Approximate Service Retrieval

Eran Toch

Approximate Service Retrieval

Research Thesis

Submitted in Partial Fulfillment of the
Requirement for the Degree of
Doctor of Philosophy

Eran Toch

Submitted to Senate of
the Technion - Israel Institute of Technology

Av 5778, Haifa, Israel, August 2008

Acknowledgments

One of the best things in completing a doctoral dissertation is that I finally have the opportunity to thank all the great people who helped me and made this moment possible. My first thanks go to my advisors: Dov Dori and Iris Reinhartz-Berger. Dubi has been a source of support and guidance. Most importantly, he allowed me to roam free in distanced research fields, and I thank him sincerely for this opportunity. Iris has been my thorough critique, always pushing me to reevaluate my views and results and to improve them. Her fruitful feedback had been extremely helpful and influential.

Lots of colleagues and friends had helped my through the Ph.D. I would like to thank Avigdor Gal, who's advice is priceless. It was a pleasure collaborating with him. I would like to thank Dizza Beimel from the bottom of my heart. Her friendship was one of the most important things for me in this doctoral course. The friendship of Yael, Lera, Victor, and Ola were also a hugh source of fun and insights.

Lastly, I would like to thank my family. My parents had shown me the path to the scholary world and I happily followed it. I am very joyful to make you proud. My kids, Ori and Shira, were born through my Ph.D. They had brought me great joy, and they are the best possible distractions from work. My foremost thank goes to Osnat, my wife and my love. Without her ongoing support, acceptance and help, I could not have written this.

The generous financial help of the Ministry of Science and the Levi Eshkol Foundation is gratefully acknowledged.

The generous financial help of the Technion - Israel Institute of Technology is gratefully acknowledged.

Contents

1 Introduction **4**

2 Background **9**

2.1 Service Representation 9

2.1.1 Syntactic Service Representation 9

2.1.2 The Semantic Web 11

2.1.3 Semantic Service Representation 16

2.2 Approaches to Service Retrieval 17

2.2.1 Classification of Approaches 18

2.2.2 Semantic Approaches 18

2.2.3 Syntactic Approaches 22

2.2.4 Service Composition 22

3 Approximate Service Similarity **24**

3.1 Definitions 25

3.1.1 Services and Operations 25

3.1.2 Compositions 25

3.2 Service Similarity Patterns 26

3.2.1 Adjustment-Based Affinity 27

3.2.2 Virtual Operations 30

3.2.3 Semantic Affinity Patterns 31

3.2.4 Functional Affinity Patterns 37

3.2.5 Similarity Function Properties 38

3.3 Evaluating Affinity Patterns 41

3.3.1 Experiment Design 42

3.3.2 Results 45

3.3.3 Discussion 52

4 Service Retrieval **55**
4.1 Query Evaluation Semantics . . . 55
4.1.1 Query Syntax . . . 55
4.1.2 Query Language Extensions . . . 57
4.1.3 Query Matching Semantics . . . 58
4.1.4 Complex Query Semantics . . . 59
4.2 The Service Network . . . 61
4.2.1 Definitions . . . 61
4.2.2 Deriving Empirical Dependencies . . . 63
4.2.3 Inferring Dependencies . . . 65

5 Algorithms for Efficient Retrieval **67**
5.1 Efficient Query Evaluation . . . 67
5.1.1 Concepts Index . . . 68
5.1.2 Composition Index . . . 72
5.1.3 Query Evaluation . . . 76
5.2 Experimental Evaluation . . . 78
5.2.1 Precision and Recall . . . 78
5.2.2 Performance and Scalability . . . 80
5.3 Implementation . . . 81
5.3.1 Architecture . . . 82
5.3.2 Liquid Interface . . . 86

6 Conclusions **89**
6.1 Summary of Results . . . 90
6.2 Future Directions and Open Problems . . . 91

7 Appendixes **93**
7.1 Object-Process Methodology (OPM) . . . 93
7.2 Aligning Ontologies . . . 94

List of Figures

1.1 The OPOSSUM search engine 7
1.2 The service prototype 8

2.1 An example of a WSDL document 10
2.2 Layers of the Semantic Web 11
2.3 An example of healthcare domain ontology 15
2.4 An example of Semantic Web Services 16
2.5 The service retrieval model 17
2.6 Categories of logic matching degrees 19

3.1 An example for affinity between operations 27
3.2 Operation *a* is simulated using operation *b* 28
3.3 Constructing virtual operations according to the set hierarchy pattern . . . 33
3.4 Constructing virtual operations according to the relation pattern 35
3.5 Operation *a* is matched with operation *b* using by simulating *b* using *a* . . 37
3.6 A sample survey page showing the model and the feedback form 43
3.7 A sample of two test cases 45
3.8 The average score of completeness, exactness, and usefulness 46
3.9 The average ranking for subclass relations, according to the subset direction from the baseline 47
3.10 The average completeness score ordered according to the c function . . . 50
3.11 The average ranking according the instance-classification distance 51
3.12 The average ranking according to the edit distance between the query and the composition 52
3.13 Comparison of logic-based similarity versus our results 53

4.1 An abstract query 56
4.2 An Example of Operations and Dependencies 62

4.3 Transformation Patterns for OWL-S Control Constructs 64

5.1 An example of $I_{concepts}$ - the concepts index 68
5.2 Interconnected components on the 71
5.3 An example of the construction of $I_{services}$ 73
5.4 An example of $I_{services}$ 74
5.5 An example set of clusters 76
5.6 Average precision at top K 79
5.7 Recall-Precision comparison with OWLS-MX 80
5.8 Precessing time for query evaluation 82
5.9 Index size as a factor of the number of services 83
5.10 The Architecture of the OPOSSUM Search Engine 84
5.11 The homepage of the OPOSSUM Search Engine 85
5.12 The results page of the OPOSSUM Search Engine 85
5.13 The Liquid-Interface UI Generator 87

7.1 OPM Legend 93

List of Tables

2.1 Abbreviated syntax and semantics of OWL 14
2.2 Categories of service retrieval 19

3.1 Context classes 41
3.2 Examples of textual feedback 48
3.3 Location of concepts within the subclass hierarchy 49
3.4 The relation between composition size and completeness decline 52
3.5 α values for different patterns 54

5.1 Average query response time of our approach vs. OWLS-MX (measured in ms) 81

7.1 Cardinality participation constraints in OPM 94

Abstract

The rigidness of enterprise software systems is a fundamental problem in information systems engineering, demonstrated by the high expenses involved in developing new systems or enhancing existing ones. The problem is becoming acute as the pace of changes in information systems becomes faster.

The field of Semantic Web addresses this problem by formally specifying Web services and extending the current World-Wide-Web with formal languages and ontologies that describe domain knowledge. However, most works in the field deal with automatically inferring Web service compositions using logic-based methods. This approach has severe limitations when dealing with incomplete and uncertain information, which is the wide spread case of Web services. Furthermore, experiments we conducted indicate that the notion of similarity used by logic-based service discovery approaches is not compatible with the perception of human subjects. Rather, humans employ a much broader concept of similarity, which includes relations between services and similarities between instance sets.

This research advocates for approximate service retrieval in which the retrieval process returns approximate results. The basic question of this research is, therefore: what is considered valid approximation in service retrieval? For answering this question, we defined and evaluated a set of similarity (affinity) patterns for semantic Web services, i.e., semantically annotated Web services. The patterns are based on both semantic approximation and functional approximation: semantic approximation is done using the structure of the ontology, while the functional approximation uses the composition structure. Similarity is calculated by measuring the inherent information value lost in bridging the differences between two services. In comparison to the state-of-the-art works in the field of semantic Web service retrieval, our approach produced results with better recall and with equal precision.

As the new notion of approximate similarity imposes new challenges on the characteristics and complexity of service retrieval algorithms, we present in this research an

efficient method for approximate retrieval of Web services, which is based on indexing of semantic and functional service properties. Our results show that this index allows us to present an algorithm for query processing which is sub-linear in time complexity on the average case.

Finally, we demonstrated a proof of concept of our approach by developing OPOSSUM, a search engine for Web services. The results of this search engine are automatically transferred into a Web-based user interface, called Liquid-Interface, allowing the user to immediately use prototypes that simulate the approximate compositions functionality.

Abbreviations and Notations

Symbol	Meaning	Page
$\mathcal{O}$	An Ontology - a formal conceptualization of a domain	13
C	Concept class	41
$i : C$	Instance relation in an ontology interpretation	41
$C_1 \equiv C_2$	Equivalent concepts	
$C_1 \sqsubseteq C_2$	Subclass relation between concepts	41
OP	Atomic operation which is part of a Web service	25
S	Service, a named set of operations	25
Com	Composition, a graph of operations (nodes) and data flows (edges)	26
VOP	Virtual operation	33
Ψ	Constructor, which construct a set of virtual operations	30
$N_\perp$	Bottom cardinality boundary	32
$N_\top$	Top cardinality boundary	32
Sim	Similarity function	38
μ	Constructor certainty function	32
Q	An abstract query structure	56
$\mathcal{SN}$	Service network, a repository of interconnected Web services	61
$\mathcal{OP}$	The set of all operations within a service network	61
γ_D	Dependency certainty	61
γ_C	Concept certainty	68
γ_C	Service relation certainty	74

Chapter 1

Introduction

Web services are distributed software components, accessed through the World Wide Web. They are considered primary objects to be reused and combined in the process of designing and implementing new applications. An important step in this process is to find the required Web services and to arrange them according to the desired application plan. Semantic description of Web services (known as *semantic Web services*) are considered the primary mechanism in providing a precise and rich data model for Web services. Semantic Web services aim 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. Languages such as OWL-S [2] and WSML [29] provide an unambiguous description by mapping service properties (such as input and output parameters) to common concepts and by providing additional meta-data regarding the service. The concepts are defined in *ontologies* [6] on the semantic Web [12], which serve as the key mechanism to globally define and reference common understanding.

A major portion of the research involved in semantic Web services has been in the context of Web service retrieval, which is the process of matching between a query and a set of existing services. It incorporates two levels of matching: *service discovery* and *service composition*. In service discovery, appropriate Web services are selected based on their matching with the query. However, in many cases, no single service can satisfy the query. Therefore, service composition yields a structured set of services, which together solve the problem at hand. Most retrieval approaches use logic-based proof inferencing in order to compare queries and service descriptions. Service properties and query elements are represented using description logics concept sets. If the concepts are identical, or the concepts of the services are subsumed by the concepts of the query, then the matching is

perfect. If the concepts of the query are subsumed by the concepts of the query, then the matching is partial [58, 50, 49].

The problem is that in real-world settings, the process of service retrieval requires a significant amount of approximation. It is often the case that only partial solutions to a query exist, as Web services are created autonomously without a-priori knowledge of their intended use. Partiality has two different aspects in this context: only part of the query can be matched or the whole query can be matched to less-than-perfect answers. Of course, the combination of the two aspects also exists, i.e., the only suitable results answer imperfectly only part of the query. While approximation is essential, existing approaches exhibit a very limited notion of approximation. The only means for approximation offered by these methods rely on concept hierarchies. For example, if an ontology includes some hierarchal structure (for instance, hospital is a type of organization), then a query asking for *organization* might receive *hospital* as a result. However, concept hierarchies cannot express valid approximations, as hierarchies are limited by nature. As a result, logic-based approaches often yield low recall [49, 69]. As a motivating example for this problem, consider the following query:

Example 1: Query. Find a set of services that accepts an address and returns the directions to the closest hospital.

Let us assume that the service repository only includes a service that accepts a *longitude/latitude* input parameter and returns the closest hospital. Human engineers would build a transformation function between the parameter longitude/latitude and the parameter address in order to answer the first half of the query. However, current retrieval approaches would fail to do so and would return a null answer.

In an attempt to answer queries such as that in Example 1, we advocate for *approximate service retrieval*. Our approach has the benefit of using existing services for increased reusability while limiting the number of required services, since adding missing code is always an option. Naturally, a human Web service composer is intuitively interested in finding the most similar composition for her needs, thus reducing the amount of coding needed. Therefore, there is a need for ranking search results according to the amount of modifications required for their utilization. The basic question of this research is therefore **what is considered valid approximation in service retrieval?**

While the amount of research in semantic Web service discovery and composition is fairly large (we counted over 500 different articles), we did not find a satisfactory answer to this significant research question. The framing of the question suggests two interesting aspects. First, when does a service approximate another service? This question calls for

a similarity measure for semantic Web services. While similarity measures exist in many fields, such as information retrieval and database research, they are absent from the field of service retrieval. Second, when is an approximation valid? In other words, when does a similar approximate service becomes too remote from the original? We seek, therefore, for an imprecise notion of similarity.

In this thesis we define and evaluate a set of approximation schema for semantic Web services. We have identified two dimensions of approximation: functional and semantic. Functional approximation relaxes the matching between the structure of the query and the structure of the services. Semantic approximation relaxes the matching between the concepts in the query and the concepts of the result. We provide a broader definition of approximation than the one of current approaches, which includes relations between services and concept hierarchies. For each dimension we defined a small set of affinity patterns, based on the structure of the ontology for semantic approximation and the composition structure for functional approximation. The patterns provide a quantitative distance metric for service similarity, which draws inspiration from information-theoretic approaches. Similarity is calculated by measuring the inherent information value lost in bridging the semantic and functional differences between two services.

The similarity measure is evaluated for relevance and correctness as judged by human subjects. The evaluation took the form of an experiment, in which software and information system engineering students were asked to rank different aspects of similarity in a service retrieval scenario. Our findings show that the notion of similarity used by logic-based service discovery approaches is not compatible with the perception of human subjects. Rather, humans employ a much broader concept of similarity, which includes elements such as relations between services and similarities between instance sets. Furthermore, human subjects evaluate similarities in a much softer form than prior approaches predict. We used the results from the experiment to enforce our definition of similarity measure.

The notion of approximate similarity measure requires updating the service retrieval paradigm, specifically, the syntax and semantics of queries. We introduce a query language, which is based on a tree-structure. The language allows the user to write queries in keyword-based language (similar to the ones used in search engines for the Web, such as Google), and an OPM (Object-Process Methodology) modeling language [32]. The basis of the query semantics definition is a graph-based model for representing the relations between Web services.

Approximate query evaluation has several implications regarding space and time complexity. For example, how do different approximation methods affect processing com-

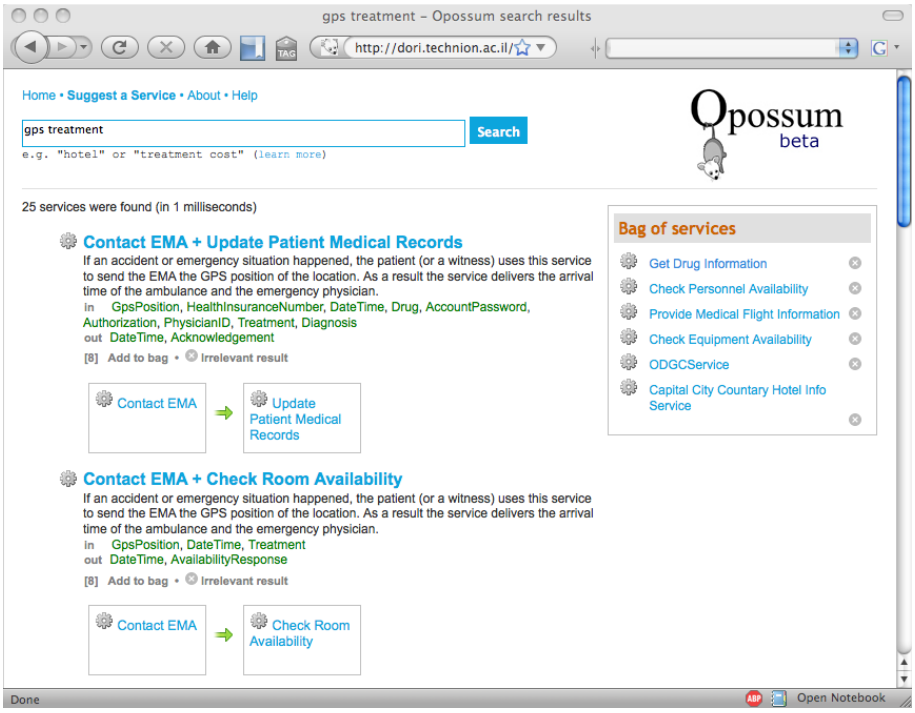


Figure 1.1: The OPOSSUM search engine

plexity? We present an efficient method for approximate retrieval of Web services, which is based on efficient indexing of semantic and functional service properties. The retrieval process relies on graph-based indexing, in which connected services can be approximately composed, where graph distance represents service relevance. The algorithm provides service ranking that is based on the certainty of matching with a query. The index data structures allows us to present a sub-linear service retrieval algorithm by using a two-level index mechanism, representing concepts and compositions. We use semantic clustering techniques in order to supply a compact representation of the index. We formally prove the correctness of the index and experimentally show its measures of preciseness and scalability.

We demonstrate a proof of concept of our results by developing OPOSSUM, displayed in Figure 1.2. **OPOSSUM** (Object-PrOcess-SemanticS Unified Matching) is a search engine for Web services which employs the methods presented in this research, including approximation methods based on semantic and functional similarity. The results of the search engine are transferred into another system, called **Liquid-Interface**, which automatically creates a prototype of the composition (see Figure 5.13). Together, the two systems allow the user to approximate search for Web services on the Web, and

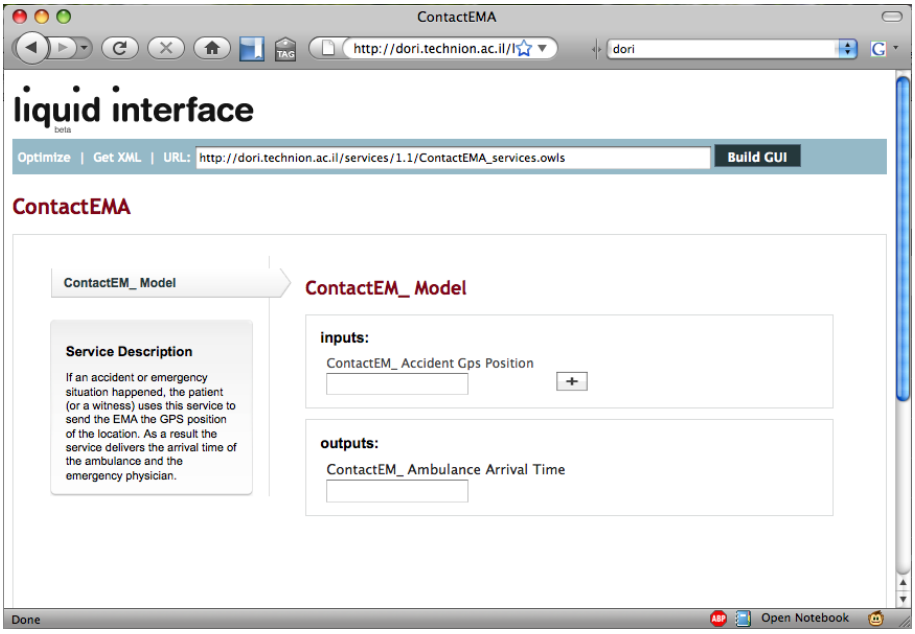


Figure 1.2: The service prototype

to immediately use a prototype that simulates the composition functionality.

The rest of the thesis is structured as follows:

- Chapter 2 covers the background of the work, including Web services, semantic Web services, ontologies, and current approaches for service retrieval.
- Chapter 3 provides a definition of similarity measures for Web services and describes their experimental evaluation.
- Chapter 4 studies how the similarity measures defined in the previous chapter can be used in a service retrieval framework. The chapter contains a description of the syntax and semantics of a query language for service retrieval and it describes a method for building service repositories for approximate service retrieval.
- Approximate retrieval leads to significant complexity challenges. Chapter 5 investigates efficient methods for indexing and query evaluation.
- The research is concluded in Chapter 6, which includes a summary of the results and a review of future directions and open problems.
- Appendixes 7.1 - 7.2 describe the Object-Process Methodology (OPM) and a method for aligning ontologes.

Chapter 2

Background

In this chapter we describe the basic framework of our study and review existing research. We present the notion of service representation and matching, and introduce the mathematical model behind service retrieval. Following this introduction we classify and present existing methods for service retrieval according to several properties.

2.1 Service Representation

We distinguish between two main types of service representation: syntactic representation and semantic representation. Formally, the definition depends on the type of service meta-data description. In syntactic descriptions, service properties are described using a flat set of text labels $\{l_1, l_2, \ldots, l_n\}$, where each label is a textual token. In semantic descriptions, properties are mapped to a complex data structures, which operate within the context of the *Semantic Web*. As these data structures are formal, and more importantly, commonly shared by stakeholders, semantic representation enables common semantics to emerge. For example, if two business parties commonly agree on a single ontology for describing their product information, they can communicate with each other with confidence, using the common vocabulary. We describe the two types of service representation in the following subsections.

2.1.1 Syntactic Service Representation

In syntactic service representation, the relations between service properties are defined, while their meaning is left unspecified. Several XML-based standards [18] ensure the regulation of the discovery and communication between Web services. A WSDL (Web Services Description Language) [25] document describes the interface and communica-

```
<wsdl:definitions targetNamespace="http://math.example.com" name="MathFunctionsDef">
<wsdl:message name="addIntResponse">
        <wsdl:part name="addIntReturn" type="xsd:int" />
</wsdl:message>
<wsdl:message name="addIntRequest">
        <wsdl:part name="a" type="xsd:int" />
        <wsdl:part name="b" type="xsd:int" />
</wsdl:message>
<wsdl:portType name="AddFunction">
        <wsdl:operation name="addInt" parameterOrder="a b">
                <wsdl:input message="impl:addIntRequest" name="addIntRequest" />
                <wsdl:output message="impl:addIntResponse" name="addIntResponse" />
        </wsdl:operation>
</wsdl:portType>
<service name="MathFunctions"/>
</wsdl:definitions>
```

Figure 2.1: An example of a WSDL document

tion protocol of Web services. An example of a WSDL document is presented in Figure 2.1. A WSDL port-type describes the interfaces (legal operations) exposed by a Web service. Each port-type is described as a set of operations (e.g., *AddInt*). Each operation exhibits a set of typed input and output parameters, such as *addIntRequest* and *addIntResponse*. All types are defined in the XML Schema standard [16]. A type can be primitive (e.g., integer - *xsd:integer*) or complex (described by a complex XML schema structure). It is important to emphasize that WSDL describe only the "flat" interface of the operation (what the operation receives and sends), and not the logic behind the operation's functionality (its pre-conditions and post-conditions, for example). The UDDI (Universal Description, Discovery, and Integration) protocol [9] extends the descriptions provided by WSDL, enabling Web services to be published and discovered through keyword search.

Substantial work, both academic and industrial, has been carried out to leverage Web services for enterprise computing. One of the main efforts is Service Oriented Architecture (SOA), which relates the different functional units of applications using Web-services as components which are combined by an orchestration model. SOA enables applications to be dynamically altered and updated according to business needs. The heart of SOA is the language in which the orchestration model is described. Currently, the most prominent SOA language is the Business Process Execution Language for Web Services (BPEL4WS) [76]. This language is an initiative by IBM, BEA, Microsoft and other companies, and is supported by several products, including IBM WebSphere Business Integration Server [27] and Oracle Process Server [28].

SOA is a significant step towards flexible computing, but due to lack of automation

it realizes only some of the potential of Web services. Since services are linked to business processes manually, SOA standards aim to solve mainly communication management problems. For instance, SOA products require XML schemas to be manually mapped to each other. The knowledge represented by these schemas remains isolated and cannot be automatically inferred or shared. Thus, SOA does not address more inherent problems such as the inflexible nature of applications. This drawback is primarily due to lack of semantic definition of Web services and applications. Web services provide syntactic information (such as message types), but they do not expose their actual function and meaning. This makes automatic integration and discovery of Web services a difficult task.

2.1.2 The Semantic Web

Semantic Web services aspire to augment the Web service architecture with additional components that would enable automatic discovery, integration and interoperation of Web services. The corner stone of this semantic Web services are semantic descriptions that allow formal and common description of Web service properties. In this section, we describe the current approach towards semantic Web services, which uses the Semantic Web as a framework for establishing semantic Web services.

Ontology is commonly defined as a **"specification of a conceptualization"** [41]. While the term is borrowed from philosophy, has a narrower definition in the field of Computer Science and AI, representing shared knowledge, described in a formal manner [42]. Ontology enables concepts to be shared among developers and systems, providing for a common vocabulary with defined semantics. The Semantic Web [12] is a recent development that implements the notion of ontologies on the scale of the World Wide Web. The Semantic Web aims to extend the World-Wide-Web by representing data on the Web in a meaningful, formal and machine-interpretable form.

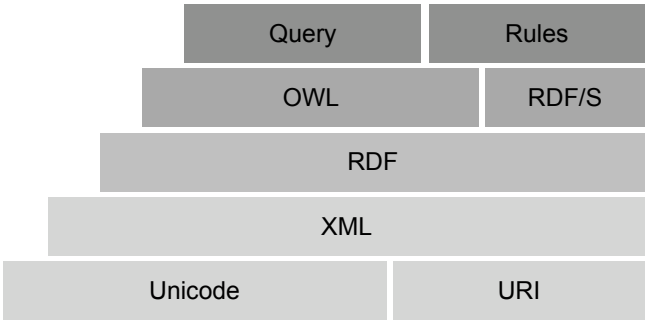


Figure 2.2: Layers of the Semantic Web

The actual implementation of the Semantic Web is a set of languages for formalizing knowledge, sharing common concepts and organizing information. Figure 2.2 depicts the *Semantic Web stack*: the set of languages that forms the Semantic Web. The visualization of the stack is borrowed from Tim Barners-Lee's illustration [10], but includes only the portion which is relevant to this research. The two bottom layers of the stack form the foundation of the Semantic Web, are part of the current World-Wide-Web, and are not unique for the Semantic Web. The lowest level of the stack includes technical standards. **Unicode** [78] is a standard for encoding multilingual characters. The **URI** (Unified Resource Identifier) [11] standard provides a framework for uniquely identifying resources (URLs, which are used to locate Web addresses, are the most common instance of URIs.) XML (Extensible Markup Language) [18] is a standard language for describing the structure of data, and is used as the foundation of Semantic Web languages syntax and type system.

