \in OP_i$ **do**
  $c = l(param)$
  $I_{concepts} \cup (c, role\,(param)) \rightarrow I_{services}.OP_i$
  $\gamma_I(\rightarrow) = 1$
  for all $c' \in$ General(c) $\cup$ Specific(c)... **do**
    $\gamma' = d(c, c')$
    if $\gamma' > \hat{\gamma_I}$ **then**
      $I_{concepts} \cup (c', role(param)) \rightarrow I_{services}.OP_i$
      $\gamma_I(\rightarrow) = \gamma'$
    end if
  end for
end for

which a basic set of concepts is expanded with concepts that increase the retrieval scope of the index. Context classes are used in order to construct the key set of $I_{concepts}$, and to assign operations associated with each concept. Algorithm 1 describes the indexing process of an operation. The algorithm traverses all the parameters of an operation, adding the parameter's concept to the index. Following that, the algorithm adds index entries for concepts whose mapping certainty is higher than a given threshold.

5.2 Compact Service Index

$I_{services}$ represents the structural summary of the service network, using a directed graph. Given two operations, the objective of $I_{services}$ is to efficiently answer whether a composite service, starting with the first operation and ending with the second, can be constructed, and to calculate the certainty of the composition. Hypothetically, this task can be performed using the service base itself, by exhaustively searching for all possible compositions on the operation graph. Furthermore, indexing each path will result in an exponential number of index entries. Therefore,

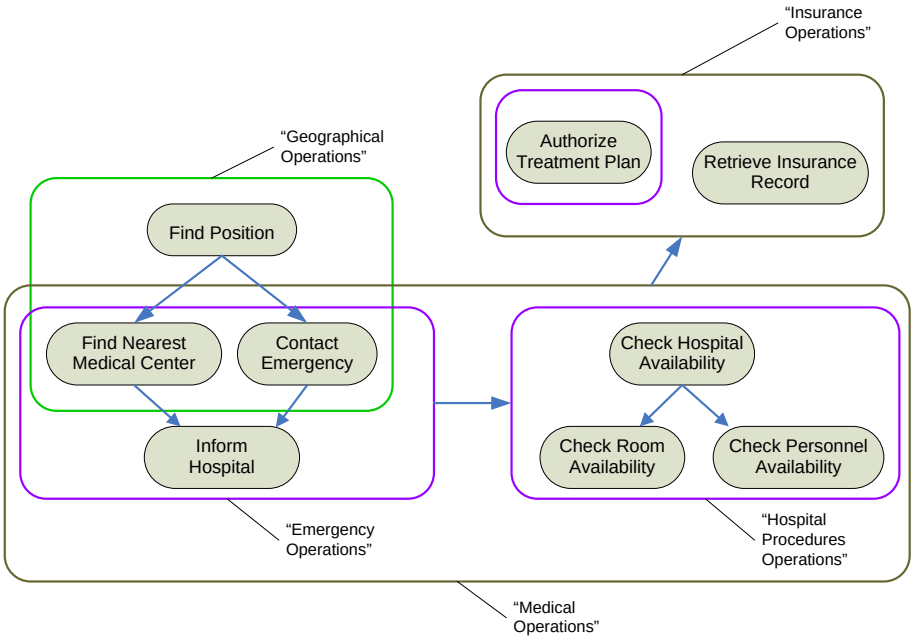

Fig. 8. An example of $I_{services}$ - the service network index

our main design goal was to design an index with minimal number of nodes and edges that would enable efficient traversal of the service network without compromising the precision of the results. The design of $I_{services}$ is based on principles taken from semantic routing in peer-to-peer networks. Due to the limited scope of the paper, we refrain from presenting the techniques in details. Rather, we present the main ideas via an example.

[Schmidt and Parashar 2004] and [Schlosser et al. 2002] proposed the use of semantic clustering to classify peer nodes to concepts and provide efficient traversal in peer-to-peer networks. In both methods the underlying ontology is segmented according to a multi-dimensional hierarchy, and each concept is assigned with a multi-level identifier that enables an efficient routing from source to destination concepts.

In $I_{services}$, operations are associated with multidimensional clusters, based on a set of clusters of their corresponding concepts. Figure 8 visualizes the operations, framed by the relevant clusters. Concept clusters are obtained by using the algorithm described in [Grau et al. 2005] for hierarchical clustering of OWL-Lite ontologies. All operations within a cluster have concepts with close affinity to each

other. For example, the operations *inform hospital* and *contact emergency* are located within the Emergency Operations cluster as they share similar concepts and have interrelated dependencies. Clusters are organized according to a hierarchy, where 0-level clusters represent atomic clusters (*e.g.* Emergency Operations), 1-level clusters contain 0-level clusters (*e.g. medical operations*) and so fourth.[3]

The number of edges in the index is reduced by replacing dependencies between operations with dependencies between respective clusters. For example, the dependencies between *inform hospital* and *check hospital availability*, and between *contact emergency* and *check hospital availability* are replaced by a single dependency between the emergency operations cluster and the Hospital Procedures Operations cluster. Dependencies exist only between clusters of the same level. For instance, if a dependency existed in $\mathcal{BASE}$ between *inform hospital* and *authorize treatment plan*, there will be no direct edge between the clusters, as emergency operations is a level-0 cluster, while insurance operations is a level-1 cluster. When evaluating a path that crosses multi-level clusters, higher level edges will be evaluated if lower ones do not satisfy the query. Thus, the search space is reduced. This method is efficient mainly due to the nature of the service network. Empirical results show that the service network is a sparse graph, and that most connections are between operations with similar semantics.

As operations contain several parameters, there is no guarantee that all of the parameters' concepts will belong to the same cluster. Therefore, operations are organized into multi-dimensional clusters, which reflect their different semantic affinities. For instance, the operation *contact emergency* has parameters involving geographical concepts and medical concepts, and is located in the Geographical Operations and the Emergency Operations clusters simultaneously. A query that requires a service that takes an address and returns hospital availability, will be answered by a path of operations that starts in the Geographical Operations cluster, goes through the Emergency Operations cluster (as there are mutual operations belonging to the two clusters), and ends at the Hospital Procedures Operations cluster. Multi-dimensional clustering is feasible as the number of parameters associated with an operation is bounded, and low. Empirical results show that over 90% of the services in our benchmark have 4 or less parameters.

[3]Clusters are nameless. We have named clusters for the sake of clarity.

5.3 Query Evaluation

In this section, we present an algorithm for query evaluation, based on the index discussed above. The algorithm is described in Algorithm 2. Given a query, Q, and the index, the algorithm returns a set of virtual services, $\{V_{(1)}, V_{(2)}, ..., V_{(k)}\}$, ranked according to their certainty. The algorithm starts with transforming the query into disjunctive normal form, resulting in a set of query parts, C. If a query part includes a single query node, then the results contains operations from $I_{concepts}$. The results are filtered by the *Prune* function, which removes services with lower certainty value than the threshold. If the query part includes more than a single node, then it contains a conjunction. The algorithm uses the *Route* function to find paths between origin operations (associated with the left-hand query node) and destination operations (associated with the right-hand query node). The functions *start* and *end* The function *Rank* ranks the virtual services according to their certainty.

We denote by $|C|$ the number of disjunctions in the query. $|OP|$ represents the number of operations associated in $I_{concepts}$ with a given query node (with certainty higher than the threshold). $|\mathcal{V}|$ is the number of results, N is the number of peers (operations) and b is the hypercube base - the number of dimensions needed to segment the ontology. The query evaluation algorithm complexity is given by:

$$O(|C| \cdot (|OP|^2 \cdot \frac{1}{2} \log_b N) + |\mathcal{V}| \log |\mathcal{V}|)$$

The main algorithm loop depends on the number of disjunctions, and runs in $|\vee|$ steps. The routing function iterates over the cartesian product of the operations returned by $I_{concepts}$. The complexity of *Route* is calculated in [Schlosser et al. 2002] to be $\frac{1}{2} \log_b N$, Finally, the complexity of the ranking of the results ($|\mathcal{V}| \log |\mathcal{V}|$) is added to the general complexity.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate our approach in three ways: a) by analyzing the precision of the search engine; b) by comparing precision and performance to OWLS-MX [Klusch et al. 2005]; c) by evaluating the scalability of our approach through simulation. Evaluation was based on an implementation of $\mathcal{OPOSSUM}$ using Java and MySQL server. A dedicated personal computer running Windows XP with 1.5GB RAM was used for all the experiments.