RDF (Resource Description Framework) [54] defines a notation for the representation of information, based on a graph structure, where nodes are resources and edges are relations between them. OWL (Web Ontology Language) [6] and RDF/S (RDF Schema) [19] augment the RDF vocabulary with methods to describe domain knowledge using ontologies. For instance, OWL and RDF/S can be used in order to describe the healthcare domain, as depicted in Figure 2.3. Designed as an ontology definition language, OWL allows greater expressiveness than RDF, mainly with respect to constraint definition and support for reasoning. OWL's utilization of RDF is also expressed in the ability to distribute sub-ontologies and manage them over the Web.

OWL comes in three versions, which differ in their expressiveness: OWL Lite, OWL-DL (Description Logic) and OWL-Full. OWL Lite and OWL DL are basically very expressive description logics with an RDF syntax. They are based on considerable existing body of knowledge driven by description logic research, in particular the decidability and complexity of key inference problems. OWL-Full contains additional constructs, which result in no guarantee of reasoning computability. For this reason, we disregard the constructs of OWL-Full and refer to OWL-Lite and OWL-DL by the term "OWL".

OWL and RDF/S are used as the basis of the top layer of the Semantic Web stack (Figure 2.2), which consists of query and rule languages. This layer is responsible for discovering, inferring, and analyzing semantic information. The standard query language for the Semantic Web is SPARQL [59], which is used to query RDF graph-based data structures using a language similar (in some sense) to SQL. One outcome of this research, languages for querying semantic services, can be considered part of this layer.

As OWL constructs are used extensively in our research, we define a subset of OWL

which we will use throughout the research[1]. We use a model of semantic which has close resemblance to ontology models in the field of description logics, for example in Baader et al. [3] and in Horrocks and Patel-Schneider [44]. However, we eliminated several features of OWL which are not used in service description [75], including value restrictions, concept intersection and concept union. This allowed us to simplify the definition of the ontology semantics. The definition requires a notion of *datatypes*, such as integer, float, string etc. In OWL, datatypes are taken from XML-Schema [16], and are denoted as $\mathcal{D}$.

Definition 1: Ontology. An Ontology $\mathcal{O}$ is a tuple: $\mathcal{O} = \langle \mathcal{C}, I, R, P, F, \mathcal{A} \rangle$, where:

- $\mathcal{C}$ is a set of concept classes.
- I is a set of individuals (also referred to as instances).
- $R : \mathcal{C} \times \mathcal{C}$ is a set of relations between concept classes.
- $P : \mathcal{C} \times label \rightarrow \mathcal{D}$ describes datatype properties by mapping concept classes to datatypes (e.g., integer, string), where a text label defines each property.
- $F : I \rightarrow \mathcal{C}$ defines the set of facts derived from the ontology as a mapping function between instances and concept classes.
- $\mathcal{A}$ is a set of axioms defined on concept classes, relations and properties. The axioms are defined below.

The semantics of an OWL ontology is specified as the assignment of the facts function (F) with relations to the axioms that the ontology includes ($\mathcal{A}$). Given an individual $i \in I$ and a class $C \in \mathcal{C}$, we denote by $i \in C^{\mathcal{I}}$ the notion that i is an instance of C in an interpretation set $\mathcal{I}$. According to the ontology definition, it implies that there exist a mapping $F(i) = C$. Similarly, we define an interpretation set for relations $R^{\mathcal{I}}$ and datatype properties $P^{\mathcal{I}}$.

This simple definition of the relations between instances and concept classes allows us to define the axioms. These axioms form the structure of the ontology. Axioms in this context are specific to each individual ontology. For example, an axiom asserting that *Hospital* is a subclass of *Organization* results in the condition that the interpretation set of *Hospital* (all the instances which belong to *Hospital*$^{\mathcal{I}}$) must be a subset of the interpretation set of *Organization*. Therefore, if *Mount Carmel Hospital* is an instance of *Hospital*, it is also an instance of *Organization*. If the concept *Hospital* has some relation to another concept (e.g., Ward), then the relation between the individual *Mount Carmel*

[1]For a formal definition of the syntax and semantics of OWL, the kind reader is referred to [6].

Hospital and some individual of *Ward* holds. An ontology $\mathcal{O}$ is considered valid if it follows the ontology axioms, which are derived from a set of constructs that are used in order to express semantic relations between ontology entities.

Construct Name	**Syntax**	**Semantics**
A concept class	C	$C^{\mathcal{I}}$
An instance i of C	$i : C$	$i \in C^{\mathcal{I}}$
Concept subclass	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
Class Negation	$\neg C_1$	$\mathcal{O}^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$
Equivalent class	$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
Datatype property	$C.p(l)$	$\{\forall i \in C^{\mathcal{I}} \mid i.p(l) \in P^{\mathcal{I}}\}$
Object property (relation)	$C_1 \rightarrow C_2$	$\{\forall i_1 \in C_1^{\mathcal{I}}, i_2 \in C_2^{\mathcal{I}} \mid (i_1, i_2) \in R^{\mathcal{I}}\}$
At least restriction	$C_1 \rightarrow_{(\geqslant n)} C_2$	$\{\forall i \in C_1^{\mathcal{I}} \mid \exists i_k \in C_2^{\mathcal{I}}\ , \#\{(i, i_k) \in R^{\mathcal{I}}\} \geqslant n\}$
At most restriction	$C_1 \rightarrow_{(\leqslant m)} C_2$	$\{\forall i \in C_1^{\mathcal{I}} \mid \exists i_k \in C_2^{\mathcal{I}}\ , \#\{(i, i_k) \in R^{\mathcal{I}}\} \leqslant m\}$

Table 2.1: Abbreviated syntax and semantics of OWL

Table 2.1 defines the semantics of ontology constructs according to the effect their axioms have on the interpretation of an ontology. We define by $\mathcal{O}^{\mathcal{I}}$ the set of all possible interpretations of the ontology (all possible concept instance assignments). The first column states the name of the constructor. The second column provides the syntax of the mathematical notation of the construct. We would use this mathematical notation throughout the research. The third column defines the semantics of the construct by describing the interpretation set which is the result of applying the construct. The notation $\#\{\ldots\}$ denotes the number of elements of a set.

In order to exemplify the different ontology constructs, we use a graphical representation of the ontology structure. Figure 2.3 depicts an example of an ontology for the healthcare domain. The graphical visualization is based on OPM (Object-Process Methodology) [32], which includes a graphical language for system specification. OPM is described in Appendix 7.1. The relations between OPM and OWL were described in [33]. The legend of the diagram includes the visual method to describe constructs: subclass, instance, datatype property and object property (relations). The different elements of the diagram are as follows:

- Concept classes (e.g., Hospital, Patient).
- Instances (e.g., Mount Carmel Hospital, Mount Sinai Hospital). In the mathematical notation, the same expression would be written as:
  $Mount\,Carmel\,Hospital : Hospital.$

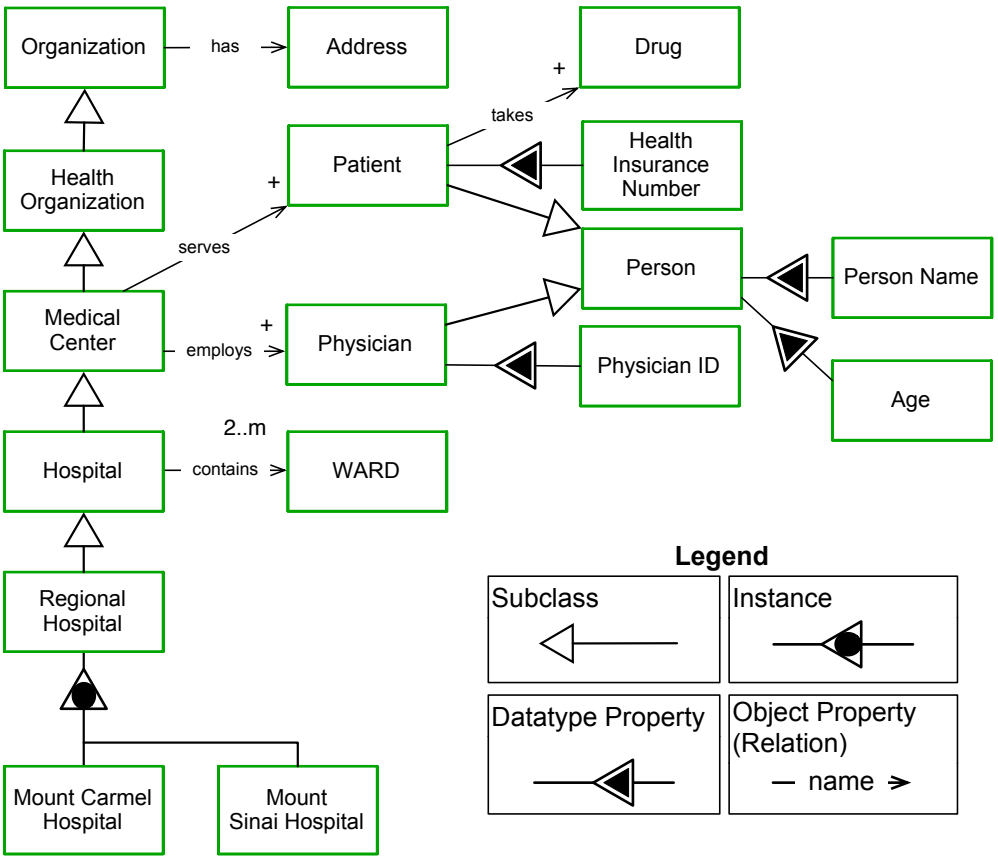


Figure 2.3: An example of healthcare domain ontology

- Datatype properties, which define labeled value frames for concepts (e.g., patient has an health insurance number). Datatype properties are atomic, i.e., they do not refer to any additional elements.
- Object properties, which define relations between classes (e.g., patient takes drug).
- Subclass relations, which define generalization relations between concept classes (e.g., Hospital is a Medical Center and Patient is a Person). In the mathematical notation, the same expression would be written as $Hospital \sqsubseteq Medical\,Center$.
- Instance relations, which define the interpretation sets derived from the ontology (e.g., Mount Carmel Hospital is classified to the concept class hospital). The diagram does not notate the **indirect** interpretations which can be inferred from the ontology (e.g., Mount Sinai Hospital is an Organization).

- Cardinality participation constraints, which define the number restriction (al least and at most). The constraints are represented as small tags near the end of the object properties. They follow OPM's convention which is described in Table 7.1 in Appendix 7.1. For example, the label $2..m$ is interpreted as a relation which connects at least 2 instances, with no upper boundary. The label $+$ is interpreted as a relation which connects at least 1 instance with no upper boundary. If no tag is presented, then the default cardinality takes place, in which the instance must be related to a single instance.

2.1.3 Semantic Service Representation

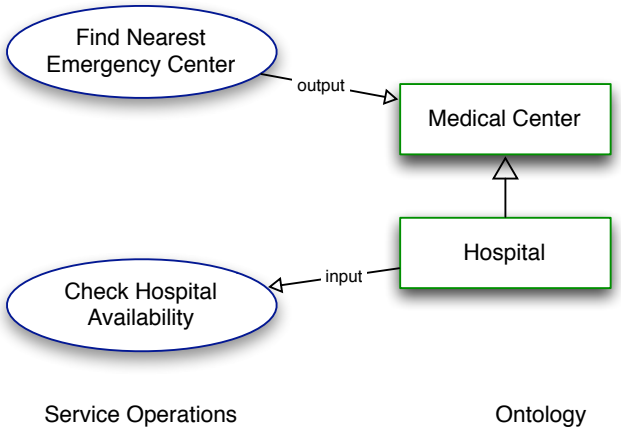


Figure 2.4: An example of Semantic Web Services

Semantic Web services are Web services that are formally described using ontologies. Semantic Web services annotation languages enable service properties to be formalized by mapping properties to ontology concepts, as depicted in Figure 2.4. Several semantic annotation languages for Web services exist. These include OWL-S (OWL-Services) [2], WSML (Web Service Modeling Language) [29] and WSDL-S/SAWSDL [1]. The languages vary in their expressibility, modus operandi and approach. While OWL-S and WSML provide a thorough process definition (similar to that of BPEL4WS), WSDL-S is more lightweight and only adds semantic operation mapping to WSDL.

In this work, we refer mainly to OWL-S because of its wide acceptance in the research community and its tight integration with OWL and the Semantic Web. An OWL-S ontology includes three sections:

1. The Service Profile, which describes the properties of a service that are viewable

externally, including categories, input/output interfaces, expected users etc.

2. The Process Model, which defines the control flow of the service by representing the active components of the service as processes. The process model define for each process its inputs, outputs, effects and preconditions. It also specifies the execution flow of the processes. There are three types of processes: atomic processes, which are executed in a single call, and composite processes, which comprise other processes. A composite process is executed according to a control construct type, such as parallel execution, sequential execution, or conditional execution. Each process has its own property specification, similarly to the service profile specification.

3. The Service Grounding, which links the process model to communication-level protocols. Specifically, it links atomic processes to WSDL-defined operations.

2.2 Approaches to Service Retrieval

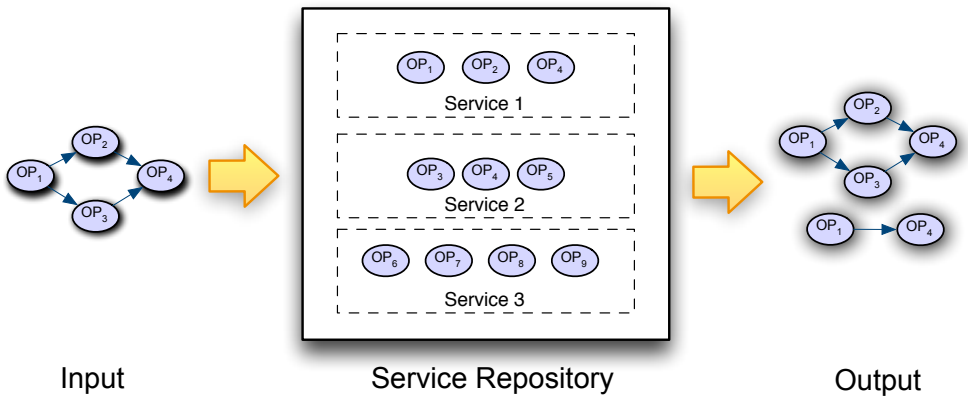


Figure 2.5: The service retrieval model

Service retrieval is the process of matching a query to a set of existing services. It incorporates two stages: *service discovery* and *service composition*. Service discovery is the process of selecting appropriate Web services based on their match to a given query (sometimes referred to as "matchmaking"). Service composition is an addition to service discovery, in which the result might contain operations from different services. As a major subset of existing research approaches cover only one of the stages, we present a single model of service retrieval, which incorporates both service discovery and service composition. In our model, which is depicted in Figure 2.5, a query is evaluated against a

repository of Web services, and returns a result set as an output. The two stages of service retrieval can be categorized according to the expressibility of their queries, the type of the repository and the type of the result set, as well as their algorithms, performance, etc.

2.2.1 Classification of Approaches

Having defined the notion of service retrieval, we can characterize retrieval approaches and investigate how they differ from each other. Table 2.2 identifies several categories of service retrieval, which are based on two dimensions: matching process and matching resulution:

- Matching process: We distinguish between *semantic-based* and *syntactic-based matching* of services. In the latter case, the matching process is based on matching the textual properties of the service representation with the query. The semantic-based matching can be further divided into three categories:
  - *Logic inference retrieval*.
  - *non-logic retrieval*.
  - *hybrid retrieval*, which combines logic and non-logic retrieval.
- Matching resolution: The granularity of the service descriptions may be:
  - *Black-box*: The matching process considers solely the interface of services. In this case, queries and results are often simple, containing graphs with a single operation.
  - *White-box*: The inner structure of each service in the service repository is considered. In this case, queries and results are often complex, and are defined as graphs with several operations.
  - *Composition*: The retrieval process might mix operations from different services, resulting in the creation of a new artifact.

The rest of this section discusses the different categories of service retrieval. Note that due to the enormous amount of papers and tools, this review only includes representative members of the categories. An extensive review is provided in [49].

2.2.2 Semantic Approaches

In this section we review semantic approaches to service retrieval. In this family of retrieval methods, the input and the repository of the process are semantically annotated using languages described in Section such 2.1.3 as OWL-S.

	Semantic			Syntactic
	Logic	**Non-Logic**	**Hybrid**	
Black-box	OWLS-UDDI [58] Inter-OWL-S [66] MAMAS [57]	Klein [48] HotBlu [26] Hau et al. [43]	LARKS [67] OWLS-MX [50] RDQL [14]	UDDI [9] Woogle [31]
White-box	Mediator [74] Bansal and Vidal [5]			RE-Search [65] BP-QL [7] Matrix-match [4]
Composition	SHOP2 [77] OWLS-XPlan [51] State-Comp [71]	DIANE [52]		

Table 2.2: Categories of service retrieval

Logic-based Approaches

Logic-based Approaches are based on reasoning over the ontological descriptions of Web services. Matchmakers, such as OWLS-UDDI [58] by Paolucci et al. and Inter-OWL-S [66] by Sirin et al., perform matching according to OWL-S profile properties. The OWL-S profile ontology describes the capabilities of services according to their input, output, effect and precondition properties in a *black-box* mode, where the inner procedure of the services is not taken into account.

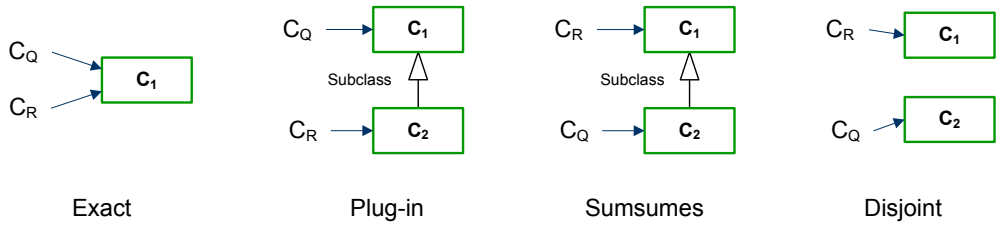


Figure 2.6: Categories of logic matching degrees

Logic-based methods classify matching degrees into four categories: *exact*, *plugin*, *subsumes* and *disjoint*. These categories were originally defined by Zaremski and Wing in 1996 [79]. As these categories are used as general means for comparison, we define them accurately. Let $C_Q = \{C_1, C_2, \ldots, C_n\}$ be the set of concepts related to a given query and $C_R = \{C_1, C_2, \ldots, C_m\}$ a set of concepts related to a given result. Figure 2.6 describes the four degrees graphically, in the simplified case where there is a single concept for either the query and the result. In each of the example, C_R is more specific concept than C_Q (for example, C_Q is a person and C_R is a patient).

Exact Exact matching is defined as perfect identification of the query properties (e.g., input) with the result properties, such that $C_Q \equiv C_R$.

Plugin The concepts of the result are fully contained in the concepts of the query, such that $C_R \sqsubseteq C_Q$. In other words, at least one concept in the query has a concept in the result which is more specific. For example, the user looked for a service that finds the name of a person and received a service that returns the name of a patient. As all patients are persons, the result does not contradict the axioms of the query.

Subsumes The concepts of the query are fully contained in the concepts of the result, such that $C_R \sqsupseteq C_Q$. In this case some of the results may not fully answer the query in the sense of set-theory. For example, the user looked for a service that finds a name of a patient in a given hospital, and received as a result a service that returns the names of all the persons in the hospital.

Disjoint None of the concepts of the result can be related to any concept of the query, such that $C_R \sqcap C_Q = \emptyset$, implying that the result is not applicable to the query.

Logic-based **white-box** retrieval, in which the process model is exploited as a mean for retrieval, is relatively new. One example is Mediator [74], where OWL-S service process models are mapped into equivalent logical statements that are then evaluated by a model checker. Another example is given by Bansal and Vidal [5] who developed a matchmaker that follows the process model of OWL-S [2], matching control constructs as well as operations. The two methods suffer from a common drawback, which is limited recall, i.e., they are prone to return results with low recall. As the final result depends on the success of all the matchings, the chances for a successful retrieval with low-recall is even lower.

Non-logic Methods

Non-logic based matching use a variety of methods, including text similarity, structured graph matching, or numeric concept distance computations over ontologies. They differ from syntactic methods, as they still use the ontology as a resource for matching.

A matchmaker by Klein and Bernstein [48] combines signature matching (by matching input/output parameters) and specification matching (by matching precondition/effect parameters). Each signature and specification of both the query and the repository services are transformed to a single graph-based structure, which is used for the matching process. Constantinescu and Faltings [26] use a numeric method in order to encode service prop-

erties. Their primary motivation was to use the numeric representation for efficient type matching through a numeric distance function.

The research of non-logic methods is rather preliminary, but it has yielded some interesting results. Constantinescu and Faltings use a numeric distance measure in order to provide an efficient inference process without losing the semantics of the logic-inference process [26]. Numeric and graph-based methods can provide higher expressibility and approximation than logic inference. For example, Hau et al. [43] define similarity metric between services according to the inferencibility of each OWL service description construct. Their method provides a softer measure for similarity while keeping a formal and explainable similarity framework. The set of relaxations is based purely on concept hierarchies.

Hybrid Approaches

Hybrid matchmakers combine logic-based methods with other matching methods, especially from the information retrieval (IR) field. LARKS [67] by Sycara et al. introduces similarity matching and type signature matching along with logic-based matching of concept classes. Type matching by examining the XML Schema types [16] provides another measure in addition to concept class matching.

Klusch et al. experimentally evaluated the contribution of hybrid matchmakers in OWLS-MX [50]. They compare pure logic matching, syntactic information retrieval, and hybrid methods that combine the two. Their findings show that hybrid approaches outperform both logic-based approaches and pure IR methods. Another hybrid matchmaker is RDQL by Bernstein and Kiefer [14], which is based on extending SPARQL with imprecise query constructs. The extension includes syntactic similarity metrics, such as token-based comparison, TF/IDF and the Levenshtein metric [15]. User-defined threshold specifies the extent of relaxation.

Hybrid approaches have two drawbacks, the most serious one being their explainability. As they use methods taken from text and string analysis (such as TF/IDF), they lack the ability to explain *why* one service was preferred over another service in the matching process. Explainability is an attractive feature of logic based methods, as inference can be used to prove (or disprove) to the user that a selection is valid. This feature enables the user of applications such as automated service composition that require high confidence, which can be provided by of logical support to the composition. Therefore, This type of application cannot be based on hybrid methods. Furthermore, as the hybrid approach use a multitude of matching methods, finding the root basis for a matching decision is often difficult.

2.2.3 Syntactic Approaches

In syntactic matching, the operations are described using a simple set of labels $l_1, l_2, \ldots, l_n$, rather than a structured ontology. In an abstract manner, UDDI [9], the industry standard for locating Web services through keyword and category search, is based on this model. The main drawback of UDDI and with other keyword-based approaches is lack of sufficient information for describing Web services. Web service interfaces are defined using WSDL descriptions, which contain limited information regarding Web service operations. Therefore, keyword search solutions fail in providing satisfactory recall for Web service search [2, 66]. Woogle [31] aims at confronting this problem by using text clustering techniques. Text-based methods can be easily deployed on large cortexts of services, but they fail when exact matching is required. For example, when querying a Web services search engine, such as Woogle, using the term **"book"**, results belonging to two different domains will be retrieved: the publication domain, where the word *book* defines a printed text, and the travel domain, where the word *book* is used in the context of booking a flight.

While syntactic methods fall short in black-box discovery, they demonstate interesting results in white-box discovery, which focuses on the analysis of process models. BP-QL [7, 8] is a query language for BPEL4WS [76] 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. Shen et al. [65] 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. Bae et al. [4] suggest a similar matching strategy, where processes are represented as matrices. These methods differ from our approach in two main aspects. First, they evaluate a query against each service independently. The result is that potential compositions that might be created from operations in different services are not considered. Second, these methods provide limited support for approximate matching and do not rank results according to their semantic and compositional certainty.

2.2.4 Service Composition

In Web service composition, several operations from possible different services are bundled together to form a new service. Service composition raises additional challenges on top of the challenges posed by service discovery, which is evising an execution plan which satisfies the given query. Most composition planners are based on the classical state-based AI planning paradigm, in which queries and operations describe their effect on the world

state and their preconditions. In OWL-S and WSML (but not in WSDL-S/SAWSDL), these elements can be specified using the effects/precondition elements within the profile or process models. 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. OWL-S based planners include SHOP2 [77], GOLOG-SCP [56], State-Comp [71], and OWLS-XPlan [51].

One limitation of service composition is mutual semantic correspondence: the separation between service discovery and composition. In service composition the semantic properties of operations must correspond to each other, as well as to the query. However, most composition planners effectively use very simple semantic annotation and tend to ignore this problem. A counter example to this rule is the DIANE [52] matchmaker, which combines operation interface matching with composition planning. DIANE uses logic-based methods, in particularly the plug-in/subsumes matching strategy for operation matching. Therefore, the approach exhibits low recall, which limits the amount of feasible compositions.

Chapter 3

Approximate Service Similarity

The motivation of this chapter is to propose a schema for ontology-based approximate retrieval including discovery and composition of Web services. The basic concepts of retrieval are *services*, which are the data entities to be searched, *operations*, which are the basic elements for composition, and *compositions*, which are the results of the retrieval process. We formally define these concepts in Section 3.1. In Section 3.2 we address the challenge of approximate service retrieval by defining approximation in service retrieval, unifying two different concepts of composition variability: semantic and functional. In order to achieve this goal we propose *virtual operations* as a general model for measuring the affinity between two compositions. The model estimates the number of operations required to bridge the gap between the compositions. The types of feasible virtual operations define the different aspects of similarities between compositions. We express them using a small set of *affinity patterns*, which formally define each construction types and assign to them a given certainty.

The affinity patterns are evaluated for relevance and correctness for human subjects. Section 3.3 describes the experiment we carried out to evaluate the affinity patterns. Our main finding is that the notion of similarity used by logic-based service discovery approaches (descibed in Section 2.2.2) is not compatible with the perception of human subjects. Rather, they employ a much broader concept of similarity, which includes elements such as relations between services and similarities between instance sets. Furthermore, human subjects evaluate similarities in a softer form than prior approaches have predicted.

3.1 Definitions

3.1.1 Services and Operations

The basic components of service retrieval are *operations* and *services*. We formally define an operation as follows:

Definition 2: Operation (OP). An operation is a tuple $OP \equiv \langle Props, label \rangle$, such that

- $Props$ is a set of properties, $p_1, p_2, \ldots, p_n$. Each property is taken from a single type domain: $p_i \in$ Input | Output | Effect | Provider | Def | ...
- $label : Props \rightarrow \mathcal{O}$ is a labeling relation that associates a property from the set $Props$ with a concept taken from an ontology $\mathcal{O}$ (as defined in Section 2.1.2). There might be several concepts for each property.

Operations are basically a mapping between the set of properties and an ontology, (denoted by $\mathcal{O}$), to which operation parameters refer. We define an *operation* as a specification of an atomic function, performing an atomic task, which is not further divided. Operations are defined using an unbounded set of properties, specifying meta-data about the functionality of the operation. The operation model allows an unbounded number of properties, including inputs, outputs, effects, and definitions. Other types of properties can be assigned, reflecting organizational concerns (e.g., department, provider) or qualitative concerns (e.g., price, quality of service). Our definition of an operation is similar to the definition of an atomic process in OWL-S (see Section 2.1.3), and is based on the syntactic definition of a WSDL operation (see Section 2.1.1).

In the same manner as in WSDL, a service is defined as a set of operations. The specification of services does not convey any information about the execution order of the operations. We ignore the notion of port-type, which reflects communication protocol aspects, which go beyond the scope of this research. We assume that each service is provided by a single *provider*. We define a service as follows:

Definition 3: Service. A service, S, is a set of operations:
$S = \{OP_1, OP_2, \ldots, OP_n\}$, provided by a single provider.

3.1.2 Compositions

A central concept in this work is *composition*, which is used as the basic model behind both queries and results. It is defined as a directed graph, where vertices represent oper-

ations and edges represent data flows. The composition captures information about how data flows from one operation to another.

Definition 4: Composition (Com). A composition is a directe and connected graph $Com \equiv \langle \mathcal{OP}, Flows \rangle$, where:

- $\mathcal{OP} = \{OP_1, OP_2, \ldots, OP_n\}$ is a set of operations.
- $Flows \subseteq OP \times OP$ is a set of directed data flows between operations.

A data flow represents a (possibly empty) set of data items which are passed from one operation to another. In order for an operation to operate, all the operations which are connected to it must be executed beforehand. Note that this definition has some common ground with another widespread type of process definition, which is the hierarchical execution ordering of operations. We chose to define compositions according to their data flow dependencies for three reasons:

1. Dependencies are more general than execution order (data dependency induces execution order dependency, but not vice versa).
2. Dependencies are the widespread approach in service composition research (see [4, 65]).
3. Dependencies are more suitable to our method of static service dependency analysis.

3.2 Service Similarity Patterns

We begin by defining *service affinity*, which describes the resemblance between two compositions. We wish to define a general metric, which will cover the two aspects that differentiate between two compositions:

- **Semantic affinity**: How the operations match each other with respect to their semantic properties.
- **Functional affinity**: How the structure of the composition, which defines its operational procedure, match each other.

In Section 3.2.1 we present the notion of *virtual operations*, the mechanism for defining the affinity schema. We then define a distance metric that quantifies the semantic and functional affinities. In Sections 3.2.3 and 3.2.4 we respectively present semantic and

functional affinity patterns. These patterns form our hypothesis for approximate retrieval. The hypothesis is evaluated in Section 3.3.

3.2.1 Adjustment-Based Affinity

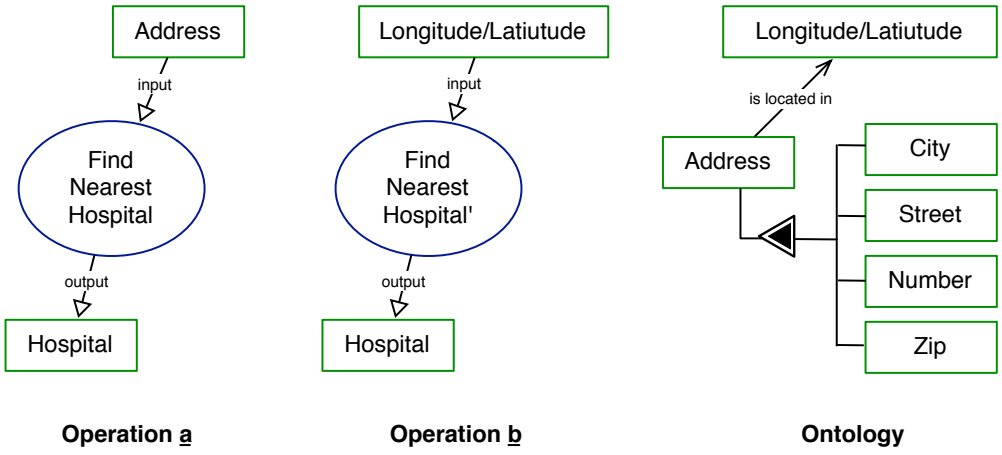


Figure 3.1: An example for affinity between operations

In this section we define a general model for analyzing and measuring the affinity between any two compositions. Consider the following example, which presents two operations that can potentially answer the query presented in Example 1 ("*Find a set of services that accept an address and return the directions to the closest hospital*".).

Example 2: Similar Operations. Two operations, *a* and *b*, differ only in their input parameter:

Operation *a* The operation *Find Nearest Hospital* receives an *address* as an input and returns a *Hospital* an an output.

Operation *b* The operation *Find Nearest Hospital'* receives a *Longitude/Latitude* object as an input and returns a *Hospital* an an output.

The two operations can be used in order to implement a partial set of requirements set in Example 1. Figure 3.1 depicts the two operations, as well as the (partial) ontology that the input concepts of the two operations are mapped to. Clearly, operation *a* is more suitable for answering the requirements, as the concept *Address* is common to both the query and the operation. However, what if the service repository contains only operation *b*? In

that case, a service retrieval framework should return operation *b*. A human engineer can **adjust** operation *b* to her needs. Note that in the ontology presented in Figure 3.2, the two input parameters, *Address* and *Longitude/Latitude*, are related through an object property relation. According to the current service retrieval approaches, the two concepts do not exhibit any similarity, as they are not related through a generalization hierarchy. However, when observing the question of similarity in the context of engineering, it is clear that a human engineer could find both operations useful. We are interested in describing a model that would predict if a given composition can be interchangeable with another composition, and if so, to what extent.

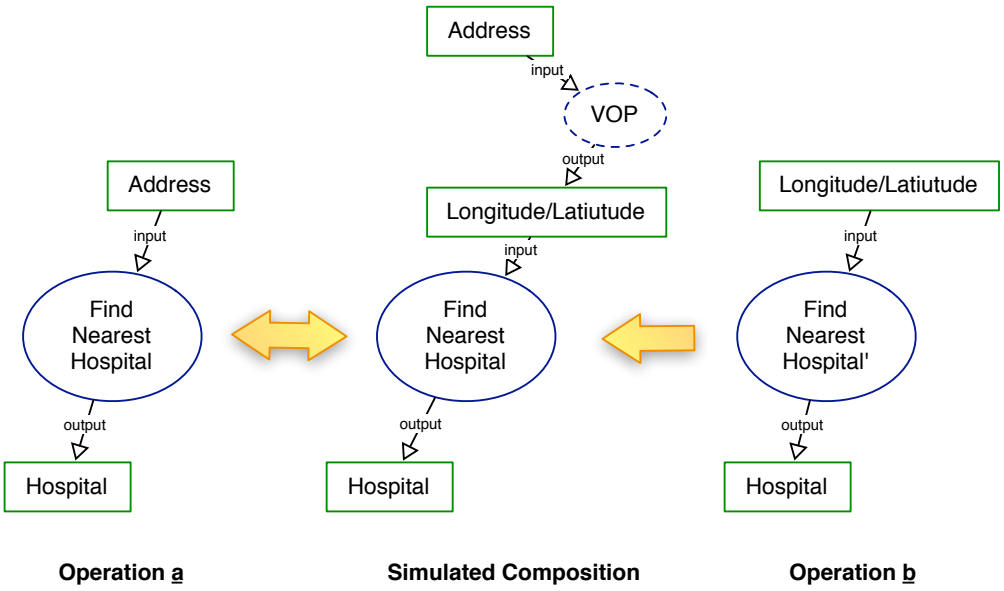


Figure 3.2: Operation *a* is simulated using operation *b*

Our model is based on the notion of simulating compositions. Given two compositions[1], Com_1 and Com_2, we say that Com_1 simulates Com_2 when Com_1 is augmented with a set of operations which imitate the functionality of Com_2. The augmenting operations are called **virtual operations**, as they reflect desired functionality, which can be inferred, but does not necessarily exist. Figure 3.2 visualizes an example of a composition simulation. The simulated composition (in the middle) simulates operation *a* using operation *b*. The virtual operation keeps the interface of operation *a*, while embedding operation *b* with the addition of a virtual operation denoted by *VOP*, which stands for

[1]This definition holds for operations as well. Operations are considered a simple subset of compositions, in which there is a single operation.

virtual operation. It is also marked with a dotted line.

The measure of similarity depends on the extent to which a simulated composition is useful. For example, the similarity between operation *a* and operation *b* is a measure that depends on the difference between the simulated composition and the original composition. Each virtual operation construction may result in different type of uncertainty regarding its reliability. The variations stem from the type of the construction and the properties of each specific construction. In this section we present a model for evaluating the amount of approximation provided by augmenting an existing composition with virtual operations.

The model quantifies the loss of information caused by using virtual operations. We call this model the **lazy programmer model**. As its name hints, we imagine a programmer whose task is to build a system using approximated matching and the ability to do some manual coding. The programmer is given a system specification, represented by a query, and a search engine result, which is intended to implement the requirements. The match between the query and the result could be:

- Perfect, if the result answers the system specification perfectly, the programmer is satisfied as implementing the system requires no effort at all.
- Partial, if the result does not fits the query perfectly, but can somehow be altered in order to answer the query.
- Irrelevant, if the result cannot satisfy the requirements, no matter how many virtual operations are added. In which case, the programmer would consider the result unusable.

Let us formally define our concept of similarity. Given two compositions, Com_1 and Com_2, we define *similarity* between them as a function on the number (and properties) of the virtual operations which are required to *simulate* Com_1 *using* Com_2.

Definition 5: Simulated Composition. Given two compositions, Com_1 and Com_2, the simulated composition Com^v is defined of Com_1 using Com_2, a set of virtual operations $\{VOP_1, VOP_2, \ldots, VOP_n\}$ and data flows $\{F_1, F_2, \ldots, F_n\}$ between the virtual operations and to/from the original composition (Com_2):

$$Com^v = Com_2 \cup \{VOP_1, VOP_2, \ldots, VOP_n\} \cup \{F_1, F_2, \ldots, F_n\}$$

Let us see how this definition is used. Given the operations in Example 2, we can set Com_1 to be operation *a* and Com_2 to be operation *b*. The simulated composition in Figure 3.2 is an example for Com^v. The set of virtual operations includes a single operation, $\{VOP\}$. The set of flows is empty. This notion of simulated composition is fairly general, and does not pause too many restriction on the way simulated compositions are built. There is a set of possible simulated composition based on any two compositions, and not a single one.

3.2.2 Virtual Operations

The possibility of constructing a virtual operation depends on inferencing it from the ontology, or from other sources. Consider the ontology model in Figure 3.1. As there is a 1:1 relation between *Address* and *Longitude/Latitude*, it seems reasonable that a mapping is possible, and an operation such as VOP can be constructed. In this section we define a schema for evaluating the constructibility of virtual operations. We also analyze the situations in which virtual operations can be constructed. The basic data structure of a virtual operation is identical to the the definition of an operation (Definition 2). It is the outcome of the construction process, described below. Virtual operations are created by a *constructor*, a function that maps the two comparable compositions to a set of virtual operations, as defined below.

Definition 6: Constructor. A constructor is a function:

$$\Psi : Com \times Com \rightarrow Com^v$$

Each constructor is parameterized by two properties:

- $type : \Psi \rightarrow \{\text{Subclass}, \text{Relation}, \ldots\}$ is a type function that assigns each constructor a finite set of types, which are specified below.
- $\mu : \Psi \rightarrow [0, 1]$ represents the certainty of the construction.

The definition of the constructor frames its context: it creates a set of virtual operations and flows, given two existing compositions. Note that these compositions can include a single operation, as well as a fully fledged composition. The function μ represents the certainty of constructing the virtual operation on the loss of information in the construction process. If the construction yields virtual operation with good preservation, then the certainty would be high.

We define a set of axioms regarding constructors:

Definition 7: Constructor Axioms. Given a constructor Ψ, the following holds:

1. The identity construction: $\Psi(Com, Com) = \emptyset$. Executing the construction on two identical compositions would result in an empty construction, which returns an empty set of operations and flows.

2. Commutativity: $\Psi(Com_1, Com_2) = \Psi(Com_2, Com_1)$. The constructor function is insensitive to the order of the construction.

We have segmented the possible constructors of virtual operations to a small number of cases, which reflect possible inference over the ontology and the structure of the composition. In the following sub-sections, we define constructor types, each of them specify a pattern for a given relationship between operations. The constructors are grouped in two groups, according to the aspect they rely upon:

1. Semantic constructors, which are based on semantic properties.

2. Functional constructors, which are based on the structure of the composition graph.

3.2.3 Semantic Affinity Patterns

In this section we define a family of semantic affinity patterns. These patterns define similarity measures of the conceptual basis of the operations. For the sake of simplicity we say that these patterns are defined on compositions that include a single operation. Each of the patterns represent a specific relation between two concepts in a given ontology. We assume that there is a single ontology, $\mathcal{O}$, which is a unification of all known ontologies in the service retrieval framework. The method for unifying the ontologies is described in Appendix 7.2.

The main challenge in designing the pattern is to provide a single frame of reference which is adaptable to all the relations existing in an ontology. These relations, which are described in Section 2.1.2, are very different in nature. They include:

- Set relations (subclass),

- relations between concepts (known in OWL as object properties), and

- instance classification relations.

We have decided to base our pattern approach on relational similarity rather than attribute similarity. An attribute is a characteristic of an entity, whereas a relation is a connection between two or more entities. Formally, an attribute is a predicate with one

argument and a relation is a predicate with two or more arguments. Relations are powerful for representing information, because all attributes can be represented by relations, but not vice versa [72]. For example, the age of a person may be seen as an attribute, $Age(Person)$, or as relations: $OlderThan(Person_1, Person_2)$ and $HasAge(Person, 50)$.

Using the general quality of relations, we assign an abstract relation between two concepts as the building block of the semantic affinity patterns. We denote the relation by $R(C_1, C_2)$, of the type $R : \mathcal{O} \rightarrow \mathcal{O}$. We assign a cardinality pair for each relation as a pair of real numbers: $n : R \rightarrow N_\perp \times N_\top$. The cardinality pair defines the bottom and top boundaries for relations between the instances of the concepts. For example, given a relation $R(C_1, C_2)$, if an instance of C_1 can be related to either 1, 2, or 3 instances of C_2, then we would define $n(R(C_1, C_2)) = (1, 3)$. The cardinality pairs are used in order to define the certainty value of each pattern construction. A cardinality pair must preserve the following properties:

- $N \in [0, \infty]$ - The cardinality boundaries range from 0 to infinity.
- $N_\perp \leqslant N_\top$ - the bottom boundary cannot be larger than the top boundary.
- If $n(R(C_1, C_2)) = (N_\perp, N_\top)$ and $N_\top = 0$ then $\nexists R(C_1, C_2)$. A relation in which the top boundary is equal to 0 is not valid.

We base the construction certainty on the cardinality pair, as an extension of a well-known information based measure. Resnik [62] associates probability p with concepts in an ontology subclass hierarchy to denote the likelihood of encountering an instance of a concept C. If $C_1 \sqsubseteq C_2$ then $p(C_1) < p(C_2)$. The information content of a concept C is then dened as function over the probability of its instance likelihood. Hau et al. [43] extend this notion to semantic Web service similarity, by defining the information carried by each concept as its set of properties, and by comparing the sets. In this work, we use methods based on comparing relations, rather than on the properties. α_i is the pattern certainty coefficient, which distinguish between the certainty of different patterns, and is bounded by 1, such that $0 \leqslant \alpha_i \leqslant 1$. These differences were shown in our experimental results (see Section 3.3.2). We define by g_i the average gradient of the similarity curve of a pattern$i$, and define α_i as: $\alpha_i = 1 - g_i$. We define the construction certainty, $\mu(\Psi_i)$, as follows:

$$\mu(\Psi_i) = \alpha_i \frac{1}{1 + N_\top - N_\perp}$$

The definition conforms to the schema of Resnik [62], as the probability of finding a related instance depends on the cardinality of the relation. For instance, if the cardinality of

the relation is $(0, 1)$, then finding the related instance is deterministic. However, assuming uniform distribution, finding the related instance when the probability is $(0, N)$ is $\frac{1}{N}$. As we cannot assume any type of distribution, we assume the worst case of the uniform distribution. Therefore, when the top boundary is unbounded the certainty is 0:

$$\lim_{N \to \infty} \mu(\Psi_i) \to 0$$

Set Hierarchy Pattern

The set hierarchy pattern defines how virtual operations can be constructed to define an affinity between two operations, where the parameters concept sets are related by set relation. As Figure 3.3 shows, an operation which has an input concept of *Person* can be transformed to another operation with an input of the subclass concept concept *Patient*.

We define two concepts, C_{sup} and C_{sub}, where C_{sub} is a subclass of C_{sup}, such that $C_{sub} \sqsubseteq C_{sup}$. For example, in Figure 3.3, C_{sup} can be the concept *Person* and C_{sub} the concept *Patient*. With no loss of generality, we define C_{sup} to be the input concept of the original composition. In order to simulate a composition that takes a concept of C_{sub} as input, the constructor creates a new virtual operation, denoted by VOP, which takes a C_{sup} as input and returns an output of C_{sub}. The construction is based on removing properties from the super-class. $P(C_i)$ is the set of properties concept C_i exhibit (both datatype properties and object proeprties). For the sake of simplicity, we define the constructor for a single input concept. However, the construction is identical for a set of concepts.

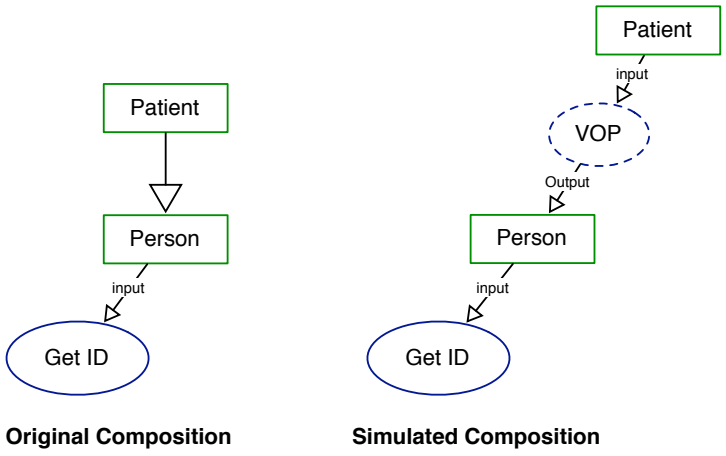


Figure 3.3: Constructing virtual operations according to the set hierarchy pattern

Constructor 1: Set Hierarchy Pattern.

$$\begin{aligned} VOP.input &= C_{sub} \\ VOP.output &= P(C_{sub} \setminus C_{sup}) \end{aligned}$$

the constructor for the opposite case, where the original composition takes C_{sub} as a parameter, is identical because when the input is C_{sup} then $P(C_{sub} \setminus C_{sup}) = P(C_{sup})$.

$R(C_1, C_2)$ is an abstract relation between two concepts C_1 and C_2, which are related by a subclass relation. We define the cardinality pair of the relation as follows:

$$\begin{aligned} &\text{Bottom boundary:} \quad N_{\perp} = 0 \\ &\text{Top boundary:} \quad N_{\top} = \frac{|P(C_1) \cup P(C_2)|}{|P(C_1) \cap P(C_2)|} \end{aligned}$$

For example, consider the ontology described in Figure 2.3. Let us look at two concepts related through a set of two subclass relations: *Organization* and *Medical Center*. The property sets of the concepts are:

$$\begin{aligned} P(\text{Organization}) &= \{\text{Address}\} \\ P(\text{Medical Center}) &= \{\text{Patient}, \text{Physician}, \text{Address}\} \end{aligned}$$

The top cardinality is calculated as the ration between the union and the intersection of the property sets:

$$N_{\top} = \frac{|\{\text{Patient}, \text{Physician}, \text{Address}\}|}{|\{\text{Address}\}|} = 3$$

Therefore, we can calculate the cardinality pair of the relation, as:

$$n(R(\text{Organization}, \text{Medical Center})) = (0, 3)$$

The certainty of the construction is:

$$\begin{aligned} \mu(\Psi_{set}) &= \alpha_{set} \cdot \tfrac{1}{1+3-0} \\ &= \alpha \cdot \tfrac{1}{4} \end{aligned}$$

Where α_{set} is the pattern coefficient.

Relation Pattern

The relation pattern allows constructing virtual operations on the basis of OWL object properties. We define two concepts, C_s and C_d, which are related through relation R, such that $R(C_s) = C_d$. The virtual operation maps between the input concept and the

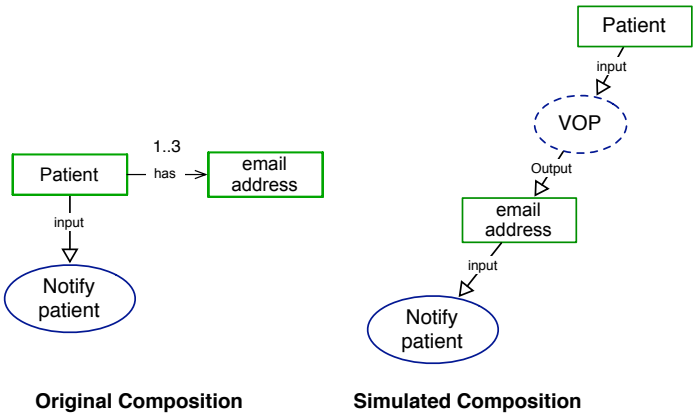


Figure 3.4: Constructing virtual operations according to the relation pattern

related output concept.

Constructor 2: Relation Pattern.

$$\begin{aligned} VOP.input &= C_s \\ VOP.output &= \{x : C_d \,|\, x \in R(C_s)\} \end{aligned}$$

The relation pattern is exemplified in figure 3.4, where $C_s = (Patient)$, $C_d = \textit{Drug}$ and $R = \textit{takes}$. The virtual operation takes as input an instance of the type *Patient* and returns an instance of the *Drug* concept. As most patients take more than a single drug, the average cardinality of the relation is higher than 1. Therefore, the virtual operation must return a set of instances, rather than a single one. The cardinality is calculates as the basic number restrictions of the object property. For example, assuming that the average cardinality of the relation is 3, the cardinality pair would be:

$$n(\textit{Patient}, \textit{Drug}) = (0, 3)$$

The certainty of the construction is:

$$\mu(\Psi_{rel}) = \alpha_{rel} \cdot \frac{1}{4}$$

Instance Pattern

An instance pattern applies to situations in which the user specifies her query using a set of instances, rather than a concept. For example, the user seeks a service that return reviews

on hotels in New York, while the service-base only includes services that return review for hotels in *any* city. As *New York* is an instance of *City*, rather than a concept, it requires different handling than the set hierarchy pattern.