Algorithm 2 Evaluate Query

Input: Q, $I_{concepts}$, $I_{services}$

Output: $\mathcal{V} = \{V_{(1)}, V_{(2)}, ..., V_{(k)}\}$

$\mathcal{V} \leftarrow \phi$
$C = toDNF(Q)$
for all $C_i \in C$ **do**
  $n = C_i$.left-node
  $S_{source} \leftarrow I_{concepts}(l(n), pc(n))$
  $S_{source} \leftarrow Prune(S_{source})$
  if C.right-node $= \phi$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup S_{source}$
  else
    $n_{dest} = C_i$.right-node
    $S_{dest} \leftarrow I_{concepts}(l(n_{dest}), pc(n_{dest}))$
    $S_{dest} \leftarrow Prune(S_{dest})$
    for all $OP_i \in S_{source}, OP_j \in S_{dest}$ **do**
      $S_{compose} \leftarrow S_{compose} \cup Route(OP_i, OP_j)$
      $S_{compose} \leftarrow Prune(S_{compose})$
    end for
    $\mathcal{V} \leftarrow \mathcal{V} \cup S_{compose}$
  end if
end for
Return $Rank(\mathcal{V})$

In order to evaluate the search engine, we used OWLS-TC, an existing benchmark for semantic service retrieval, supplied by [Klusch et al. 2005]. OWLS-TC includes more than 550 services, which are semantically annotated using more than 40 different ontologies, from various domains, including economy, communication, and healthcare. In addition, OWLS-TC includes a set of predefined queries and relevance sets that enable to calculate precision and recall values of query results. OWLS-TC was augmented with queries and relevance sets that reflect composed services.

Ranking serves as the main method for expressing relevance and certainty in our

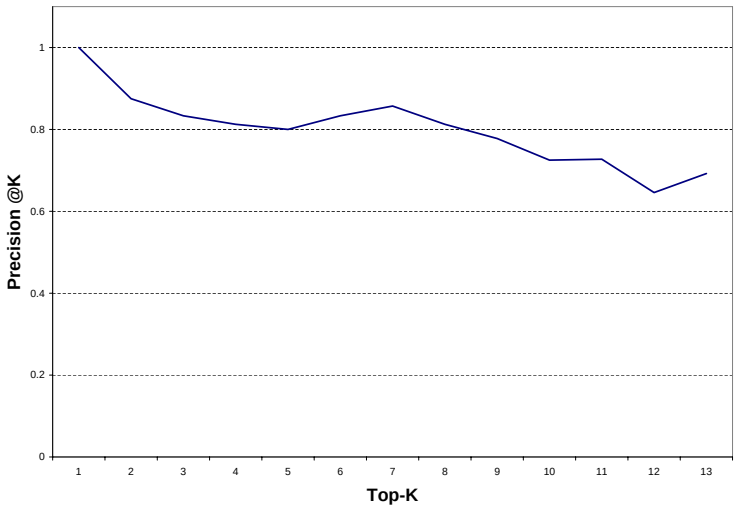

Fig. 9. Average precision at top K

approach. Therefore, we had measured the precision of the results in the top-K places, as depicted in Figure 9. Precision at top-K is calculated as $\frac{L_{q,k} \cap S_q}{L_{q,k}}$, where S_q is defined in the benchmark as the set of services that are relevant to a query q, and $L_{q,k}$ is the top k results on the list. The results show that services with high certainty (and therefore, higher ranking) were found to be more relevant than services with lower certainty. We explain the descent of precision around the top 3 and 4 results by the precedence of shorter services, derived from the method of calculating the compositional certainty. If this precedence will be canceled, the precision of the top 1 and 2 places will descend.

We had compared the precision/recall values of $\mathcal{OPOSSUM}$ with those of OWLS-MX by running the OWLS-TC queries. Our results show that we succeeded in matching our precision/recall performance to those of OWLS-MX. However, the two methods vary considerably in the query response time. Table I presents a comparison of average response time of $\mathcal{OPOSSUM}$ and OWLS-MX.[4] The results clearly show the benefits of an indexing mechanism, which improves the performance of the query evaluation algorithms by an order of magnitude.

[4]It is worth noting that the average query response time we had measured of OWLS-MX were slightly higher than those reported by [Klusch et al. 2005]. The difference can be attributed to the different hardware configurations of the testing platforms.