Let us define by I^* some arbitrary set of instances. Let us define $Ins(C_i)$ as the set of all the instances which belong to the concept class C_i, such that:

$$Ins(C_i) = \{t \in \mathcal{O} \mid t : C_i\}$$

The constructor is defined as follows:

Constructor 3: Instance Pattern.

$$\begin{aligned} VOP.input &= I_1, I_2, \ldots, I_n \\ VOP.output &= \{x : C_i \mid I_i \in Ins(C_i)\} \end{aligned}$$

We define an abstract relation between a concept class and a set of instances, $R(C_i, I^*)$. The cardinality is defined as follows:

$$\begin{aligned} &\text{Bottom boundary:} \quad N_\perp = 0 \\ &\text{Top boundary:} \quad N_\top = \frac{|Ins(C_i) \cup I^*|}{|Ins(C_i) \cap I^*|} \end{aligned}$$

The definition is based on the ratio between the intersection of the two instance sets and its union. If the two sets are identical, i.e., every instance in the arbitrary instance set is contained in the class instances, then the top boundary will be 1. In that case, the sets would be considered equivalent. In the healthcare case (Figure 2.3), consider a situation in which a user searches for a service which returns the current number of available beds in *Mount Carmel Hospital*. We define the instance set to be:

$$I^* = \{\textit{Mount Carmel Hospital}\}$$

A similar concept class might be *Hospital*. Its instance set is:

$$Ins(\textit{Hospital}) = \{\textit{Mount Carmel Hospital}, \textit{Mount Sinai Hospital}\}$$

The relation between *Hospital* and *Mount Carmel Hospital* yields the following top cardinality:

$$N_\top = \frac{|\{\textit{Mount Carmel Hospital}, \textit{Mount Sinai Hospital}\}|}{|\{\textit{Mount Carmel Hospital}\}|} = 2$$

The certainty is defined as:

$$\mu(\Psi_{ins}) = \alpha_{ins} \cdot \frac{1}{3}$$

3.2.4 Functional Affinity Patterns

Functional patterns reflect the functional similarity between compositions. They express inexactness stemmed from the structure of the composition graph, rather than from the semantic properties of the operations.We define the functional pattern as a framework for calculating similarity between compositions. The patten describe a situation in which one of the compositions has more, or less, operations than the other.

For example, in Figure 3.5, Operation *b* has one excessive operation, OP_3, than Operation *a*. The simulated composition is defined as the intersection of the operations, containing all the shared operations of the compositions. In our example, the shared operations are OP_1 and OP_2. If an operation that was removed was connecting two shared operations, such as operation OP_3 in the example, then a special virtual operation, called the *Empty Transition Virtual Operation*, denoted as VOP_ϵ, connects the operations which were removed. VOP_ϵ serves as a channel for transforming information between operations, without affecting their interface. We define the virtual operation, VOP_ϵ, using its constructor, Ψ_ϵ:

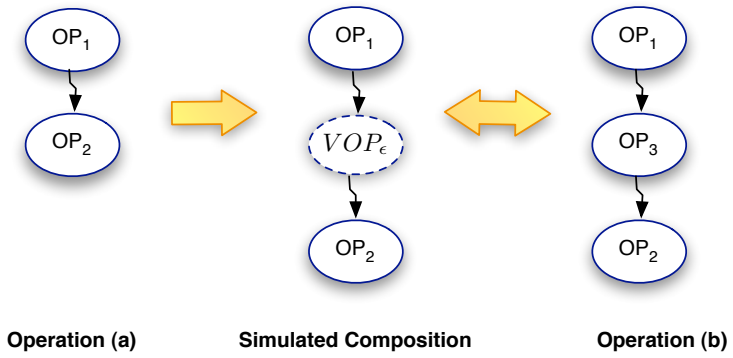


Figure 3.5: Operation *a* is matched with operation *b* using by simulating *b* using *a*

Definition 8: Empty Transition Virtual Operation Constructor - Ψ_ϵ.

$$\begin{aligned} \Psi_\epsilon(OP_i, OP_j) &= \{OP_i \cup OP_j \cup VOP\} \\ \text{s.t.} &\quad VOP.inputs = OP_i.outputs \wedge VOP.outputs = OP_j.outputs \end{aligned}$$

The certainty of the construction is the *graph edit distance* between the two original compositions and the simulated composition. The graph edit distance is defined as the number of node and edge deletions or insertions necessary to transform one graph into another [23]. It is a simple measure for graph similarity, quite similar to edit distance on strings. If the graphs are identical, the $edit(Com_1, Com_2) = 0$. If the graphs are foreign (do not contain any common subgraph), then $edit(Com_1, Com_2) = |Com_1| + |Com_2|$.

The certainty of VOM_ϵ virtual operation is set to 1, the maximal value, as we cannot assume anything regarding the reliability of the construction. The common operation certainty, μ_{common}, is calculated as the sum of the semantic certainties, over all the constructed virtual operations:

$$\mu_{common} = \sum_{V_j \in Com_1 \cap Com_2} \mu(\Psi_i)$$

The calculation uses the common operations is divided by the power of the union between the compositions. This way, identical compositions would receive the maximal value of 1. We define the total certainty of the construction as follows:

$$\mu(Com_1, Com_2) = \frac{\mu_{common}}{|Com_1 \cup Com_2|}$$

3.2.5 Similarity Function Properties

In this section we define several properties of the similarity measurement. We define the similarity function and prove that it is a distance metric. We then prove that the set of affinity patterns is complete.

Similarity Metric

Virtual operations serve as a method for measuring the similarity between operations and between compositions. In this section we formally define the similarity function and discuss its properties. The function is based on the construction certainty, which is embodied in the certainty function: $\mu : \Psi \rightarrow [0, 1]$ that represents the certainty of a given construction. Higher values indicate higher certainty and vice versa.

Definition 9: Similarity function. Let Com_1 and Com_2 be two compositions. Let us define $\widehat{\Psi} = \{\Psi_1, \Psi_2, \ldots, \Psi_m\}$ as the set of all feasible constructions for the two compositions. We denote by Ψ_{max} the constructor that maximizes the following constraint

satisfaction equation:

$$\max_{\Psi_i} \mu(\Psi_i)$$
$$\text{s.t.}$$
$$0 \leqslant \mu(\Psi_i) \leqslant 1$$

The similarity function is defined as a function of the type: $sim : Com \times Com \rightarrow [0, 1]$:

$$sim(Com_1, Com_2) = \mu(\Psi_{max}(Com_1, Com_2))$$

The function measures the resemblance between two compositions, reflecting the minimal amount of work necessary for adapting a composition to a query. Now, we prove that this function is a distance metric. A distance metric (M, d), where $d : M \times M \rightarrow \mathbb{R}$, satisfies four constrains: it is non-negative, it keeps the identity property, it is symmetric and it satisfies the triangle inequality property.

Theorem 1: . The similarity function, defined in Definition 9 is a distance metric.

Proof. sim satisfies these properties as follows:

1. Non-negativity: As $sim = \mu(\Psi_{max})$, and $\mu(\Psi_{max}) \geqslant 0$ (according to Definition 9), then $sim \geqslant 0$.

2. Identity: Each constructor Ψ, including Ψ_{max}, is a valid constructor and hence the distance function preserves the identity property (see Definition 7).

3. Symmetry: Again, derived from the commutativity property in Definition 7.

4. Triangle inequality ($sim(x, z) \leqslant sim(x, y) + sim(y, z)$). Let us assume that there exist three compositions, x, y and z, such that $sim(x, z) > sim(x, y) + sim(y, z)$. According to definition 9, Ψ_{max} is the construction that yields the maximal certainty. We can construct $\Psi' = \Psi_{max}(x, y) \cup \Psi_{max}(y, z)$, with $\mu(\Psi') \geq \mu(\Psi(x, z))$. Because this construction is feasible, it would be chosen as the maximal certainty construction, such that, $\Psi' = \Psi_{max}(x, z)$, and $sim(x, z) = \mu(\Psi')$. Therefore, $sim(x, z) \not> sim(x, y) + sim(y, z)$

□

We now define a normalized similarity function. The normalized similarity reflects scale issues which cannot be expressed by affinity patterns. Unlike affinity patterns, which are used to evaluate the similarity of any two compositions, the normalized similarity function reflects results which are directly related to the context of querying a service

retrieval framework for services. The normalization is derived from two observations, which are supported by our emperical findings:

- Impact ratio: The distance function depends on the relation between the size of the compositions and the number of changes.
- Decay: The similarity decreases in a higher magnitude than the number of changes. In our model.

For any pair of compositions, we define M to be the largest composition, such that $M = argmax(|Com_1|, |Com_2|)$.

Definition 10: Normalized Similarity.

$$sim_n(Com_1, Com_2) = \left(\frac{|M|}{|M \cup \Psi_{max}|} \mu(\Psi_{max}) \right)^d$$

The d is the decay factor, such that $d > 1$. As the expression within the brackets is bounded by 1, a high decay factor would suggests a strong decrease in the similarity, while a low decay factor would suggest a weaker decrease.

Completeness of Affinity Patterns

An important issue regarding affinity patterns are their completeness. Can they be used to measure the similarity between any two possible compositions? In order to answer this question we need to look at semantic and functional patterns. Proving that the functional pattern is complete is simple: any two graphs can be compared by adding and removing nodes and edges. Our proof regarding semantic affinity is more complex and is based on comparing the abilities of affinity patterns with the definitions of our world. 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.

Theorem 2: . The set of semantic affinity patterns is complete.

Proof. For any given concept, $\check{C}$, 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. Each concept in the ontology is classified to *one* of the context classes. For example, all equivalent concepts to concept $\check{C}$ would be classified to the Equivalents context class. A complete set of semantic affinity patterns would satisfy all possible axioms within an ontology, as defined in Definition 1:

Context Class	Definition
Equivalents	$\{C_i \in \mathcal{O} \mid C_i = \check{C}\}$
Instances	$\{C_i \in \mathcal{O} \mid C_i : \check{C}\}$
Types	$\{C_i \in \mathcal{O} \mid C_{anchor:C_i}\}$
Subclasses	$\{C_i \in \mathcal{O} \mid C_i \sqsubseteq \check{C}\}$
Superclasses	$\{C_i \in \mathcal{O} \mid \check{C} \sqsubseteq C_i\}$
Properties	$\{P_i \in \mathcal{O} \mid P_i \in P(\check{C})\}$
Relations	$\{C_i \in \mathcal{O} \mid \exists R, R(\check{C}) = C_i\}$
Composed	$\{C_i \in \mathcal{O} \mid \exists C_1, C_2, \ldots, C_n, C_1 \in Class(C_2) \ldots C_n \in Class(\check{C})\}$
Unrelated	$\{C_i \in \mathcal{O} \mid C_i \notin Class(\check{C})\}$

Table 3.1: Context classes

The set of context classes is complete. All the classes until the Composed class cover all the axioms of the ontology. Theorizing on the ontology as a graph, these classes define all direct relations between every two concepts. A concept which has an indirect relation to the anchor concept is defined through the Composed class as a concept which is related to the anchor concept through a set of $C_1, C_2, \ldots, C_n$ concepts. The context class Unrelated defines all concepts which are not related through any other context class (including the composed class).

Each of the context classes is covered by an affinity pattern, as follows:

1. The set-hierarchy pattern covers all cases of the equivalent, subclass and superclass.
2. The relation pattern covers all cases of object and datatype properties.
3. The instance pattern covers all cases of instances and types.
4. Pattern composition covers all cases of the composed pattern.
5. Unrelated concepts are not similar, and therefore cannot be bridged by a virtual pattern.

Therefore, there cannot be a pair of concepts which cannot be evaluated by the affinity patterns. The conclusion is that the set of affinity patterns is complete. □

3.3 Evaluating Affinity Patterns

The relevance of affinity patterns was measured through an experiment, with the objective of predicting the way human users would benefit from retrieval systems that utilize the

patterns. The patterns introduced above provide a catalogue of similarity measurement tools. However, we need to evaluate them against a benchmark of composition similarity. To this end we designed a detailed experiment, in which human subjects were asked to assess the similarity between models. The studies of Budanitsky and Hirst [22] and Bernstein et al. [13] show that in a setting of ontology-based knowledge systems, human judgments give the best assessments of the quality of a measure for affinity between concepts. The first study compared WordNet [37] similarity measures, while the second one compared an ontology for service description. For the best of our knowledge, this experiment is the first one measuring human conceptions of similarity in the context of semantic Web services.

3.3.1 Experiment Design

The experimental process required software and information systems engineering students to go through a set of tasks. In each of the test case tasks the student is asked to assess a set of pairs, each containing a query describing a simple set of requirements for a system and a model of a possible composed system. The following sub-sections describe the experimental setting, the research population, and the test cases used for the experiments.

Setting

The participants were asked to assess the relation between the set of requirements and the composition, ranking and explaining their ranking. The requirements are displayed as a short textual fragment, resembling the format of the query, in a service retrieval framework, hence they are referred to as queries. The composition was visualized by OPM (Object-Process Diagram), the visual formalism of OPM (Object-Process Methodology) [32], offering a comfortable graphical user interface. An explanation of OPM is provided in Appendix 7.1. Each test case, i.e., a query and composition pair, was displayed to the participants using a Web-based user interface[2], as shown in Figure 3.6.

The ranking is provided on a Likert scale [55] of 1 to 5, for each one of the following three parameters:

- *Usefulness*: The degree to which the model can be used in order to implement the query.
- *Completeness*: The degree to which the model meets all of the query's requirements.

[2]The experimental system can be accessed at http://dori.technion.ac.il/Survey-Manager/

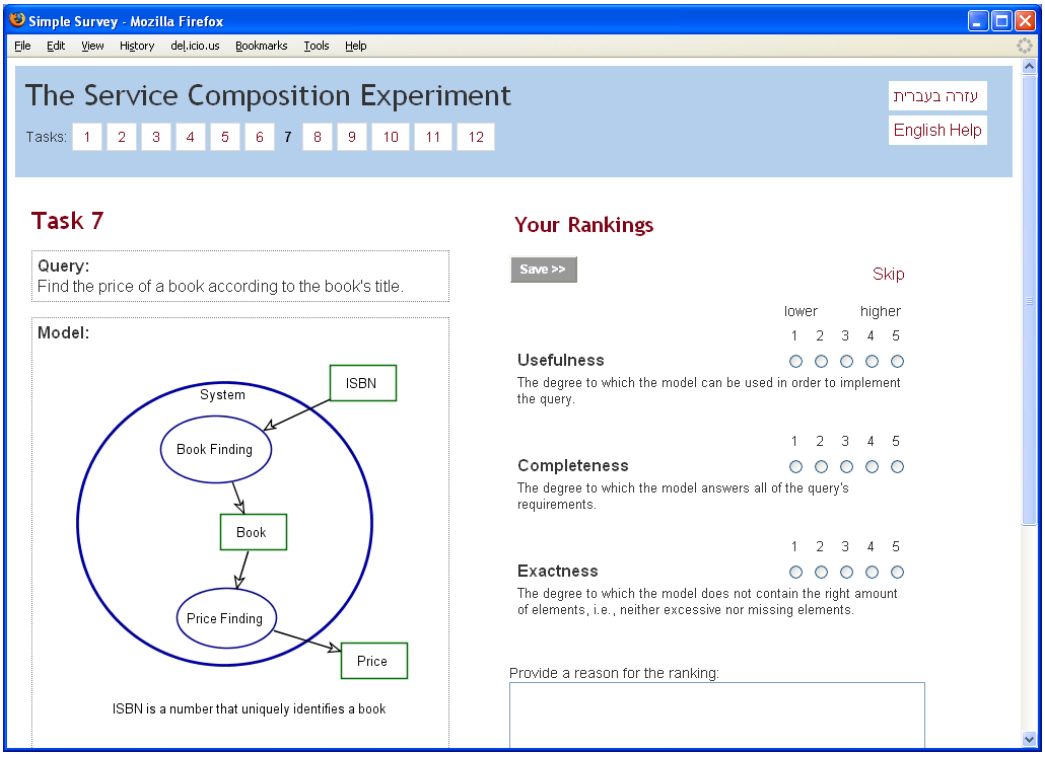


Figure 3.6: A sample survey page showing the model and the feedback form

- *Exactness*: The degree to which the model does contain the right amount of elements, i.e., neither excessive nor missing elements.

The first parameter, usefulness, measures the correspondence between the query and the composition in the context of *reuse*. The two other parameters measure the correspondence in a more general context. Exactness and completeness overlap, as an exact matching is also a complete matching. However, we wanted to gain a more subtle distinction between a situation in which the composition contains excessive elements and a situation in which the composition has missing elements. The completeness parameter would distinguish between the two cases. The participants were also asked to provide additional text that describes their rankings, in the **explanation** field. In the **improvement** field, they were asked to provide a textual description on what changes they would make to the composition in order to turn it more similar to the query. The objective of the last description is to strengthen the aspects related to reuse, supporting the ranking according to the usefulness parameter.

Each of the participants was presented with an introduction that includes explanations regarding the experiment and the required feedback. The participants were also asked

to supply demographic information on their age, years of study, and education (high-school, bachelor, etc). Following that, each participant was presented with a sequence of 12 query/composition pairs. The pairs were randomly selected from the possible 30 pairs, and randomly ordered. After ranking all the pairs, the participant had the option of modifying her rankings and answers.

Research Population

The research population included 127 participants, of whom 15% were studying towards their masters or doctoral degrees, while the other 85% were bachelor students in their 5th semester. 70% of the students were students or graduates of the faculty of Industrial Engineering and Management at the Technion IIT, Israel, while the other 30% were students or graduates of the faculty of Computer Science at the Technion IIT, Israel. 95% of the participants took the course "Analysis and Specification of Information Systems", taught by the author of this thesis. For the participants who were studying towards a bachelor degree, participating in the experiment was defined as a bonus task at the course, crediting the participants with 3% of the final grade of the course. The grade itself was independent of their answers. Rather, it was based on their thoughtfulness in providing the feedback.

This population is a good proxy to the general user community which is relevant to our research, which are software engineers and system analysts. All the participants had took at least a basic and an advance course in software engineering. All of the participants took also at least one course in system analysis. Therefore, their education reflects the type of education of our target population. Furthermore, we argue that their relative lack of experience compared to skilled software engineers strengthens our results. One of the characteristics of experienced software engineers and system analysts is their ability to intuitively evaluate semantic and structural approximations. As we have conducted our experiments on novice engineers, these intuitive skills are less developed.

Test Cases

Our experiment was based on a set of four ontologies in three different domains: e-commerce, geography, and publications. We specified eight different queries, and matched them to 3-6 different compositions in each domain (for a total of 30 different test cases). The concepts and the relations used in each composition were taken from the ontologies. Figure 3.7 depicts two test cases. Each set of compositions which were related to a single query met the following criteria:

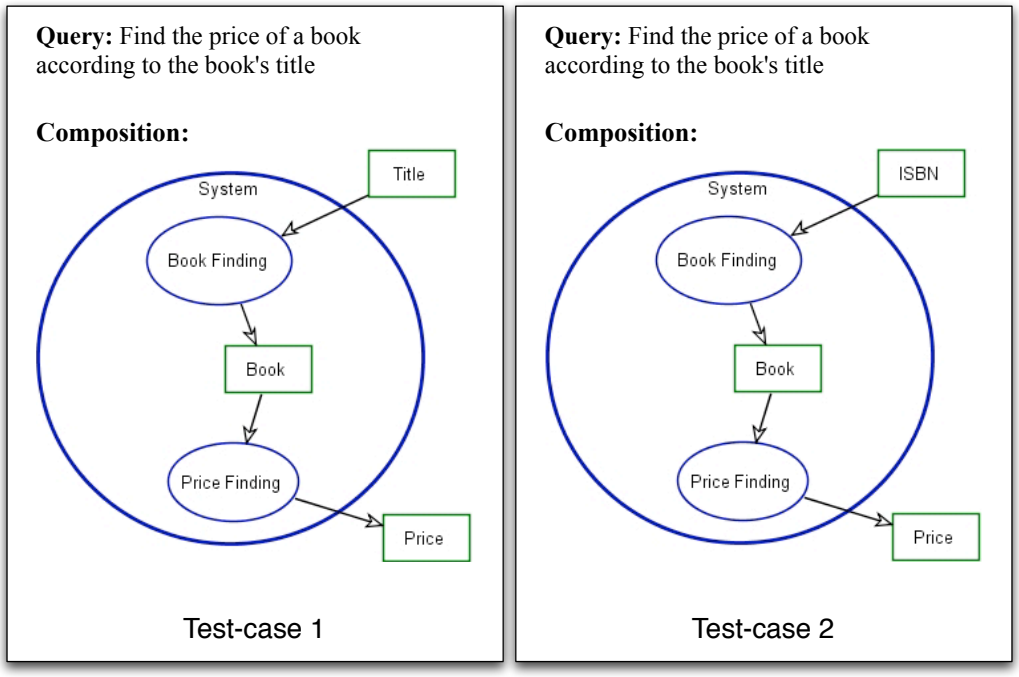


Figure 3.7: A sample of two test cases

- At least one composition answers the given query perfectly. We denote this composition as the *baseline* composition.
- At least one composition has a minor difference with respect to the query.
- At least one composition has a major difference with respect to the query.

The compositions and their relation to the query, were assessed by two fellow Ph.D. students before the experiment.

In order to produce results which are relevant to our service retrieval framework, we have used a subset of OPM's expressibility, which does not go beyond the scope of our definition. This scope includes processes (e.g., operations), objects (e.g., parameters), result links (e.g., outputs and effects), consumption links (e.g., inputs and preconditions) and states (e.g., instances). All the participants were knowledgeable in OPM.

3.3.2 Results

The results of the experiment are organized in two categories. First, we look at general characteristics of service similarity, such as how different parameters, e.g. usefulness and exactness, differ. Second, we evaluate each affinity pattern.

Parameter Rankings

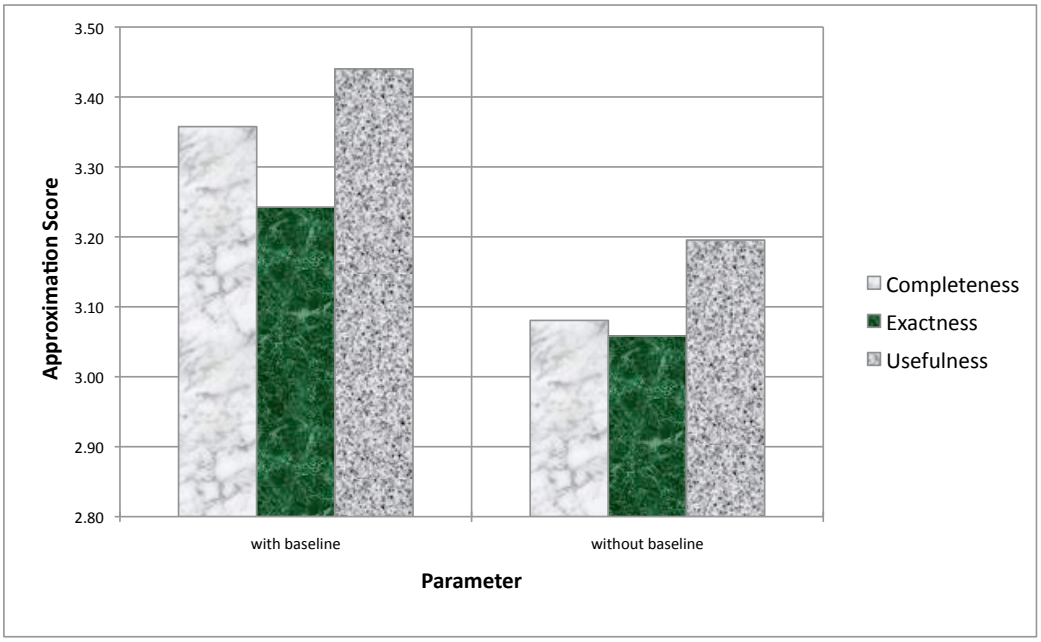


Figure 3.8: The average score of completeness, exactness, and usefulness

A clear observation refers to the difference between the three parameters. While the correlation between them is high (≈ 0.93), there are significant differences between their average values. Figure 3.8 represents the average ranking for completeness ($E(C)$), exactness ($E(E)$), and Usefulness ($E(U)$). Participants tended to give higher values to *usefulness*. This is more noticeable when the baseline results are removed. The context of *reuse* enables participants to accept more approximate results.

The result of higher usefulness values is supported by qualitative analysis of the text feedback given by participants in the *explanation* parameter versus the *improvement* parameter. When describing their ranking in the explanation field, the participants tended to base their feedback on the existing aspects of the composition. They thoroughly described the differences between the composition and the query in terms of missing/excessive elements, different process order, different interface and so fourth. However, when describing the improvements, participants tended to be more creative, specifying new operations and data structures, and relating them to the existing composition using inheritance, relations, etc.

Table 3.3.2 demonstrates how the same participants gave textual feedback. The task and participant are identical for each row in the table. The left column includes frag-

ments form the *explanation* field, while the right column includes fragments from the *improvement* field. The context of improvement allows the participants to think of the composition/query pair in a more flexible terms. As the context is considered less rigid, the participants suggest changes in several aspects. Three main categories of changes were identified in the improvement feedback:

- Altering the interface of the composition. In this category, the type of the input and output parameters is altered in order to accommodate missing or different data. In many cases the alteration takes the form of a *filter*: which adds or removes unnecessary information form the input and output parameters.
- Adding functionality. In this category, missing processes are added to the composition.
- Altering functionality. In this category, the inner behavior of processes is altered, by adding loops, preconditions, etc.

Set Hierarchy Pattern

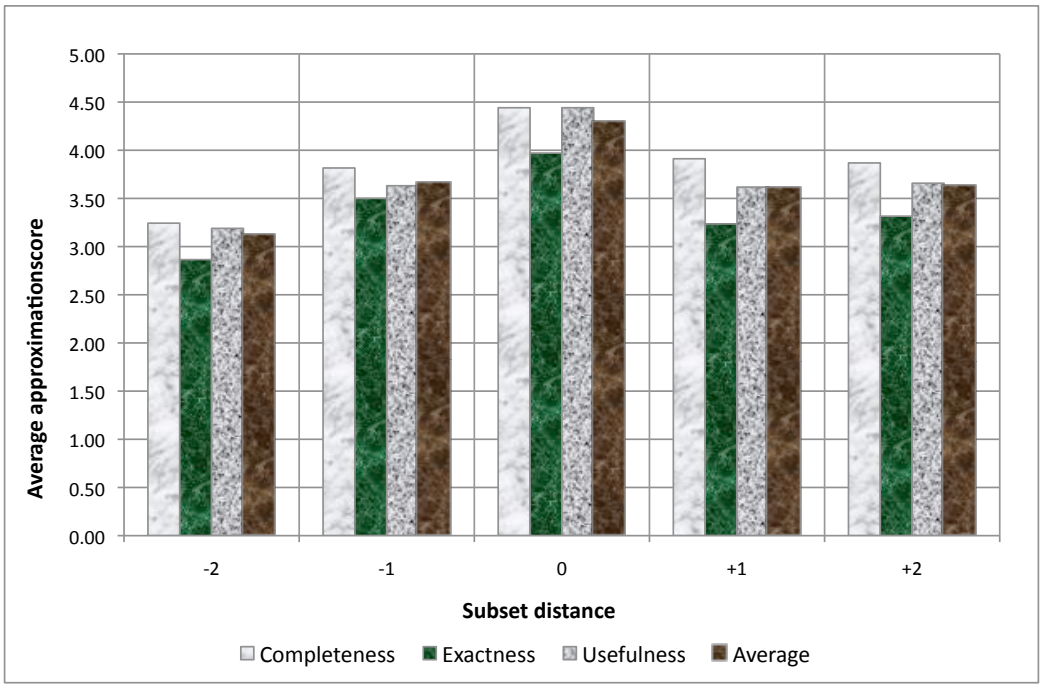


Figure 3.9: The average ranking for subclass relations, according to the subset direction from the baseline

Case	Explanation Feedback	Improvement Feedback	Category
1	"The book finding gets the isbn but according to the query we should search according to the book title"	"I would replace isbn by book title and make characrerization of the book"	Alter interface
2	"Model is more specific, query is more general - they should be compatible to each other. Price finding is unnecessary (could be replaced with simpler solution)."	"Change 'professional book finding' to 'book finding'. Change 'Computer related book' to 'Book'. Replace 'price finding' with 'price', which will be an attribute of 'book'."	Alter interface
3	"There is no way to add products to the shopping card"	"add the appropriate process (adding product) between login and checkout. The process accepts Product and updates the cart."	Add functionality
4	"The main issue with the model is that it takes as an input an existing cart, and doesn't provide the required functionality of adding products to a cart."	"Add process for adding products the a shopping cart as the first procss in the system (before login); this processes should be wrapped with something equivalent to a while loop."	Add functionality
5	"The search is more general, so if person has many phone numbers, maybe in this search we won't find the number at the faculty."	"I will customize the Person Profile Finding to find profile in the faculty (by generalization/specification link."	Alter functionality
6	"Why only computer related books can be found?"	"I would add a temp result list which is created by the Professional book finding and then a choice should be made within this list."	Alter functionality

Table 3.2: Examples of textual feedback

The analysis of the subclass relations yields a clear patter, shown in Figure 3.9. Each column in the graph stands for an average score given by participants for all the three test cases related to the set hierarchy pattern. The first three columns in each series depict the average score for completeness, exactness and usefulness, while the fourth is the average value. The results are ordered according to their subset distance, which is their location in the class hierarchy. Column -2 and column 2 indicate compositions including concepts

which are more specific (minus) or more general (plus) than the baseline (0). For example, in one of the test cases, the query was: *Find the price of a book according to its title*. The concept was the catalogue used for the search process, and the location in the hierarchy of the exhibited values were:

Concept	**Hierarchy location**	**Subset distance**
C++ programming books	Most specified	-2
Computer related books	Specified	-2
All books	Baseline	0
All publications	General	1
All products	Most general	2

Table 3.3: Location of concepts within the subclass hierarchy

The average score by subset distance form a bell pattern, where the highest score for completeness, exactness and usefulness, is received for the baseline query/composition pairs. The curve is steeper on the left hand side of the graph, indicating that participants tended to rank more specific compositions lower than more general compositions. In fact, the difference between more general results becomes insignificant after the initial difference from the baseline, so that the average score (the fourth column in each subset distance set) of the $+1$ set and the $+2$ set are almost identical. This phenomenon recurs in each of the test cases of the set hierarchy pattern.

Relation Pattern

The relation pattern is analyzed via the cardinality class. The certainty function μ reflects the probability that given a concept, a related concept would be found. For the relation pattern μ is calculated as a function over the cardinality of the relations. In Figure 3.10, the average completeness scores are presented for three test cases (TC-1, TC-2, and TC-3). The usefulness and exactness values behave similarly, and they are therefore omitted from the chart for the sake of clarity. Each point along the X-axis represent a certainty value, starting from the baseline, where the query concept is equivalent to the composition concept, through $1:1$ relations (such as ISBN $\leftrightarrow$ Book), $1:2$ relations, $1:3$ relations, and $1:n$ relations.

The results show a decline in average completeness scores as the cardinality grows. The baseline yields the highest similarity values, while higher cardinality yields lower similarity. The negative slope of the curve becomes moderate as the cardinality grows. In higher values ($1:3-1:n$), the curve flattens to the point of insignificance. This result

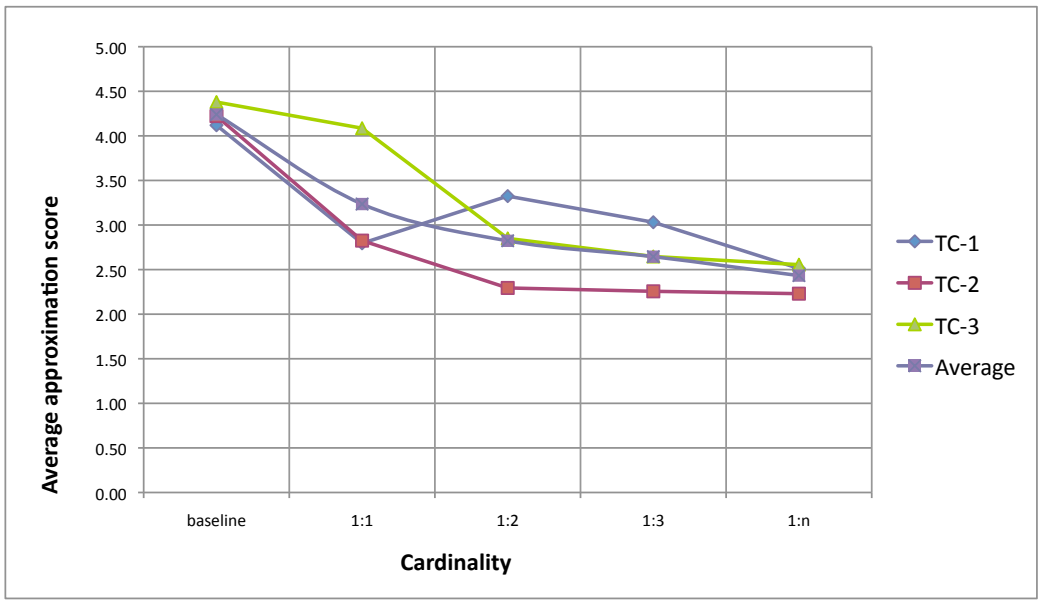


Figure 3.10: The average completeness score ordered according to the c function

shows that similarity decays as the cardinality grows. In one of the test-cases (TC-1), the data point in in 1:2 cardinality has an average score which is higher than the point 1:1 cardinality, due a concept which can be considered either a related concepts or a generalized concept.

Instance Pattern

The instance pattern approximates concept classes using instances and vice versa. In the experiment we tested only approximating a class by its instances. Figure 3.11 depicts the results. As in previous charts, each column represents a parameter (completeness, exactness and usefulness), the X axis represents the instance classification distances, and the Y axis represent, the average score. The instance classification distance is derived from the number of instances divided by the number of all possible instances. The value 0 represents perfect matching, while higher values represent descending probabilities. The main result is the negative correlation between the instance classification distance and the average approximation score, which average descends rather sharply when the participant is presented with a set of instances rather than a concept class.

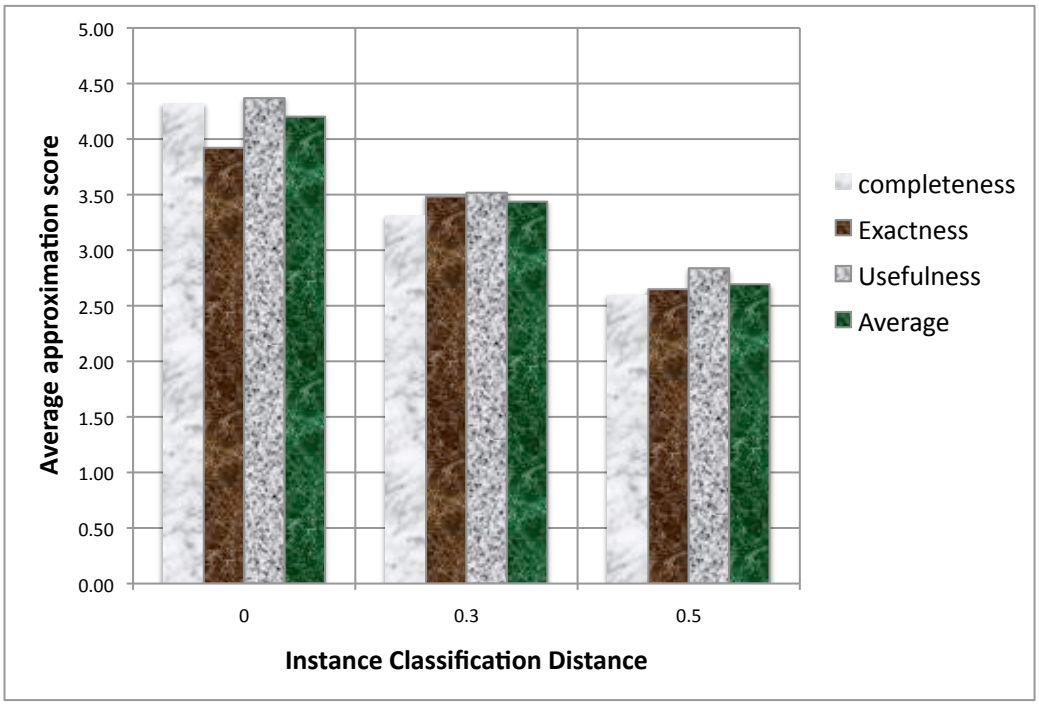


Figure 3.11: The average ranking according the instance-classification distance

Functional Affinity Pattern

In Figure 3.12, the results for the functional similarity pattern are displayed. The X-axis represents the number of graph edit distance carried out between the composition and the query, which is the edit distance defined in Section 3.2.4. These edits include removing and adding nodes and edges. The 0 set represent is the baseline pairs set, in which compositions are functionally equivalent to the query. The 1 set represents a single edit, and so fourth. The Y-axis represents the approximation score, over all the test cases and participants.

The most significant trend is apparent for usefulness, which decreases by about 25% percent from the 0 set to the 2 set. It is not surprising that missing or excessive elements make the composition less usable than an identical composition. The exactness parameter was more sensitive to excessive elements, while the completeness parameter was more sensitive to missing elements.

An interesting phenomenon is the relation between the size of the composition (the overall number of elements within the composition), and the similarity. For example, Table 3.3.2 compares two test cases. The first containing 5 elements and the second 9 elements. The completeness decline between the baseline (0 set) and the 1 set rises

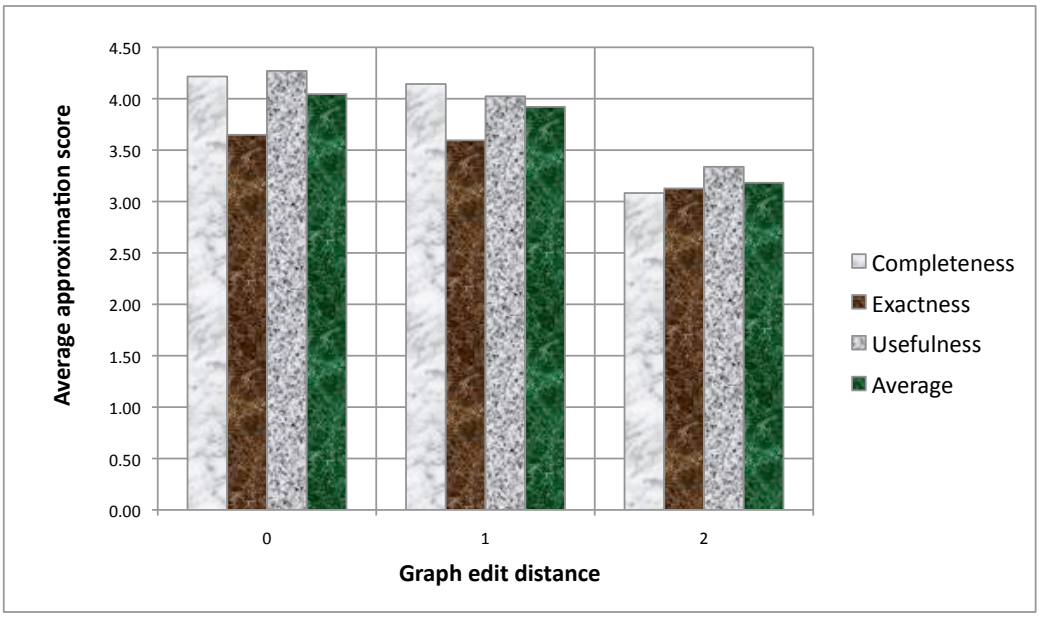


Figure 3.12: The average ranking according to the edit distance between the query and the composition

Average size	Average Decline
5 elements	16%
9 elements	30%

Table 3.4: The relation between composition size and completeness decline

sharply, with direct relation to the size of the composition. As the number of elements in the composition grows, the effect of the change becomes less significant. A similar significance

The same result is relevant to usefulness and exactness, but with lower significance. A possible reason for that is the different contexts. While completeness requires the participants to evaluate how the whole composition is comparable to the query, in usefulness, the participant is focused only on the relevant parts of the composition.

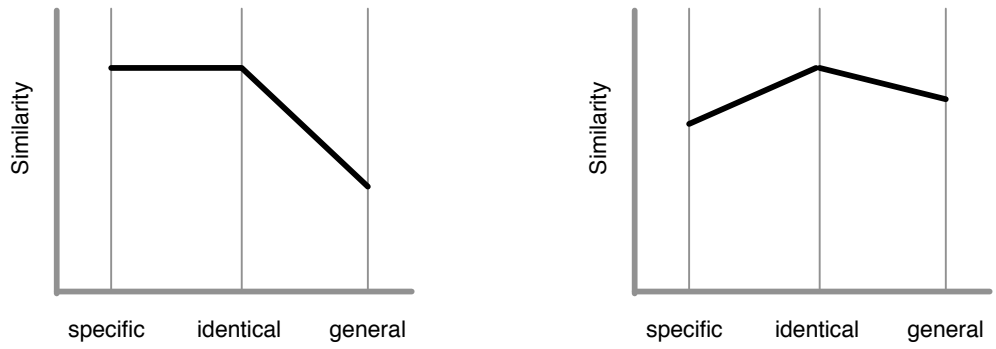
3.3.3 Discussion

The experiment results provide a basis for evaluating the abstract affinity patterns presented in Section 3.2. As far as we know, this is the first experiment in which paradigms of service retrieval were examined with human participants. The results capture characteristics of service retrieval which were not apparent in the current research. The experiment

did not yield exact models that can be represented with mathematical formulae but indicated general trends in the patterns. These pattern trends can support or contradict existing theories.

Two different modes for service retrieval were identified. The first one, employing the completeness and exactness parameters, is based on semantic and functional comparison between the query and the composition. The second model, which employes the usefulness parameter, reflects the context of **reuse**. Participants were more forgiving for inexactness in the reuse context, as the average score for usefulness was higher than that for completeness and exactness. The difference between the contexts was also apparent in the textual feedback of the participants, who suggested various means for changing the composition in order to answer the query.

The results for the set hierarchy pattern point against the relevance of logic-based methods for service discovery. Logic-based results assign equal importance to plugin (more specific) and to the exact (baseline) results [49], because more specific results follow the axioms of the general results. This is depicted in Figure 3.13 (a), where the Y-axis represents some abstract similarity, while the X-axis represents the specification dimension. The difference starts with more specific results, through the baseline results (identical to the query), and ending with the more general results.



(a) Logic-based Methods **(b) Our Results**

Figure 3.13: Comparison of logic-based similarity versus our results

For logic-based methods (Figure 3.13 (a)), the similarity is identical for specific results and the baseline results, and it declines sharply for more general results. However, according to our results (Figure 3.13 (b)), human participants perceive specific results as inaccurate like general results, yielding an upside-down This notion is depicted in chart *b*, where the similarity forms an upside **v**-shaped curve. Moreover, we discovered that that

human participants perceive a "softer" notion of similarity than the one defined by logic-based methods compared to [50, 58]. This result is depicted visually by the less steep declining slope of the similarity curve, compared with the slope of logic-based methods.

The second result of the experiment is the evaluation of the relation and instance patterns. The evaluation proved that human participants perceive relations and instance-classifications as valid means for approximation. There is a relation between query-composition similarity and the probability of instance selection.

The experiment allows us to set the α values, which are the pattern certainty coefficient, which is defined in page 32. The motivation behind the coefficients is to differentiate between the similarity values of different patterns. The α value is calculated as a function of the gradient of the similarity curve. Therefore, patterns with a steeper gradient, such as the instances pattern, would receive a lower coefficient. The following table includes the exact α values calculated according to the experiment.

Pattern	α coefficient
Set hierarchy (general)	0.6
Set hierarchy (specific)	0.45
Relation	0.7
Instances	0.2

Table 3.5: α values for different patterns

Chapter 4

Service Retrieval

In this chapter we discuss the syntax and semantics of query evaluation in the context of approximate service retrieval. We introduce an extensible abstract query language, which can be used to represent a wide variety of query language modals (Section 4.1). We specify the semantics of approximate query evaluation using the notion of μ-satisfiability: a mean for quantitatively matching a query segment with a set of services.

We then turn to the way service data is analyzed from service descriptions. We introduce the *service network* - a data structure that stores information about the services (Section 4.2), and the way it is constructed from existing Web services on the World-Wide-Web.

4.1 Query Evaluation Semantics

In this section we provide an overview of the query evaluation semantics. Adopting a bottom-up approach, we start with a detailed element description and continue with a complete structure of the query. Section 4.1.2 describes extensions of the basic query model.

4.1.1 Query Syntax

A query in the context of service retrieval is a tree of operations augmented with logical operators. Each operation serves as a template for matching. Naturally, the data contained in the query is sparse, with respect to the data contained in the service description. The following structure defines how queries are abstractly specified. The abstract query is implemented by a concrete query language, which can take the form of a keyword-based language or a model-based language (see Section 5.3 for more details).

Definition 11: Abstract Query, Q. A query is a directed tree $Q \equiv \langle N, Flows \rangle$, where:

- $N = \mathcal{OP} \cup \{\wedge, \vee\}$, where $\mathcal{OP} = \{OP_1, OP_2, \ldots, OP_n\}$ is a set of operations and $\wedge, \vee$ are conjunction and disjunction nodes, respectively.
- $Flows \subseteq N \times N$ is a set of directed data flows between nodes. The flows are ordered form left to right.

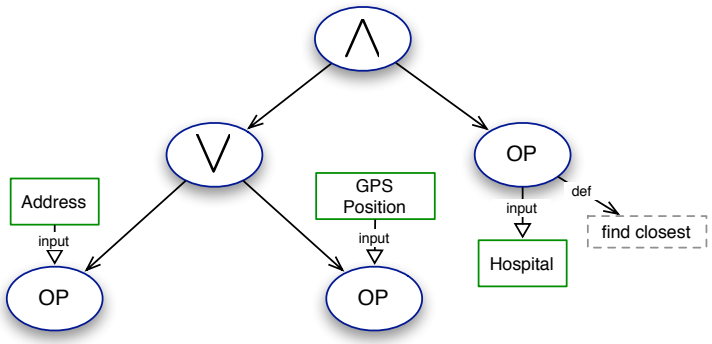


Figure 4.1: An abstract query

We call an operation node a **leaf** node, and a conjunction or disjunction node a **composed** node. For the sake of simplicity, each composed node can include a maximal number of two flows. Figure 4.1 depicts an example of an abstract query structure. The query formally represents the set of requirements expressed in Example 1. The requirements and the representation differ in one aspect: in order to exemplify the disjunction, the query contains the input of GPS position, which is not part of the example. The top node is the $\wedge$ node that conjunct the rest of the query nodes. The conjunction includes the $\vee$ node which defines two equally acceptable query fragments: an operation that receives an *address* and an operation that receives a *GPS position*. The other conjunctive part represent the second half of the query: An operation that is returns a *hospital* concept, which is *defined* as the closest hospital.

We use a second notation for queries, which is based on text, rather than on diagrams, and, hence, more compact. Operations are described using a set of label/value elements, where colons separate the label from the value. A set of labels is separated by semicolons. In some cases, we omit the label from the pair. In this case, the label can be any type of label (in - input, out-output, def-definition, etc). Conjunctions and disjunctions are denoted by $\wedge$ and $\vee$. For example, the following is the same query which is identical to the one in Figure 4.1:

$$((in, Address) \vee (in, GPSPosition)) \wedge (out, Hospital; def, \textit{find closest})$$

4.1.2 Query Language Extensions

We present an extension for the query language, which allows users to write complex queries without compromising the simple syntax of the query language. There are two types of syntax extensions and property extensions. Syntax extensions add syntactic sugar to the query language. 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* $\wedge$ *hospital* $\wedge$ *optional(availability)*", the last token is optional. Therefore, results which contain the *availability* property will be ranked the same as results which do not contain it.

The implementation of this extension 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 only 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 wishes to select services with an output which is hospital, clinic, or doctor, the following query pattern can be used: "*address* $\wedge$ *any(hospital, clinic, doctor)*". This is automatically be translated to the following pattern: "*address* $\wedge$ *(hospital* $\vee$ *clinic* $\vee$ *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 *def* (for a concept which defines the functionality of 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 during which the service is available.
- **Provider:** The organization which provides the service.
- **Location:** The geographical location in which the service is provided.

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

Extension properties are defined by the users 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 of the original concept before continuing with the retrieval process.

4.1.3 Query Matching Semantics

The result of the query evaluation is a *result-set*, i.e., a ranked set of results, each being a *composition* of operations. A result may contain operations that originate from diverse sources. Each result is associated with a *similarity value*, expressing the certainty with which the set of operations answers the query. The notion of similarity is embodied in the μ-satisfiability relation, defined as follows:

Definition 12: μ-satisfiability. Let R be a result, and let Q be a query. The μ-satisfiability relation, denoted as $R \models_{\mu} Q$, indicates that R satisfies the requirements of Q with a certainty of μ.

We define the levels of matching recursively. 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. We can now define the operation satisfiability of a query leaf node as follows.

Definition 13: Operation Satisfiability. An operation OP satisfies n_{Q_l} if the following applies: $sim(n_{Q_l}, OP) \geqslant \hat{\mu}$ - the similarity between the two concepts is higher than a threshold $\hat{\mu}$.

The method for calculating μ, the semantic correspondence function, is given in Section 3.2.5.

4.1.4 Complex Query 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 $R \models_\mu Q$, when a query can be satisfied by a result, in a given μ level of certainty.

In disjunction, the query is transformed into a disjunctive normal form. For instance, the example query (depicted in Figure 4.1), 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 result 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 14: Disjunction Matching. $R \models_\mu (n_{Q_1} \vee n_{Q_2}) \Leftrightarrow R \models_\mu n_{Q_1} \vee R \models_\mu n_{Q_2}$. The certainty is defined as $\mu = max\{\mu_1, \mu_2\}$, where 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 operations. 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 reduces 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 a set of operations, resulting in a correspondence value that depends on the correspondence of the nodes and the certainty of the composition. In the simplest case, the set of operations can include a single operation. In this case, the pattern matches the operation, where the left query node is matched against the inputs of the operation while the right query node is matched against the outputs of the operations.

If the conjunction is matched against a set of operations, which include two or more operations, then the matching must satisfy not only the query nodes, but also each operation in the result must be composed to its consecutive operation. Composability of two operations depends on their respective input and output parameters. Let us denote by OP^{in} a partial view of a given operation, OP, which includes only its input parameters. We define this new view by restricting the $label$ according to its domain. OP^{in} is achieved by removing all labeling which are not mapped to the $input$ property. Similarly, we define OP^{out} as a partial view of an operation, which includes only its output properties.

Definition 15: Composability. Two operations, OP_1 and OP_2 can be composed if $sim(OP_1^{out}, OP_2^{in}) \geqslant \hat{\mu}$. Composed operations are denoted by $OP_1 \oplus OP_2$.

Having defining composability, we formally define the μ-satisfiability specification of conjunction matching as follows:

Definition 16: Conjunction Matching. We say that $R \models_\mu (n_{Q_1} \wedge n_{Q_2})$ if the following conditions hold:

1. R contains a source and destination operations, OP_s and OP_d, such that $OP_s, OP_d \subseteq R$ and $OP_s \models_\mu n_{Q_1} \wedge OP_d \models_\mu n_{Q_2}$.
2. There exists a set of operations $Path = \{OP_1, OP_2, \ldots, OP_k\}$ such that:
   (a) If n_{Q_i} is a leaf node, $Path$ 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*, $Path$ 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).

3. The overall composition certainty of the path (detailed below) is higher than a given threshold.

Since the query has been transformed into disjunctive normal form, any node can only be either an *and-node* or a leaf node. The composition certainty reflects the certainty of the *dependencies* between operations. We define a composition path, $OP_1 \overset{\cdots}{\oplus} OP_n$ as the set of edges belonging to the shortest path between two operations, OP_1 and OP_n. The certainty of the composition is defined as the product of all the similarity values, as is common in the literature [30]:

Definition 17: Composition Certainty. The certainty of a composition path, $P = OP_1 \overset{\cdots}{\oplus} OP_n$ is defined as:

$$\mu(P) = \prod_{(OP_i, OP_{i+1}) \in P} sim(OP_i, OP_{i+1})$$

Note that Definition 16 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 values, as their composition certainty is maximal. As the similarity function is bounded by 1, $\mu(P)$, which is a product of similarity values is also bounded by 1.