Table I. Average query response time of $\mathcal{OPOSSUM}$ vs. OWLS-MX (measured in ms)

Query	**OWLS-MX**	**OPOSSUM**
hospital investigating	1710	33
book price	1647	35
country skilled occupation	1742	20
car price service	1682	15
geopolitical entity weather process	1364	27
government degree scholarship	1782	32
novel author	1662	40

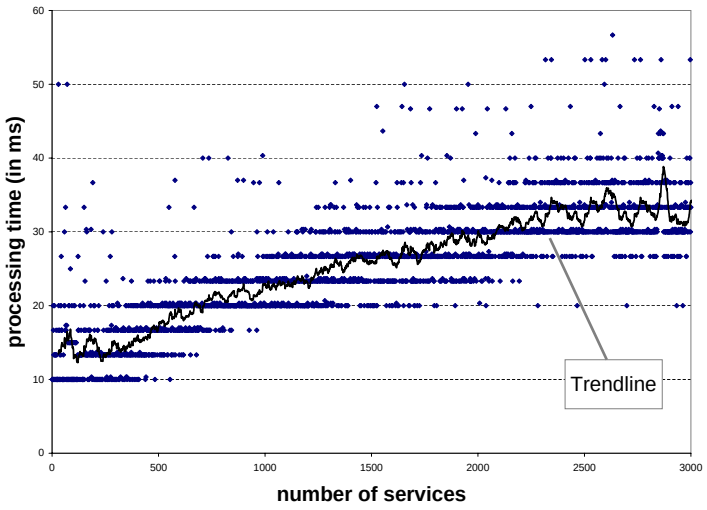

Fig. 10. Precessing time for query evaluation

The scalability of our approach was evaluated by simulating large numbers of semantic Web services. Using the existing 500 OWLS-TC benchmark services as a core, 2500 additional services were simulated by imitating the properties of core services. The service generation function was parameterized using 3 random variables: p - the number of parameters, nc - whether to associate the parameter with a new concept or with an existing one, c - the identity of the associated concept, if nc is false. Figure 10 represents the average query response time, depending on the number of services in the index. The black line represents a linear trend line on top of the discrete measurements. While the number of services increased by a factor of 3000 (from 500 to 3000), the average response time increased by a factor of 2.3

(from 15 ms to around 35 ms). The results exhibit the scalability of our indexing approach.

7. RELATED WORK

Activity in the area of service-retrieval can be divided into three main approaches: keyword-based, semantics-based and behavioral matching. In this section, we overview these approaches in that order.

7.1 Keyword-Based Approaches

Currently, the keyword-based approach is the most wide spread in the industrial attempts to implement service retrieval. The most prominent example is the UDDI protocol [Bellwood et al.], which is an industry standard for locating Web services through keyword and category search. The main drawback of UDDI, and with other keyword-based approaches, is the lack of sufficient information for describing Web services. Web service interface are defined using WSDL descriptions, which contain a very small amount of information regarding Web service operations. Therefore, keyword search solutions fail in providing satisfactory recall for Web service search [Ankolekar et al. 2001; Sirin et al. 2003].

7.2 Semantic Approaches

Several techniques have been proposed to deal with service discovery using logical inference. These approaches are based on an ontology-based formal description of Web services. Several works, including those of Paolucci et al. [Paolucci et al. 2002] and Sirin et al. [Sirin et al. 2003], propose a method based on OWL-S for matching requests and advertisements of semantic Web services. The OWL-S profile ontology is used to describe the capabilities of services, and service matching takes the form of logic inference over the properties of the services. Another approach for semantic matching of Web services is based on planning methods taken from the AI research domain. In [Traverso and Pistore 2004] semantic Web services are translated into state transition machines, and the composition problem is defined as a planning problem over the available services, with the required composition defined as the planning goal. While all of these works provide precise matching, they exhibit a limited notion of relaxed matching, based on the hierarchy of sub types.

Several research initiatives suggest hybrid approaches for semantic Web services

discovery, augmenting logic-based methods with content-matching techniques. Traverso and Pistore [Syeda-Mahmood et al. 2005] describe a hybrid approach for Web services discovery, combining methods based on thesaurus and ontological inference. In [Klusch et al. 2005] and in [Bernstein and Kiefer 2006] logical inference is compared with content-based matching. The latter was found to performs better both in terms of recall and precision. Furthermore, hybrid approaches that combine properties of logic inference and content matching were shown to outperform any of the pure approaches.