4.2 The Service Network

4.2.1 Definitions

The *service network* is a data model for representing approximate service composition. 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. Each dependency is associated with a certainty value, representing the similarity measures specified in Section 3.2.

Definition 18: Service Network. A service network is a graph $\mathcal{SN} = \langle \mathcal{OP}, Flows, cat, \gamma_D \rangle$, such that

- $\mathcal{OP} = \{OP_1, OP_2, \ldots, OP_n\}$ is a set of operations.
- $Flows \subseteq OP \times OP$ is a set of asymmetric dependencies between operations.

- $cat : Flows \rightarrow \{inferred, empirical\}$ assigns a category to each of the dependencies.
- $\gamma_D : Flows \rightarrow [0, 1]$ assigns a certainty value to each dependency.

Intuitively, the service network ($\mathcal{SN}$) can be thought of as a union of all possible compositions that can be retrieved by a retrieval framework. The basic data structure of the service network is similar to that of a composition (Definition 4), but with several differences: it may contain circles, it is not necessarily connected, and it contains additional information regarding flows. The category function, cat differentiates between *inferred dependencies*, which are inferred from similarity between operations, and *empirical dependencies*, which are derived from existing service models. The certainty value, γ_D, indicates the certainty values, as inferred by the similarity function.

An example of a service network is depicted in Figure 4.2. As noted above, the directed arrows represent dependencies between operations. Input and output message parameters (labeled *in* and *out* respectively) are mapped to ontological concepts, described in Figure 2.3. The directed arrows form dependencies, which represent procedural links between operations, connecting separate operations into service networks.

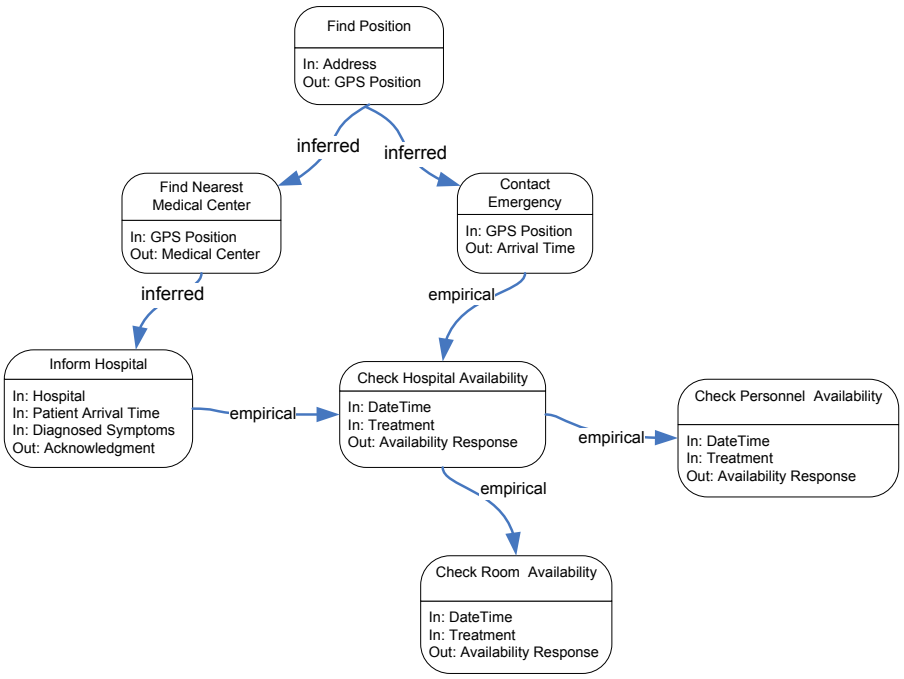


Figure 4.2: An Example of Operations and Dependencies

In this work we focus how OWL-S service repositories can be represented by the

service network. Services expressed in syntactic languages, such as WSDL [25], can also be represented by the service network (see [39] and [68]). As the service network contains certainty values for dependencies and operation properties, any mapping that maps service properties to $[0, 1]$ range can be used. Constructing the service network from a repository of services is a three-step process:

1. Deriving operation properties.
2. deriving dependencies from the repository, and
3. inferring dependencies from similarities between operations.

In the first step, an operation in $\mathcal{SN}$ is created for each atomic process in the OWL-S repository. In OWL-S services, there are basically four types of parameters: Inputs, outputs, preconditions and effects. Each parameter is transformed to a service property in the $\mathcal{SN}$ data structure. In order to simplify our model, and without loss of generality, the effect parameters are transformed to output properties and the precondition parameters are transformed to input parameters. Consider, for example, the operation *Check Personnel Availability* in Figure 4.2. It contains three parameters, which are transformed to the following instantiation:

Example 3: Check Personnel Availability. The operation is represented according to the Operation definition:
$Props = \{\text{input}, \text{output}\}$
$label = \{\text{input} \rightarrow \text{DateTime}, \text{input} \rightarrow \text{Treatment}, \text{output} \rightarrow \text{Availability Response}\}$

The second step in the construction of the service network is learning about dependencies between operations from the service models (*empirical* dependencies). This method is described in Section 4.2.2. The third step is to infer dependencies by recognizing similarities between operator parameters (*inferred* dependencies). This method is described in Section 4.2.3. Appendix 7.2 describes a method for aligning ontologies, which is necessary for deriving flows which rely on concepts from different ontologies.

4.2.2 Deriving 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 semiformal manner. OWL-S serves as a representative of Web service specification

languages. OWL-S was chosen as the primary language of reference, for the considerable amount of research and tools associated with it. WSMO [53] and BPEL4WS [76] have adequate transformations to OWL-S, and therefore the transformation we present is applicable for these languages as well.

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-S model is transformed to an operation $OP \in \mathcal{OP}$. The input and output properties of OP 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. The certainty that the two operations can be composed is high, due to the fact that there is an empirical evidence of composition. Therefore, for any type of empirical dependency, the certainty value of the dependency is: $\gamma_D = 1$.

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 retrieval framework, as defined in this work. 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 specifies transformation patterns for OWL-S control constructs. A visual representation of the patterns is depicted in Figure 4.3.

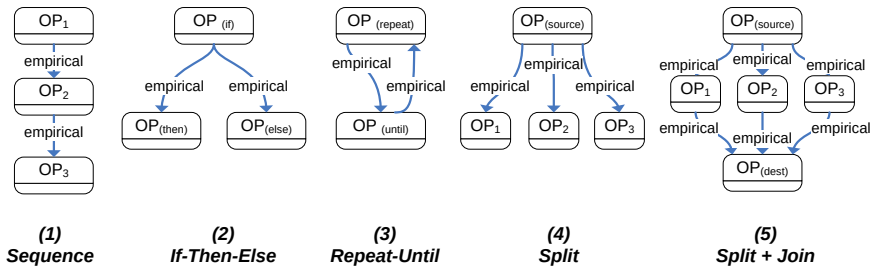


Figure 4.3: Transformation Patterns for OWL-S Control Constructs

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) and the conditioned operations: the *then* and the *else*.

3. **repeat-until**: An empirical dependency is 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 (see Section 5.1.2).

4. **split**: For each split construct, a special operation, $OP_{(source)}$ is 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 $OP_{(source)}$).

4.2.3 Inferring Dependencies

Inferred dependencies are used as a mean for representing relations between operations which are not related a-priori in the original OWL-S description. These dependencies enable compositions which are based on operations from different services, a core feature of service composition. The inference process starts with a set $\mathcal{OP}$ of operations which originates form copying the OWL-S operation properties. The process goes through all possible pairs of operations, trying to establish similarity between the *input and precondition parameters* of the first operation and the *output and effect parameters* of the second. For the sake of simplicity, we refer only to input and output parameters. If the similarity is positive, and higher than a threshold $\hat{\mu}$, then the relation between the operations is represented by a directed edge in the service network, with a given certainty, represented by the certainty value γ_D.

We relax the original definition dependency inference 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. For example, in Figure 4.2, the operation *inform hospital* receives three parameters, while inferred dependency can be derived on the basis of a single parameter (*i.e., hospital*). In

calculating the certainty, we give higher certainties to dependencies that satisfy as much of the input parameters of the operation.

The calculation of the certainty is done by comparing the parameters of each pair of operations. Given two operations, a source operation OP_s and destination operation OP_d, we define the set of output parameter concept of OP_s as $out_1^s, out_2^s, \ldots, out_n^s$ and the set of output parameter concepts of OP_d as
$in_1^d, in_2^d, \ldots, in_m^d$. We define a dependency between the operations as (OP_s, OP_d). We define a *feasible pair* of parameters $p_f = (out_s^i, in_d^j)$ as a pair that contains an output parameter from the source operation and an input parameter from the destination operation which has a similarity value higher than the threshold $\hat{\mu}$. As our similarity function is defined on operations rather than on concepts, we define a special similarity function, $simp$ that evaluates the similarity on an operation that contains the parameter. As there might be several pairs that contain the parameters, we chose the one with the maximal certainty that still :

$$p_f(i,j) = \text{argmax}_{i,j} simp(out_s^i, in_d^j)$$
$$\text{s.t. } simp(out_s^i, in_d^j) > \hat{\mu}$$

We denote by FP the set of all feasible pairs with maximal certainty. We then calculate the certainty value of the dependency:

$$\gamma_D(OP_s, OP_d) = \frac{1}{n} \sum_{\forall p_f(i,j)} simp(out_s^i, in_d^j)$$

Dependencies are added to the service network if, and only if, $\gamma_D(OP_s, OP_d) > \hat{\mu}$.

Chapter 5

Algorithms for Efficient Retrieval

This chapter describes efficient algorithms and methods for evaluating approximate service compositions. The approximate query semantics described in Chapter 4 has several implications regarding space and time complexity. For example, how different approximation methods affect processing complexity? What is the effect of threshold values for retrieval satisfiability? In this chapter we investigate these implications and others.

Section 5.1 describes data structures and algorithms for efficient query evaluation. Two concerns direct us in this section. The first is query evaluation time complexity. Our assumption is that users will use service retrieval in way similar to the way they use Internet search engines. In this mode, queries are refined through an iterative process, which requires receiving immediate results from the retrieval framework. Therefore, we discuss indexing approaches that trade time complexity with space complexity. Our second concern is the scalability of indexing methods for a large number of services. In Section 5.2 we evaluate our indexing methods with respect to precision, processing time, and scalability.

5.1 Efficient Query Evaluation

The service network data structure compactly represents operations and compositions for efficient retrieval. In this section we describe additional data structures and algorithms which allow efficient query evaluation and results ranking based on the semantics described in Section 4.1.3. The additional data structures include $I_{concepts}$, which is an index for semantic properties, and $I_{services}$, an index for compositions. We describe each of these indices in the following sections. Finally, algorithms for efficient query evaluation are specified using these indices.

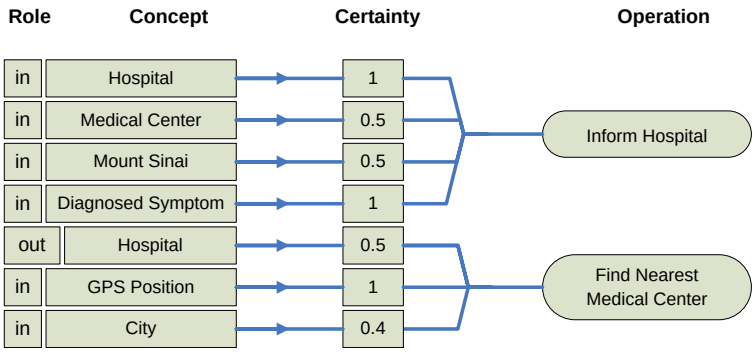


Figure 5.1: An example of $I_{concepts}$ - the concepts index

5.1.1 Concepts Index

$I_{concepts}$ is based on a hash table, where each entry represents a concept, pointing to an operation in the service network. Formally, $I_{concepts}$ induces the following function:

$$I_{concepts} : \mathcal{C} \times Props \rightarrow \mathcal{OP}$$

$\mathcal{C}$ is defined as a set of concepts, $Props = \{input, output, def, \dots\}$ is a property type (as defined in Definition 2), and $\mathcal{OP}$ is a set of operations. Each mapping is associated with a certainty function which reflects the semantic affinity between the concept and the concepts of the operation:

$$\gamma_C : I_{concepts} \rightarrow [0, 1]$$

Figure 5.1 is an instance of $I_{concepts}$, which reflects the healthcare services running example (Figure 4.2). 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 which a basic set of concepts is expanded with concepts that increase the retrieval scope of the index. The $I_{concepts}$ construction algorithm starts with the basic set of concepts derived from the

service network. We denote the concept set as $\mathcal{C}_{init}$ and define it as follows:

$$\mathcal{C}_{init} = \bigcup_{OP_i \in \mathcal{OP}, p_i \in Props} label(OP_i, p_i)$$

Where $\mathcal{OP}$ is the set of operations of the service network, and $label$ is the concept labeling function in Definition 2. Each of the concepts is mapped through $I_{concepts}$ to its designated operation. As this set is known to be related to an operation, the certainty value, γ_C is set to 1 for each of the mappings. At the second step, $I_{concepts}$ is augmented with other concepts which provide an approximate semantic correspondence, as defined in Section 4.1.3. The augmented set is denoted by $\mathcal{C}_{approx}$, and is defined as the set of concepts which include semantically related concepts. Concepts in this set are the result of some virtual operation which receives as input a concept in $\mathcal{C}_{init}$. It is formally defined as:

$$\mathcal{C}_{approx} = \{C \in \mathcal{O} | \exists \hat{C} \in \mathcal{C}_{init},\ C \rightarrow sim(C, \hat{C}) > \hat{\mu}\}$$

In order to efficiently find $\mathcal{C}_{approx}$, we first find and map all possible similarities between concepts in the ontology, and later use this mapping to produce approximations for the operations. As many operations share the same concepts, this order of calculation saves the need to recalculate concepts which are common to several operations. Another method for limiting the complexity of the construction is to search the ontology graph through a DFS (Depth First Search). As semantically similar concepts are related through the ontology, a walk through neighboring concepts can efficiently find all similar concepts. Before describing the algorithm, let us define several data structures. We define a similarity mapping function:

$$conceptRelation : \mathcal{C}_{init} \rightarrow \mathcal{C}_{approx}$$

The certainty values are expressed using another function:

$$simRelation : conceptRelation \rightarrow [0, 1]$$

The first *for* loop in Algorithm 5.1 goes through all the concepts of the ontology and populates the semantic approximation mapping, $conceptRelation$. *Recursive Search* Algorithm is specified next, representing a DFS on the ontology graph. After the mapping has been obtained, the next *for* loop goes through all the operations, and all the concepts in $\mathcal{C}_{init}$ that are mapped to the given operation. The approximated concepts for each operation OP_i are accessed through $\tilde{\mathcal{C}}_i$, and added to the $I_{concepts}$ index. The $\rightarrow$ indicates

a mapping a concept pair to a certainty value.

Algorithm 5.1 Operation indexing in $I_{concepts}$

for all $C_r \in \mathcal{C}_{init}$ **do**
  Recursive Search(C_r)
end for
for all $OP_i \in \mathcal{OP}$ **do**
  $\mathcal{C}_i = \bigcup_{p_i \in Props} label(OP_i, p_i)$
  $\tilde{\mathcal{C}}_i = \bigcup_{p_i \in Props} conceptRelation(C)$
  for all $C \in \tilde{\mathcal{C}}_i$ **do**
    $I_{concepts} = I_{concepts} \cup \{(C, p_j) \rightarrow OP_i\}$
  end for
end for

Algorithm 5.2 Recursive Search

Input: C_r - a root concept, C_n - the node concept
if $C_n = \emptyset$ **then**
  $C_n = C_r$
end if
for all C related to C_n in the ontology $\mathcal{O}$ **do**
  if $(C_r, C) \notin conceptRelation$ **then**
    if $sim(C_r, C) > \hat{\mu}$ **then**
      $conceptRelation = conceptRelation \cup (C_r, C)$
      $simRelation = simRelation \cup \{(C_r, C) \rightarrow sim(C_r, C)\}$
      Recursive Search(C_r, C)
      $conceptRelation = conceptRelation \cup (C_n, C)$
      $simRelation = simRelation \cup \{(C_n, C) \rightarrow sim(C_n, C)\}$
    end if
  end if
end for

In order to validity of the algorithm, we first prove that the operations in the ontology are classified to interconnected components. Given three operations, OP_x, OP_y and OP_z, if OP_x is not similar to OP_y ($sim(OP_x, OP_y) = 0$), but OP_y is similar to OP_z, then OP_x *cannot* be similar to OP_z. As can be seen in Figure 5.2, which provides a visualization of this situation, similar services form an interconnected component within a disconnected graph. If some similarity occurs between the parameters of operations that belong to an interconnected component, then the similarity is larger than zero for any connected operation. We formally define and prove this observation:

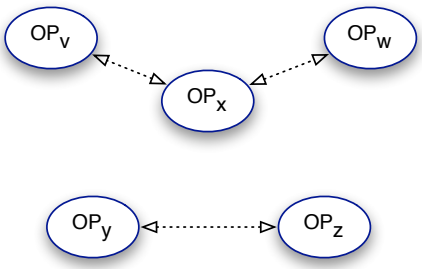


Figure 5.2: Interconnected components on the

Lemma 1: Similarity Chaining. Given the operations OP_x, OP_y and OP_z, if $sim(OP_x, OP_y) = 0$ and $sim(OP_y, OP_z) > 0$ then $sim(OP_x, OP_z) = 0$.

Proof. Let us assume that given the same setting, the conclusion is that $sim(OP_x, OP_z) > 0$. Therefore, there is some virtual operation construction $\Psi(OP_x, OP_z)$, that provides a non-negative cost. So, we can build a constructor $\Psi(OP_x, OP_y)$ that provides a non-negative cost as:

- $sim(OP_x, OP_z) > 0$ (according to the assumption)
- $sim(OP_y, OP_z) > 0$ (according to the lemma's setting)
- sim is a non-negative distance metric.

Therefore, $sim(OP_x, OP_y) > 0$, in contradiction with the lemma's settings. □

Lemma 1 helps us proving the that the DFS in the algorithm finds all the necessary similarities and with an optimal certainty. Let us define this notion formally:

Theorem 3: . Given an ontology $\mathcal{O}$ and a set of initial concepts $\mathcal{C}_{init}$:

1. All ontology concepts with over-threshold similarity are traversed.
2. The certainty is optimal.

Proof. For each concept C, if $C \in \mathcal{C}_{init}$, then it is traversed by the first loop in algorithm 5.1. If $C \notin \mathcal{C}_{init}$, then it will be traversed by algorithm 5.2. Let us assume negatively that for some concept, $C' \notin \mathcal{C}_{init}$, there exists an approximate concept $C \in \mathcal{C}_{init}$, but it not traversed by algorithm. Therefore, there must be a path of related concepts, $C, C_1, \ldots\ C_{n-1}, C'$ which was not discovered by the algorithm. However, for the second concept in the path C_1, there must be a path as proved in Lemma 1, as it is directly related

to C and $C \in \mathcal{C}_{init}$. Inductively, the rest of the path, $C_3, \ldots, C_{n-1}$ must be discovered by the algorithm as well. Therefore, there is a discoverable path to C'. The similarity between C and C' is the optimal according to the definition of the similarity function. □

5.1.2 Composition 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, 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. [64] and [63] 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 an identifier that enables an efficient routing from source to destination concepts. We also noticed some resemblance with IR clustering methods [46].

The first step in constructing the composition index is to reduce the data structure that represents the composition. We define $I_{services}$ as follows:

Definition 19: $I_{services}$ **Index**. $I_{services}$ is a labeled directed graph $I_{services} \equiv \langle \mathcal{C}, E, \gamma_S \rangle$, where:

- $\mathcal{C}$ is the set of all concepts stored in $\mathcal{C}_{approx}$.
- $E \subseteq \mathcal{C} \times \mathcal{C}$ is a set of directed relations between concepts.
- $\gamma_S : E \rightarrow [0, 1]$ is a function which maps each relation to a certainty value.

The index is built by inferring relations between concepts according to the probability that the service retrieval framework would return a composition that includes the concept. Figure 5.3 depicts an example of the way the $I_{services}$ index is created. The left side of the diagram includes an example of the service network, which is used as a basis of the index. The lightning arrow from operation OP_1 to OP_2 represents a dependency. The certainty values γ_D for dependencies and γ_C for concepts are also displayed. The right

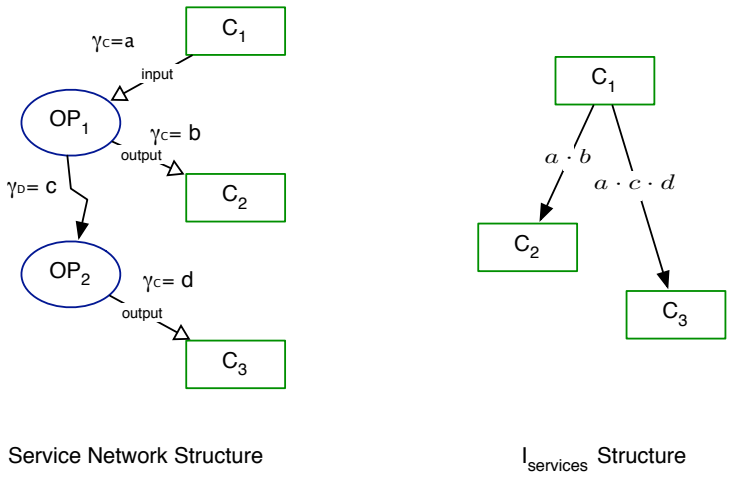


Figure 5.3: An example of the construction of $I_{services}$

side of the diagram depict a part of the service network. C_1 has a relation to C_3 because in the service network there is a path from C_1 to C_2 that goes through operations OP_1 and OP_2. According to the query semantics, the service retrieval framework would return the composition $OP_1 \rightarrow OP_2$ for a query that starts with C_1 and ends with C_3 with the certainty of $a \cdot c \cdot d$. Therefore, an edge $e = (C_1, C_2) \in E$ is constructed if *one* of the following cases occur:

- If there is exists an operation, OP_1, for which C_1 is mapped to an input concept and C_2 is mapped to the output concept. This case is exemplified in the relations between C_1 and C_2 in Figure 5.3.
- If there exists two operations, OP_1 and OP_2, for which C_1 is mapped to the input concept of OP_1, C_3 is mapped to the output concept of OP_2 and there exists a dependency between OP_1 and OP_2. This case is exemplified in the relations between C_1 and C_3 in Figure 5.3.

Figure 5.4 depicts how the service network of Figure 4.2 (page 62) is represented as by $I_{services}$.

The action of adding an edge is subject to checking the overall certainty of the edge. The certainty of an edge, $e = (C_1, C_2)$, is calculated on the basis of the certainties of the concepts and dependencies. γ_C denotes the certainty of a concept mapping and γ_D denotes the certainty of a dependency between operations. The same certainty calculations

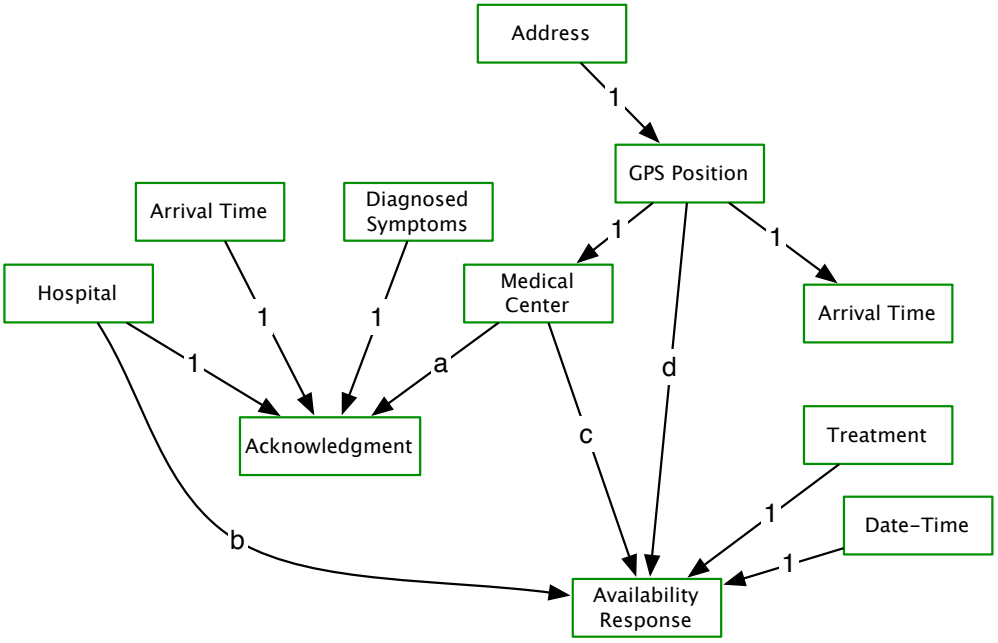


Figure 5.4: An example of $I_{services}$

are exemplified in Figure 5.3.

$$\gamma_S(e) = \begin{cases} \gamma_C(C_1) \cdot \gamma_C(C_2) & C_1, C_2 \in OP_i \\ \gamma_C(C_1) \cdot \gamma_C(C_2) \cdot \gamma_D(d) & C_1 \in OP_i, C_2 \in OP_j, d = (OP_i, OP_j) \\ 0 & \text{otherwise} \end{cases}$$

If the certainty is lower than the threshold $\hat{\mu}$ it is nullified, and the edge is not added to the index.