Both logic-based approaches and hybrid approaches aim at automatic composition. Therefore, they require a highly accurate and trustful description of services and queries, as well as an unambiguous matching process. In contrast, our work relies on adjustable confidence values for semantic mappings. It allows the use of syntactic service description, such as WSDL. Also, our work relaxes the semantic and structural evaluation of compositions, expressing the approximation level through ranking. Furthermore, we use a user-oriented form of query language, automatically translating a simple keyword query to a formal structure of concepts. While the hybrid approach relaxes the semantic matching by using content-based similarity methods, our work relies solely on ontologies for semantic matching. Our relaxation methods rely on a broader definition of equivalence between ontological concepts and on approximating the structure of the composition imposed by the query. Finally, $\mathcal{OPOSSUM}$ outperforms the semantic approaches described above in response time, due to the utilization of indexing methods.

7.3 Behavioral Matching

BP-QL [Beeri et al. 2006] is a query language for BPEL4WS [Wohed et al. 2003] process definitions. It uses a graph-based visual query language that represents a BPEL script and searches for a subgraph isomorphism in a repository of BPEL scripts. While BP-QL supports an expressive query language, it ignores semantic attributes of services. Klein et al. [Klein and Bernstein 2004] present a method for recognizing semantically-annotated services through pattern matching of activity sequences. Shen et al. [Shen and Su 2005] encode semantic Web services and queries as regular expressions, defining matching as the intersection between them. Indexing methods for regular expressions were introduced to enhance behavioral matching performance. These methods differ from our approach in two main as-

pects. First, these methods evaluate a query against each service independently while our proposed work matches queries against service networks, dynamically built from isolated operations by analyzing and inferring relations between operations. Second, these methods provide limited support for approximate matching, and do not rank results according to their semantic and compositional certainty.

8. CONCLUSIONS AND FUTURE WORK

In this work we have introduced a semantic approach to Web service retrieval. Our approach is based on three aspects: **a)** using the current research in semantic Web services to enrich the querying abilities of users, **b)** using approximation techniques to increase the recall of possible service compositions, and **c)** exploring indexing techniques for sub-linear response time. To motivate our approach, we have proposed Web service composition as an exploratory process, in which designers seek the use of existing Web services to gain a leading edge in their businesses. Composition as exploration suggests that services can be composed by the use of additional gluing effort even if they do not match exactly. Such an approximate matching therefore accounts for the amount of extra effort needed, and this is reflected in the ranking of the results. The ranking combines both the semantic distance between the query and the result and the partiality of the result. Thus, a lower ranking suggests additional gluing effort in order to bridge the semantic distance between the components or to implement the missing functionality.

We propose a general framework of a service search engine, where the various components of our work provide solutions to issues of indexing, retrieval, and ranking within this framework. As a proof of concept, we have built $\mathcal{OPOSSUM}$, a Web service search engine and share our experiences with discovering, indexing, querying, and ranking using real-world data.

The contributions of this research are at conceptual, semantic, and computational levels. First, we define an efficient, graph-based data structure for organizing services. Second, we provide a semantically-rich query language, allowing both simple and advanced service search capabilities. Finally, we provide a sub-linear service retrieval algorithm.

There are several directions for future work. We are currently extending our research to include a broader notion of service ranking, based on service reusability (the ability to use a service in a given context). Furthermore, we intend to extend

the applicability of our approach to syntactic Web services (represented by WSDL documents) and Web forms. Also, we intend to extend $\mathcal{OPOSSUM}$ with advanced capabilities, including multi-lingual ontology alignment and content-based matching techniques. We plan to offer $\mathcal{OPOSSUM}$ as a service to the general Web community, allowing users to submit Web services and to query the database. By opening the engine to the public we hope to gain realistic information that would help us characterize searching and utilization patterns for Web services. Finally, we intend to apply $\mathcal{OPOSSUM}$ for model-driven engineering of information systems, embedding semantic service retrieval within service development environments.

REFERENCES

Ankolekar, A., Martin, D. L., Zeng, Hobbs, J. R., Sycara, K., Burstein, Paolucci, M., Lassila, O., Mcilraith, S. A., Narayanan, S., and Payne. 2001. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop (SWWS)*. 411–430.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., and Stein, L. 2004. OWL web ontology language reference. W3c candidate recommendation, W3C.

Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T. 2006. Querying business processes. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06)*. VLDB Endowment, 343–354.

Bellwood, T., Clement, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y., Januszewski, K., Lee, S., B., M., Munter, J., and von Riegen, C.

Bernstein, A., Kaufmann, E., Bu"rki, C., and Klein, M. 2005. How similar is it? towards personalized similarity measures in ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*.

Bernstein, A. and Kiefer, C. 2006. Imprecise rdql: towards generic retrieval in ontologies using similarity joins. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. ACM Press, New York, NY, USA, 1684–1689.

Cardoso, J. and Sheth, A. 2003. Semantic E-Workflow Composition. *Journal of Intelligent Information Systems 21,* 3, 191–225.

Christensen, E., F., Meredith, G., and Weerawarana, S. 2001. Web services description language (wsdl) 1.1. Specification document, W3C. Mar.

Do, H. and Rahm, E. 2002. COMA: A System for Flexible Combination of Schema Matching Approaches. *Proceedings of the 28th Conf. on Very Large Databases (VLDB'02)*.

Euzenat, J. and Valtchev, P. 2004. Similarity-based ontology alignment in OWL-lite. *Proceedings of ECAI*, 333–337.

Gal, A., Modica, G., Jamil, H., and Eyal, A. 2005. Automatic ontology matching using application semantics. *AI Magazine 26,* 1.

Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A. 2005. Automatic Partitioning of OWL Ontologies Using E-Connections. *Proc. of the 2005 International Workshop on Description Logics*.

Klein, M. and Bernstein, A. 2004. Towards high-precision service retrieval. *IEEE Internet Computing 8,* 1 (January), 30–36.

Klusch, M., Fries, B., Khalid, M., and Sycara, K. 2005. Owls-mx: Hybrid semantic web service retrieval. In *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*. AAAI Press.

Lara, R., Roman, D., Polleres, A., and Fensel, D. 2004. A conceptual comparison of wsmo and owl-s. In *Proceedings of the European Conference on Web Services (ECOWS'04)*. Lecture Notes in Computer Science, vol. 3250. Springer-Verlag, 254–269.

Medjahed, B. and Bouguettaya, A. 2005. A multilevel composability model for semantic web services. *IEEE Transactions on Knowledge and Data Engineering 17,* 7, 954–968.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. 2003. Composing web services on the semantic web. *VLDB J. 12,* 4, 333–351.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. 2002. Semantic matching of web services capabilities. In *International Semantic Web Conference*. 333–347.

Schlosser, M., Sintek, M., Decker, S., and Nejdl, W. 2002. Hypercup: hypercubes, ontologies, and efficient search on peer-to-peer networks. *Proceedings of the 1st Workshop on Agents and P2P Computing*.

Schmidt, C. and Parashar, M. 2004. A peer-to-peer approach to web service discovery. *World Wide Web Journal 7,* 2, 211–229.

Shen, Z. and Su, J. 2005. Web service discovery based on behavior signatures. In *2005 IEEE International Conference on Services Computing (SCC'05)*. 279–286.

Sirin, E., , Hendler, J., and Parsia, B. 2003. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*.

Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A.-A., and Goodwin, R. 2005. Searching service repositories by combining semantic and ontological matching. In *Third International Conference on Web Services*.

Toch, E., Gal, A., and Dori, D. 2005. Automatically grounding semantically-enriched conceptual models to concrete web services. In *Proceedings of the International Conference on Conceptual Modeling (ER'05)*. Lecture Notes in Computer Science, vol. 3716. 304–319.

Traverso, P. and Pistore, M. 2004. Automated composition of semantic web services into executable processes. In *Proceedings of the International Semantic Web Conference 2004 (ISWC'04)*. Springer-Verlag, 380–394.

Vossen, P. 1998. Eurowordnet: a Multilingual Database with Lexical Semantic Networks. *Computational Linguistics 25,* 4.

Wohed, P., van der Aalst, W. M. P., Dumas, M., and ter Hofstede, A. H. M. 2003. Analysis of web services composition languages: The case of bpel4ws. In *Proceedings of the International Conference on Conceptual Modeling (ER'03)*. Lecture Notes in Computer Science, vol. 2813. Springer-Verlag, 200–215.