Theorem 4: $I_{services}$ **validity**. All possible compositions that can be achieved from the service network can be achieved from $I_{services}$.

Proof. Let us assume that there exists a composition which can be retrieved from the service network, but cannot be retrieved from $I_{services}$ with the same certainty. There are two possible options:

1. $I_{services}$ does not contain the composition. The construction process of $I_{services}$ excludes this case, as:

   - Each concept which is mapped to an operation in $\mathcal{OP}$ is also part of $I_{services}$.

- Each path in $\mathcal{SN}$ is included in $I_{services}$.

2. $I_{services}$ does contain the composition, but the certainty of path is below the threshold of the composition in the $\mathcal{SN}$. However, as the certainty of an edge in the $I_{services}$ is directly derived from the certainty of $\mathcal{SN}$ (compare the definition of γ_S (page 74) and the definition of the composition certainty (page 61).

□

One of the benefits of constructing $I_{services}$ is the reduction in the size of the composition graph. Our experimental results show that by constructing $I_{services}$, the number of edges was reduced by 36% (from 26,250 edges to 16,538 edges on a graph containing). However, the worst-case time complexity of evaluating a query over $I_{services}$ is NP-complete. As a query is a graph and $I_{services}$ is a graph, finding a composition is reducible to the problem subgraph isomorphism, which is in NP-complete in the worst-case, regarding time complexity [73].

In order to reduce the complexity, we organize the concepts to multidimensional and hierarchical clusters. Figure 5.5 visualizes the concepts, framed by the relevant clusters. Concept clusters are obtained by using the algorithm described in [40] for hierarchical clustering of OWL ontologies. All concepts within a cluster have close affinity to each other. For example, the concepts *Hospital* and *Drug* are located within the same 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., the cluster holding Address, GPS Position and Map), 1-level clusters contain 0-level clusters, etc.

The number of edges in the index is reduced by replacing dependencies between operations with dependencies between respective clusters. For example, the dependencies between *Address* and *Hospital*, and between *GPS Position* and *Hospital* are replaced by a single dependency between the their corresponding clusters. Dependencies exist only between clusters of the same level. 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, concepts are organized into multi-dimensional clusters, which reflect their different semantic affinities. For instance, the concept GPS Position has relations with geographical concepts and medical concepts. Multi-dimensional clustering is feasible as the number of parameters associated with an

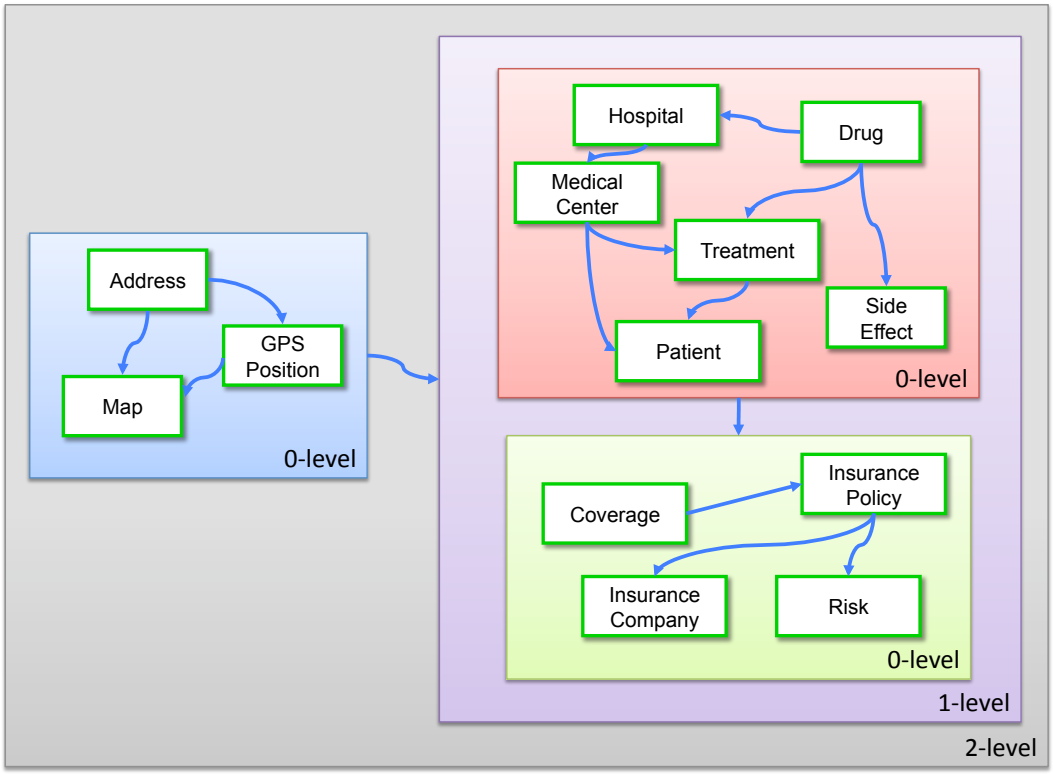


Figure 5.5: An example set of clusters

operation is bounded, and low. Empirical results show that over 90% of the services in our benchmark have 4 or less parameters. Around 60% of the services have 3 or less parameters.

5.1.3 Query Evaluation

In this section, we present an algorithm which evaluates the query against the service repository, using the indices discussed above. Given a query Q, and the indices, the algorithm returns a set of results, $\{R_{(1)}, R_{(2)}, ..., R_{(k)}\}$, ranked according to their certainty. The algorithm uses several sub-procedurs:

- $toDNF(query)$: Transforms the query into into disjunctive normal form.
- $pc(node)$: Finds the property type (input, output, etc) of a query node.
- $Prune(set)$: Takes a set of compositions and removes all compositions with μ-

Algorithm 5.3 Evaluate Query

Input: Q, $I_{concepts}$, $I_{services}$
Output: $\mathcal{R} = \{R_{(1)}, R_{(2)}, ..., R_{(k)}\}$

$\mathcal{R} \leftarrow \phi$
$C = toDNF(Q)$
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**
    $\mathcal{R} \leftarrow \mathcal{R} \cup S_{source}$
  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**
      $S_{compose} \leftarrow S_{compose} \cup Route(OP_i, OP_j, I_{services})$
      $S_{compose} \leftarrow Prune(S_{compose})$
    end for
    $\mathcal{R} \leftarrow \mathcal{R} \cup S_{compose}$
  end if
end for
Return $Rank(\mathcal{R})$

satisfiability values lower than the threshold $\hat{\mu}$.

- $Route(OP_{source}, OP_{dest}, I_{services})$: Finds a path on $I_{services}$ starting with operation OP_{source} and ending with operation OP_{dest}.

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 compositions 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 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). $|\mathcal{R}|$ 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.

Theorem 5: Query evaluation complexity. The query evaluation algorithm complexity is given by:

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

Proof. The main algorithm loop depends on the number of disjunctions, and runs in $|C|$ steps. The routing function iterates over the cartesian product of the operations returned by $I_{concepts}$. The complexity of $Route$ is calculated in [63] to be $\frac{1}{2} \log_b N$, Finally, the complexity of the ranking of the results ($|\mathcal{R}| \log |\mathcal{R}|$) is added to the general complexity. □

5.2 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 [50]; c) by evaluating the scalability of our approach through simulation. The evaluation was based on an implementation of our framework using Java and MySQL server. A dedicated personal computer running Windows XP with 1.5GB RAM and a 50GB hard disk ROM was used for all the experiments.

We used OWLS-TC, an existing benchmark for semantic service retrieval, supplied by [50]. OWLS-TC includes than 1463 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 analyzing precision values of query results. OWLS-TC was augmented with queries and relevance sets that reflect composed services.

5.2.1 Precision and Recall

Ranking serves as the main method for expressing relevance and this is also the case in our approach. Therefore, we had measured the precision of the results in the top-K places, as depicted in Figure 5.6. 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 high ranking) were found to be more relevant than services with low certainty. We explain the descent in precision around the top 3 and 4 results by the precedence of shorter services,

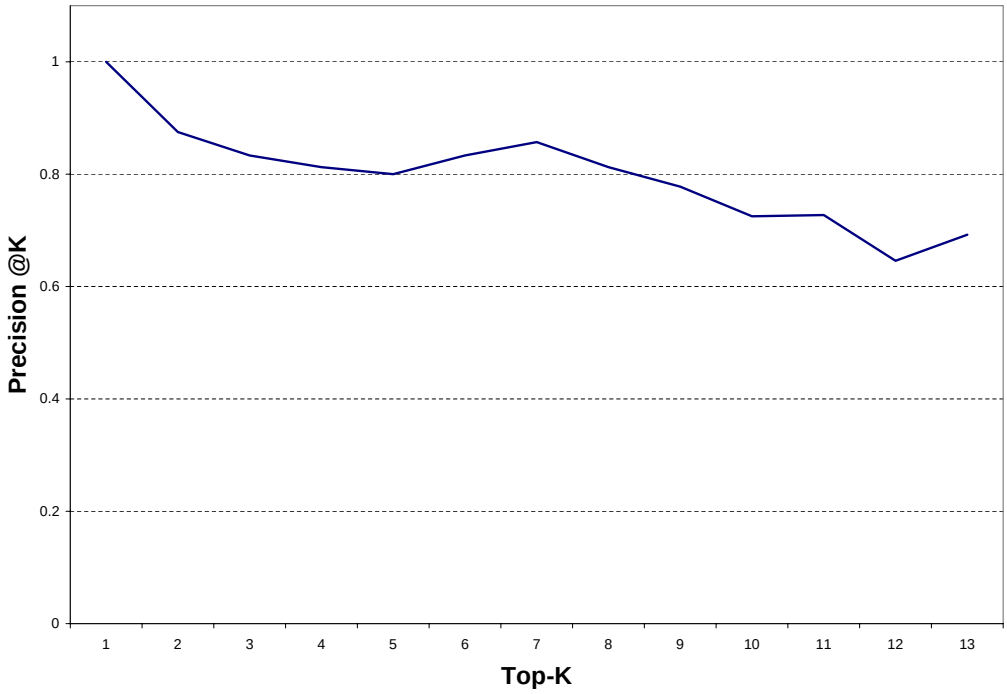


Figure 5.6: Average precision at top K

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.

In Figure 5.7 the precision and recall of our approach is compared to OWLS-MX Logic. The comparison is carried out only for queries that can be answered through subclass approximation, as OWLS-MX Logic does not support broader approximation. We adopted an evaluation strategy used by [50] which measures the average recall/precision for a given set of queries using a micro-averaging technique. Let Q be the set of test queries taken from OWLS-TC, S_q is the set of relevant services for a query $q \in Q$, and $\mathcal{S}$ as the sum of all services which answer some query in Q, such that $\mathcal{S} = \bigcup_{q \in Q} S_q$. We define by R_q all the results which were returned for a query q. For each query q, we define the set of relevant services up to a constant λ steps up to its maximum recall value, and measure the number $|B_{\lambda q}|$ of relevant services retrieved at each of these steps. We chose $\lambda = 20$ in order to be consistent with the OWLS-MX evaluation. Similarly, we measure precision with the number $|B_\lambda|$ of retrieved services at each step λ. The micro-averaging

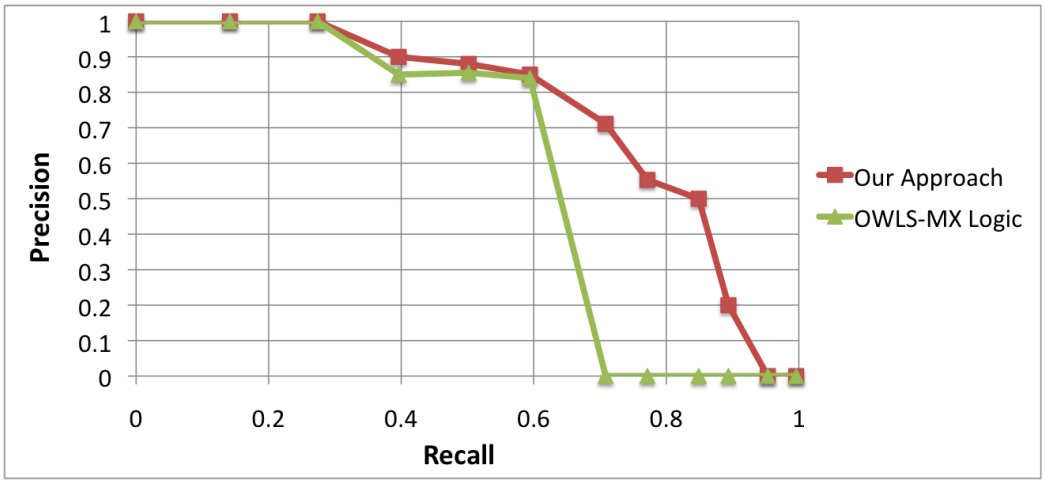


Figure 5.7: Recall-Precision comparison with OWLS-MX

of recall and precision is calculated for each of the steps, and is defined as:

$$Precision_\lambda = \frac{1}{|Q|}\sum_{q\in Q}\frac{|S_q \cap B_{\lambda q}|}{|B_\lambda|},\ \ Recall_\lambda = \frac{1}{|Q|}\sum_{q\in Q}\frac{|S_q \cap B_{\lambda q}|}{|\mathcal{S}|}$$

Each 0.1 points at the y-axis and at the x axis represents a λ step. The results show that while the two approaches behave similarly at high precision results, the logic-based approach has a sharp decline in precision for results which are have recall below 0.6. In contrast, our approach provides a gradual decline in precision, exhibiting flexible retrieval by returning results which have lower precision. While our approach returns also less precise results, our ranking mechanism ranks them below more precise results.

5.2.2 Performance and Scalability

We compared the precision/recall values of our approach with those of OWLS-MX by running the OWLS-TC queries. OWLS-MX was chosen as it exhibit a leading and characterizing approach to logic-based methods. Our results match our precision/recall performance to those of OWLS-MX. However, the two methods vary considerably in the query response time.

Table 5.2.2 presents a comparison of average response time of our approach and OWLS-MX.[1] The right hand column shows the percentage of the performance of our approach in relations to the performance of OWLS-MX. OWLS-MX is 63 times slower

[1]The average query response time we measured of OWLS-MX were slightly higher than those reported in [50]. The difference can be attributed to the different hardware configurations of the testing platforms.

than our approach on average. The results clearly show the benefits of an indexing mechanism, which improves the performance of the query evaluation algorithms by an order of one or two magnitudes.

Query	OWLS-MX	Our approach	Percentage
hospital investigating	1710	33	1.93%
book price	1647	35	2.13%
country skilled occupation	1742	20	1.15%
car price service	1682	15	0.89%
geopolitical entity weather process	1364	27	1.98%
government degree scholarship	1782	32	1.80%
novel author	1662	40	2.41%

Table 5.1: Average query response time of our approach vs. OWLS-MX (measured in ms)

The scalability of our approach was evaluated by simulating large numbers of semantic Web services. Using the OWLS-TC benchmark services as the core, 2500 additional services were simulated by imitating the properties of core services. The number of edges in the service network graph exceeded 120,000. 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 5.8 shows the average query response time as a function of the number of services in the index. The trendline represents a linear trend line on top of the discrete measurements. While the number of services increased by a factor of 6 (from 500 to 3000), the average response time increased only by a factor of 2.3 (from 15 ms to around 35 ms). The results exhibit the scalability of our indexing approach.

Figure 5.9 provide information about the same experiment from a different angle. The number of index entries grows linearly with the number of services. This is the result of bounding the similarity values by a threshold.

5.3 Implementation

In order to provide a proof-of-concept and to create a testbed for the experimental evaluation, we developed OPOSSUM (Object-PrOcedure-SemanticS Unified Matching), a Web-based search engine for Web services[2]. OPOSSUM crawls the Web for WSDL and OWL-

[2]The code is distributed under open-source license, and can be downloaded from http://projects.semwebcentral.org/projects/opossum/

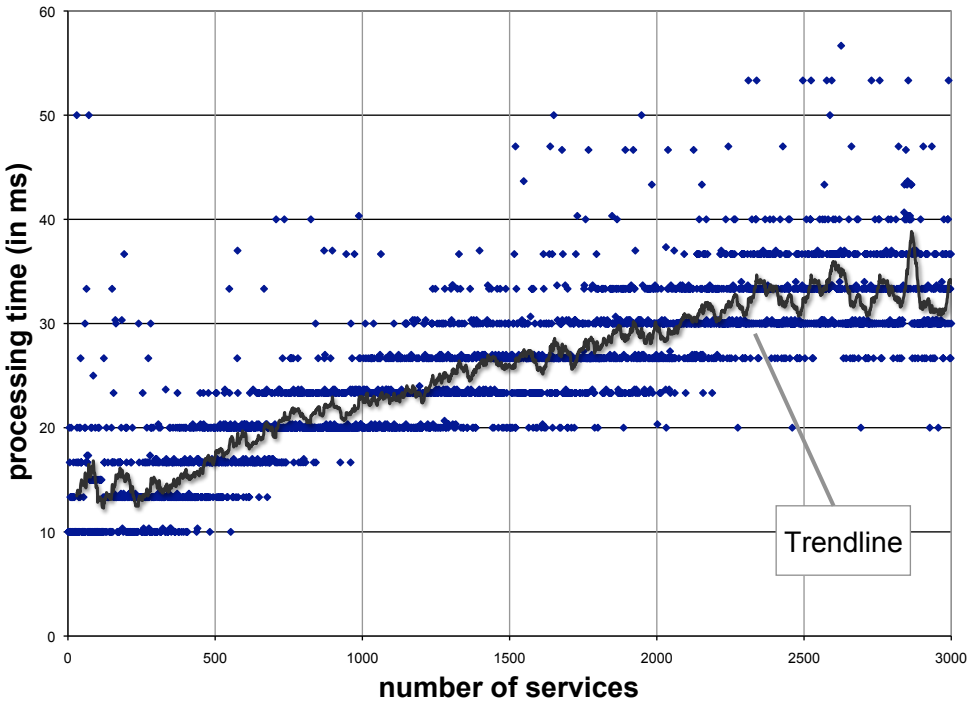


Figure 5.8: Precessing time for query evaluation

S descriptions, transforming them into ontology-based models of the Web services. It does so by automatically augmenting the service properties with existing concepts, which are collected from ontologies on the Semantic Web [12]. The following sections describe the architecture of the search engine. The implementation includes components which are not part of the research of this thesis, such as Liquid-Interface and the query parser. However, we discuss them as they illustrate the implacability of our approach.

5.3.1 Architecture

The architecture of OPOSSUM resembles the architecture of traditional, text-oriented, search engines [20], but with several notable changes: Queries and search results are expressed and represented differently. Furthermore, the index system is built to support compositions, and follows the structure defined in Section 5.1.2. Technologically, the system is based on MySQL 5.0 as a database server [45], Apache Tomcat [38] as a Web application server, and the Java programming language. Additional components includes the Jena Semantic Web Framework [61], and the OWL-S API [24].

Figure 5.10 depicts the main components of OPOSSUM, and the relations between

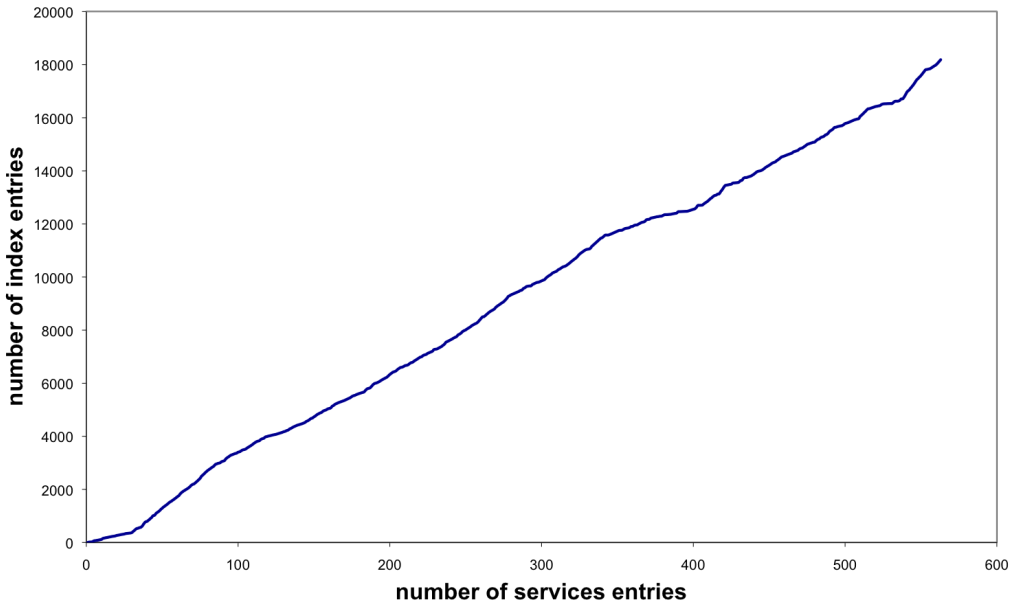


Figure 5.9: Index size as a factor of the number of services

them. The search engine allows queries entered by two modals: through a text-based query language and through a system modeling tool (OPCAT [34]). Following is a description of the main components of the system.

- **Query Parser**: When a user submits a query to a Web server, the Web server runs the *Query Parser*, which analyzes the text of the query and constructs a model of the query. An on-the-fly query analyzer matches query terms against concepts stored in the *concepts index*. The Query Parser receives queries in two forms: (1) as a simple keyword query language (similar to the one used by Google and other search engine); (2) as an OPM model created by the Opcat tool [32].

- **Query Processor**: The Query Processor receives a query model and processes it according to the algorithm described in Section 5.1.3. A screenshot of the results page is shown in Figure 5.12.

- **Crawler**: The Web crawling (downloading of Web services WSDL, OWL-S and ontology documents) is done by several distributed *crawlers*, operating offline. The crawlers download the Web content, parse it and add the content to the designated index. The crawler follows the algorithm described in Section 5.1, including the concepts index ($I_{concepts}$) and the composition index ($I_{services}$).

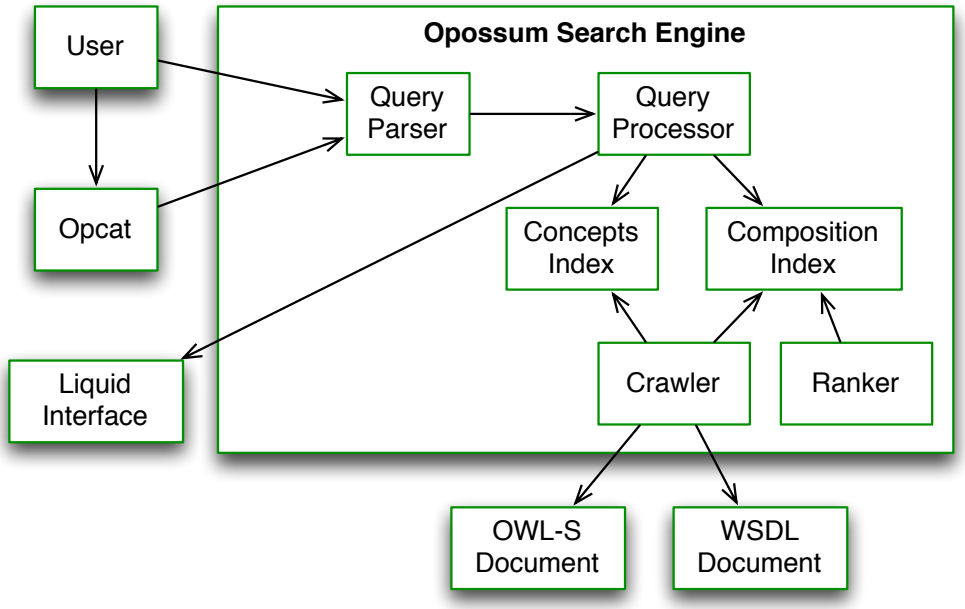


Figure 5.10: The Architecture of the OPOSSUM Search Engine

- **Ranker**: A-prioiri ranking is defined using a technique called ServiceRank. It is the equivalent of PageRank [20] in Information Retrieval. The output of ranking process is a number representing the usefulness of a service, which is general and unrelated to any specific query. The ranking is calculated in a batch mode, without considering any specific query.

  The calculation is based on the number of incoming and outgoing links to other services, and is based on a model of a "random designer", which is similar to the random surfer undermining PageRank. We imagine a designer who wants to compose a service, and chooses an arbitrary atomic service to start with. The designer then flips a coin and chooses whether to add to the composition a linked service or to abandon the composition and start over. Roughly, a useful service is a service which has high probability to be picked up by the designer, and this usefulness depends on the number of incoming links and the usefulness of the linked services. The rank is calculated on the base of the Eigenvalue of the network matrix described in Section 5.1.2.

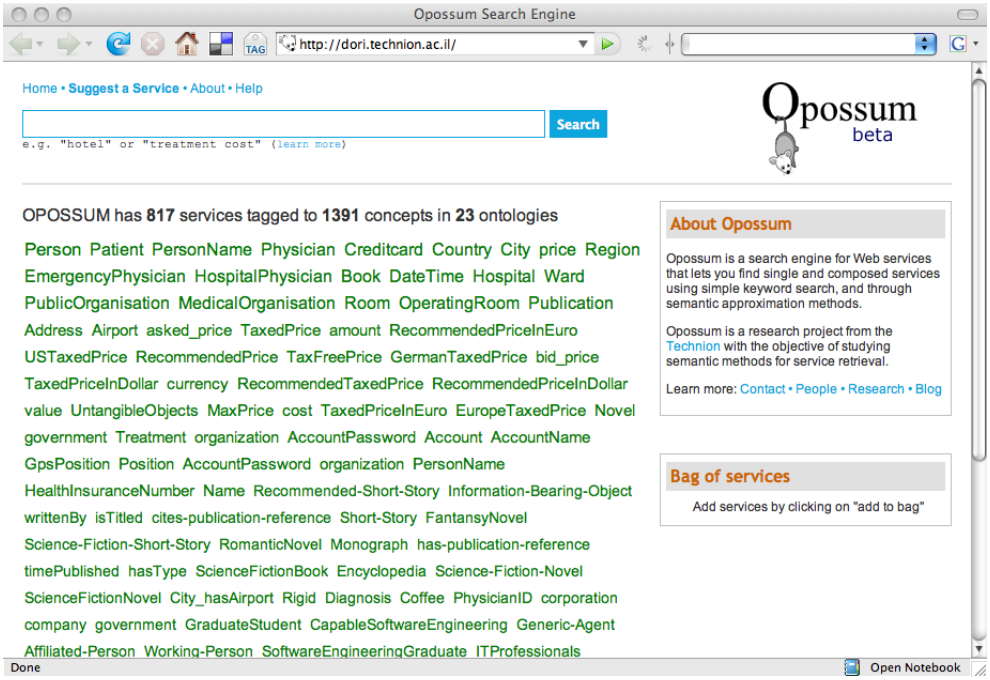


Figure 5.11: The homepage of the OPOSSUM Search Engine

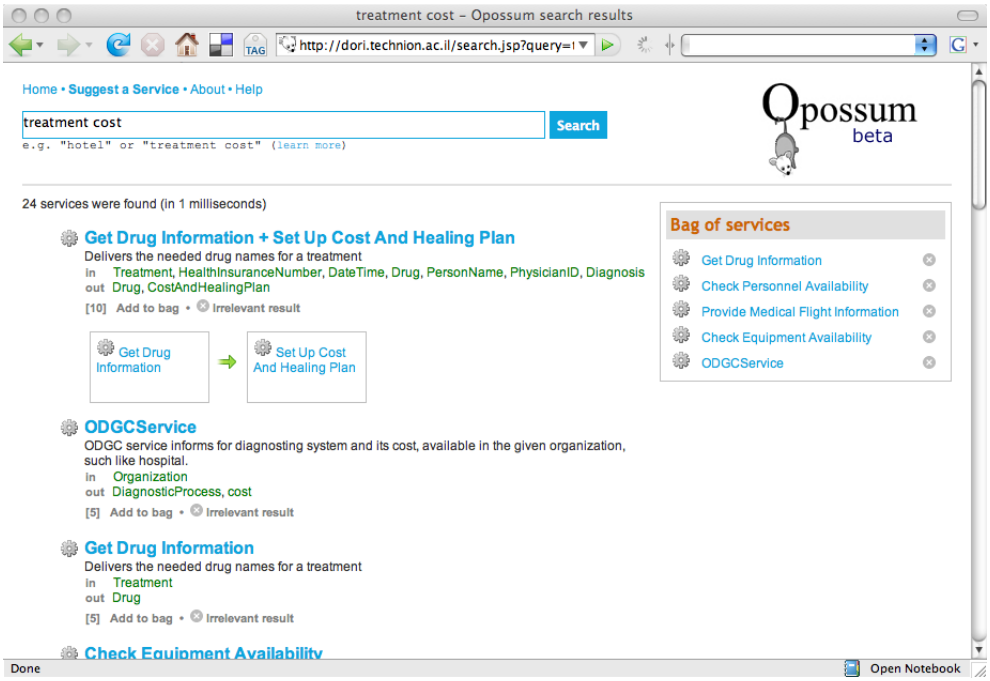


Figure 5.12: The results page of the OPOSSUM Search Engine

5.3.2 Liquid Interface

In traditional software development processes, the user interface is derived from the requirements and desired functionality of the application model. It can be carefully designed and tested in order to ensure its usability. In contrast, in dynamically composed applications, the functionality is not set during the design of the system. Therefore, the user interface cannot be designed, let alone tested for usability. The conclusion is that the user interface should be generated dynamically as well, reflecting the temporary functionality of the application. Liquid-Interface [70] is a framework for generating and optimizing graphical user interfaces (GUI) from models of semantic Web services[3].

Automatic User Interface Generation

The field of automatic generation of user interfaces attempts to formally define the elements of user interfaces, including presentation and interaction, and using the formal model in order to generate user interfaces [60, 47]. While model-based user interfaces provide the foundations for automatic generation of user interfaces, they do not deal with usability optimization as they presume the models are already usable. However, this approach will not suffice for dynamic compositions, as these compositions are not optimized for usability.

In Liquid-Interface, we provide a model of user interface generation and optimization for dynamically-composed applications. Our framework automatically generates form-based user interface, as seen in Figure 5.13, from dynamic compositions. The Framework can be used online at:
http://dori.technion.ac.il/liquidInterface. The code is distributed under open-source license, and can be downloaded from
http://projects.semwebcentral.org/projects/liquidinterface/. The composition is created by submitting queries in a natural language to a service matcher. The output of the generation process is a *prototype*: a visual presentation of a design that approximates what the final application will look and behave. The prototype then goes through a process of optimizations, in which the model is transformed according to a set of design patterns that reflect the usability of the application.

The input to the generation process is a model of dynamically-composed application, written in OWL-S. The parts of the OWL-S model which are relevant to this part of the research are the process model, which defines the execution order of the processes, and the process specification, which defines the input and output parameters of processes using

[3] Liquid-Interface was implemented by Amir Lahav and Leonid Goifman.

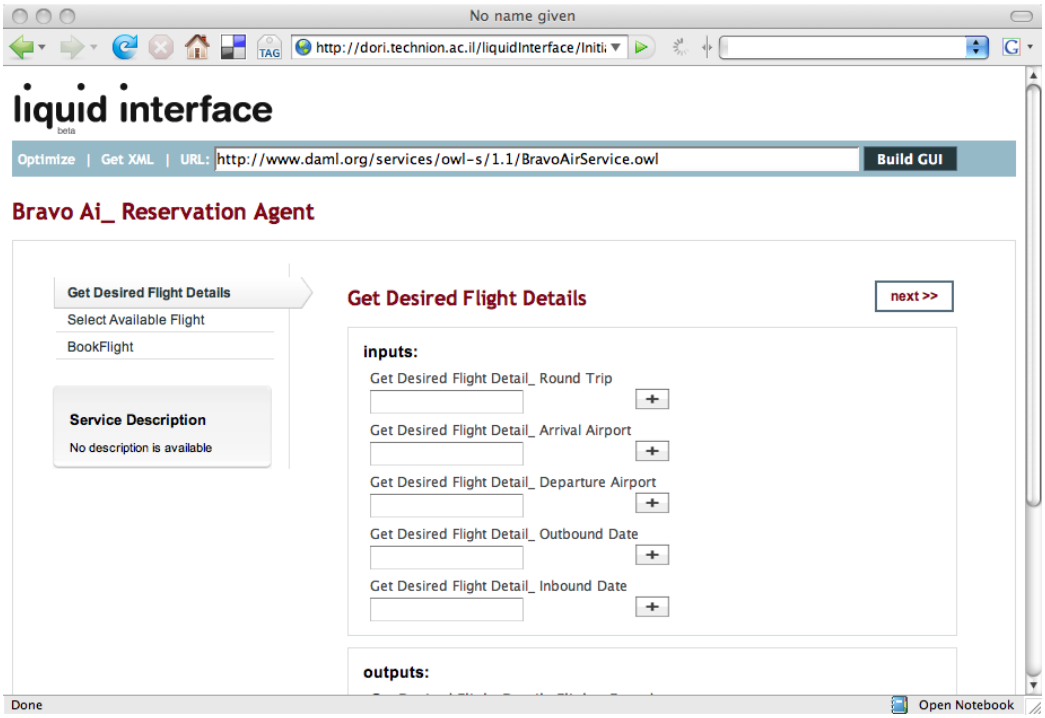


Figure 5.13: The Liquid-Interface UI Generator

ontological concepts.

The user interface generation process creates a Web-form for each sub-process, generating the form's fields from the input and output parameters. The navigation between the forms is based on the execution order of the sub-processes. For instance, if the processes are ordered in a sequential form, then the user would be able to navigate between the forms through a wizard-like fashion, using *next* and *back* buttons. If the sup-processes are ordered as parallel, the user would be able to interact with each of the processes using independent tabs.

Optimization by Model Transformation

The usability of the user interface is enhanced through two dimensions:

- Optimizing semantics: The user interface is brought nearer to the user's concepts and vocabulary by providing additional information and explanations taken from ontologies which are related to the application.
- Optimizing navigation: This optimization deals with modifying the navigation of the application with the intention of making it more usable, secure and manageable.

The semantic optimization process is based on analyzing semantic concepts, which are part of the OWL-S process specification. In OWL-S, each input and output parameter is mapped to a concept that formally defines its essence. In order to provide richer semantics to the users, these concepts are expressed using interface widgets. For example, as the *Patient* concept contains several properties, such as *name*, *insurance company* and *symptoms*, these concepts are displayed as additional fields, presented in the context of the parent field. The type of the user interface widget is adjusted to the semantic type of the concept. For example, concepts that express dates are displayed using a calendar, and concepts that have a bounded set of values (e.g. countries or currencies) are displayed as lists. Other semantic characteristics are expressed using user interface elements, including cardinality, concept generalization, multi-lingual concepts, and input validity checks.

Navigation optimization modifies the process execution order of the original OWL-S model according to a set of *user interaction design patterns*. As measures for evaluating the quality of user interface navigation are rather vague, we created a taxonomy of user interaction design patterns, selecting patterns which are relevant to navigation. For example, the *Flat and Narrow Tree* design pattern [17] defines optimal measures to link distribution between the pages. Each of the selected patterns were modeled as functions that assign a *navigational score* to a configuration of the application's navigational properties, such as the number of links between pages and the number of fields within a page. The Liquid-Interface framework includes an open architecture that allows new design patterns to be defined and added dynamically to the optimization process, in a given order.

We had tested the implementation with several different compositions from various sources and observed an improvement in the overall usability of the application. The preliminary results also reveal interesting relations between design patterns, including patterns that contradict (or enforce) each other.

Chapter 6

Conclusions

Ring the bells that still can ring
Forget your perfect offering
There is a crack in everything
That's how the light gets in.

Leonard Cohen, "Anthem"

The objective of the proposed research is to address the rigidness of current semantic service discovery and composition by defining and evaluating approximate service retrieval. As services can vary considerably in their structure and behavior, the chances of a perfect match are low. Thus, current methods are limited in their utilization with real-world services. The song "Anthem" by Leonard Cohen suggests that one should *"forget your perfect offering"* and that *"there is a crack in everything"*. We believe that this is the case with semantic Web services. Our results show that perfect compositions of real-world services are hard to achieve, and that human users can handle a considerable amount of approximation in the context of service retrieval.

In approximate service retrieval we relax some of the constrains of traditional service retrieval in order to increase the recall of the retrieval process. The research investigates two aspects: The user-perception of the retrieval process and second the properties of the approximation algorithms, including their precision and satisfiability.

6.1 Summary of Results

We identified four affinity patterns that capture the essence of similarity between compositions of service operations.

1. Set hierarchy pattern,
2. relation pattern,
3. instance-classification pattern, and
4. functional affinity pattern.

We have proved that the list of affinity pattern is complete and that they are sufficient to capture the similarity between every two compositions. While the first patterns formally defines a notion of similarity which is already discussed in the research of semantic Web service composition, the last three patterns define a new notion of similarity. Unlike logic-based similarity methods, the patterns define flexible means of similarity. Unlike text-based similarity methods, the outcome of the matching process is explainable and rooted in formal ontological structure. Therefore, our similarity measures exhibit some desired properties, such as explainability, as each similarity-based decision can be analyzed and explained to the user.

The experimental results provide a basis for evaluating the patterns. As far as we know, this is the first experiment in which paradigms of service retrieval were examined with human participants. The experiment contradict several wide spread assumptions in the literature of service retrieval. Logic-based methods assign equal importance to plug-in (more specific) and to the exact (baseline) results [49], because more specific results follow the axioms of the general results. However, according to our results, human participants perceive specific results as inaccurate like general results. Moreover, we discovered that human participants perceive a "softer" notion of similarity than the one defined by logic-based methods. Our results show that there are two contexts of service similarity that differ in their nature. The first context is based on semantic and functional comparison between the query and the composition. The second context reflects service reuse. Participants were more forgiving for inexactness in the reuse context, as the average score for usefulness was higher than that for completeness and exactness.

Introducing approximation to service retrieval has several implications to algorithmic and computational aspects. 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. Third, we analyze the complexity

of service retrieval and provide a sub-linear service retrieval algorithm in the average case. We suggest an indexing-based method for semantic and functional service properties.

Evaluating a graph-based query on a graph index is reducible to the problem of subgraph isomorphism, and is therefore NP-complete for the worst-case. However, by clustering the graph according to dependency distribution, we were able to suggest an algorithm which is sub-linear in time complexity for the average case. We show that our approach is scalable to large number of services. We show that by raising the number of services from 1 to above 3,000 (and the number of dependencies from 0 to 120,000), the average query processing time had risen from 10 milliseconds to 30 milliseconds. Our approach also outperforms a leading approach [50] in the field of service composition by a one to two orders of magnitude.

Finally, we demonstrate a proof of concept of our results by developing OPOSSUM (Object-PrOcess-SemanticS Unified Matching), a search engine for Web services which employ the methods presented in this research, including approximation based semantic and procedural similarity. The results of the search engine are transferred into another system, called Liquid-Interface, which automatically creates a prototype of the composition. Together, the two systems allow the user to search for Web services on the Web, and to immediately use a prototypes that simulates the composition functionality.

6.2 Future Directions and Open Problems

Our research can be developed in different directions for improving approximate service retrieval. Possible future research directions include the following:

- Service analysis: In order to build searchable service repositories, it is necessary to automatically analyze and index large collections of services from existing resources, programming code, Web forms and so fourth. The research will utilize probabilistic ontologies in order to represent services, and evaluate several analysis methods (i.e., textual and structural analysis algorithms) that map service properties to ontological concepts.

- Composition complexity: While our research used a heuristic-based approach to approximate composition planning, it is not a general solution which work for any type of ontology structure and any type of semantic service specification.

- Human-Computer interaction: The success of service retrieval depends on the way it is used by developers, designers and novice users. Thus, the research would

investigate how users perceive the affinity between services, the results ranking, and other aspects of service retrieval.

- Model-driven development: An interesting research direction is to merge approximate service composition with model-driven development, embedding semantic service retrieval within service development environments. As our notion of similarity is based on the mechanism of virtual operations, it is interesting to see if these virtual operations can be automatically built.

Chapter 7

Appendixes

7.1 Object-Process Methodology (OPM)

Object-Process Methodology (OPM) [32] is an integrated approach to the study and development of systems in general and information systems in particular. OPM offers a bimodal visual-lingual model representation that is both intuitive, catering for humans, and formal so machines can process it. The basic premise of the OPM paradigm is that objects and processes are two types of equally important classes of things. Figure 7.1 provides a legend for central OPM constructs. Processes in OPM can stand alone, allowing intuitive modeling of the system's behavior that involves several object classes, possibly cutting across the system's structure. Procedural links, such as *result link* and *enabling link* are used to express the behavior of processes. Structural links, including *specialization* and *characterization* are used to express structural information regarding processes and objects. The kind reader is refereed to [32] for a thorough review of OPM.

OPM is suitable for process specification, which is at the heart of Semantic Web Ser-

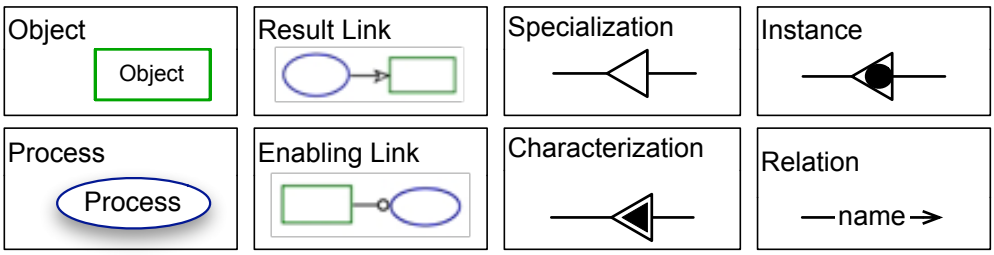


Figure 7.1: OPM Legend

vices specification. OPM-based semantic modeling and annotation has been investigated in two relevant research directions. The Visual-Semantic Web (ViSWeb) [33] enhances the Semantic Web and the RDF standard using OPM. In [35] it has been shown that OPM can be used to model expressive OWL-S-based compositions. OPM includes a notation for cardinality participation constrains. Table 7.1 defines the labels (presented in the upper row) and the minimal and maximal number of instances participating on the relation (presented in the bottom row).

Symbol	?	m	none	1	+	m..n
$q_{min}..q_{max}$	0..1	0..∞	1..1	1..1	1..∞	$m..n$

Table 7.1: Cardinality participation constraints in OPM

OPCAT [34] is a software environment that supports OPM-based modeling through a computerized environment. OPM has been found suitable as infrastructure for this research for several reasons. OPM offers a single encompassing modeling solution for the three aspects of Semantic Web Services modeling which are integrated into a single frame of reference: process, structure, and semantics. This way, OPM avoids the main drawback of UML-based semantic modeling, namely the large number of diagram types with overlapping modalities.

7.2 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.

We take a similar approach to [21], where several small ontologies are linked together and merged in order to create a superset of the ontologies. The first step in merging ontologies is to map the relations between their concepts. We have adopted the approach of Euzenat and Valtchev [36], 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.

Bibliography

[1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics-WSDL-S. Technical report, A joint UGA-IBM Technical Note, version, 2005.

[2] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop (SWWS)*, pages 411–430, July 13 2001. URL `http://www.daml.ri.cmu.edu/site/pubs/daml-s.pdf`.

[3] F. Baader, I. Horrocks, and U. Sattler. Description logics. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 3–28. Springer, 2004. ISBN 3-540-40834-7. URL `download/2004/BaHS04a.pdf`.

[4] J. Bae, L. Liu, J. Caverlee, and W. B. Rouse. Process mining, discovery, and integration using distance measures. *icws*, 0:479–488, 2006. ISBN 0-7695-2669-1.

[5] S. Bansal and J. M. Vidal. Matchmaking of web services based on the daml-s service model. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 926–927. ACM, New York, NY, USA, 2003. ISBN 1-58113-683-8.

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

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

[8] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with bp-ql. *Information Systems (In Press)*, 2008.

[9] T. Bellwood, L. Clement, D. Ehnebuske, A. Hately, M. Hondo, Y. Husband, Januszewski, K., S. Lee, M. B., J. Munter, and C. von Riegen. The UDDI version 3.0 technical report. http://www.uddi.org/, 2002.

[10] T. Berners-Lee. WWW Past and Future. W3C Presentation, 2003. URL `http://www.w3.org/2003/Talks/0922-rsoc-tbl/`.

[11] T. Berners-Lee, R. Fielding, and L. Masinter. Unified resource identifier (rfc 3986), 2005. URL `http://tools.ietf.org/html/rfc3986`.

[12] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

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

[14] A. Bernstein and C. Kiefer. Imprecise rdql: towards generic retrieval in ontologies using similarity joins. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1684–1689. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-108-2.

[15] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *Intelligent Systems, IEEE*, 18(5):16–23, Sep/Oct 2003. ISSN 1541-1672.

[16] P. V. Biron and A. Malhotra. Xml schema part 2: Datatypes. W3c recommendation, W3C, 2001.

[17] J. Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, Inc., 2001. ISBN 0471498289. Foreword By-Frank Buschmann.

[18] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and o. Y. Franc. Extensible markup language (xml) 1.0 (fourth edition). W3c recommendation, W3C, August 2006.

[19] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3c recommendation, W3C, February 2004.

[20] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998. URL citeseer.ist.psu.edu/brin98anatomy.html.

[21] A. Brogi, S. Corfini, J. F. A. Montes, and I. N. Delgado. Automated discovery of compositions of services described with separate ontologies. In *ICSOC*, pages 509–514, 2006.

[22] A. Budanitsky and G. Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics*, pages 29–34. Pittsburgh PA, June 2001.

[23] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997. ISSN 0167-8655.

[24] M. R. Center. Owl-s api. URL http://www.mindswap.org/2004/owl-s/api/.

[25] E. Christensen, F., G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. Specification document, W3C, Mar. 2001. URL http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[26] I. Constantinescu and B. Faltings. Efficient matchmaking and directory services. *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, pages 75–81, Oct. 2003.

[27] I. Corp. WebSphere Business Integration Server: http://www-306.ibm.com/software/integration/wbiserver. URL http://www-306.ibm.com/software/integration/wbiserver.

[28] O. Corp. Oracle BPEL Process Manager: http://otn.oracle.com/products/ias/bpel. URL http://otn.oracle.com/products/ias/bpel.

[29] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language wsml: An overview. Technical report, DERI, 6 2005. URL http://www.deri.at/fileadmin/documents/deri-tr-2005-06-16.pdf.

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

[31] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004.

[32] D. Dori. *Object-Process Methodology - A Holistic Systems Paradigm*. Springer-Verlag, 2002.

[33] D. Dori. ViSWeb - the visual semantic web: unifying human and machine knowledge representations with object-process methodology. *VLDB*, 13(2):120–147, 2004.

[34] D. Dori, I. Reinhartz-Berger, and A. Sturm. OPCAT - a bimodal CASE tool for Object-Process based system development. In *IEEE/ACM 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, pages 286–291, 2003.

[35] D. Dori, E. Toch, and I. Reinhartz-Berger. Modeling semantic web services with opm/s – a human and machine-interpretable language. In *World Wide Web Conference (WWW'04) Third International Workshop on Web Dynamics*, May 2004.

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

[37] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998. ISBN 026206197X. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path% =ASIN/026206197X`.

[38] A. S. Foundation. Apache Tomcat: http://tomcat.apache.org/. URL `http://tomcat.apache.org/`.

[39] A. Gal, A. Segev, and E. Toch. Semantic Methods for Service Categorization: An Empirical Study. *SDSI Workshop, VLDB Conference*, 2007.

[40] B. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Automatic Partitioning of OWL Ontologies Using E-Connections. 2005.

[41] T. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer, 1993.

[42] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems*, pages 3–15. IOS Press, 1998.

[43] J. Hau, W. Lee, and J. Darlington. A semantic similarity measure for semantic web services. In *Web Service Semantics Workshop 2005 at WWW2005*, 2005.

[44] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *J. of Web Semantics*, 1(4):345–357, 2004. ISSN 1570-8268. URL `download/2004/HoPa04b.pdf`.

[45] S. M. Inc. MySQL Database Server: http://www.mysql.com/. URL `http://www.mysql.com/`.

[46] N. Jardine and van C. J. Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7(5):217–240, 1971.

[47] D. Khushraj and O. Lassila. Ontological approach to generating personalized user interfaces for web services. In *International Semantic Web Conference*, pages 916–927, 2005.

[48] M. Klein and B. König-Ries. Coupled signature and specification matching for automatic service binding. In *ECOWS*, pages 183–197, 2004.

[49] M. Klusch. Semantic service coordination. In H. S. M. Schumacher, H. Helin, editor, *CASCOM - Intelligent Service Coordination in the Semantic Web*, chapter 4. Birkhaeuser Verlag, Springer, 2008.

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

[51] M. Klusch and A. Gerber. Evaluation of service composition planning with owls-xplan. In *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, pages 117–120. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2749-3.

[52] U. Küster, B. König-Ries, M. Stern, and M. Klein. Diane: an integrated approach to automated service discovery, matchmaking and composition. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1033–1042. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-654-7.

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

[54] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. W3c candidate recommendation, W3C, February 1999.

[55] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22 (140):1–55, 1932.

[56] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *KR*, pages 482–496, 2002.

[57] T. D. Noia, E. D. Sciascio, and F. M. Donini. Semantic matchmaking as non-monotonic reasoning: A description logic approach. *J. Artif. Intell. Res. (JAIR)*, 29:269–307, 2007.

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

[59] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3c recommendation, W3C, January 2008.

[60] A. R. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems*, 12:433–442, 1999.

[61] H. Research. Jena semantic web framework. URL `http://jena.sourceforge.net/`.

[62] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999. URL `citeseer.ist.psu.edu/resnik99semantic.html`.

[63] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup: hypercubes, ontologies, and efficient search on peer-to-peer networks. Springer-Verlag, 2002.

[64] C. Schmidt and M. Parashar. A peer-to-peer approach to web service discovery. *World Wide Web Journal*, 7(2):211–229, 2004. ISSN 1386-145X.

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

[66] E. Sirin, , J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*, April 2003. URL `http://www.mindswap.org/papers/composition.pdf`.

[67] K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002. ISSN 1387-2532.

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

[69] E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A semantic approach to approximate service retrieval. *ACM Trans. Inter. Tech.*, 8(1):2, 2007. ISSN 1533-5399.

[70] E. Toch, I. Reinhartz-Berger, A. Gal, and D. Dori. Generating and optimizing graphical user interfaces for semantic service composition. In *Proceddings of the International Conference on Conceptual Modeling*. Elsevier Science Inc., 2008.

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

[72] P. D. Turney. Similarity of semantic relations. *Computational Linguistics*, 32:379, 2006. URL `http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0608100`.

[73] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976. ISSN 0004-5411.

[74] R. Vaculin and K. Sycara. Towards automatic mediation of owl-s process models. *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1032–1039, 9-13 July 2007.

[75] T. D. Wang, B. Parsia, and J. A. Hendler. A survey of the web ontology landscape. In *International Semantic Web Conference*, pages 682–694, 2006. URL `http://www.informatik.uni-trier.de/~ley/db/conf/semweb/iswc2006.html#W% angPH06`.

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

[77] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic web services composition using shop2. In *Twelfth World Wide Web Conference*, 2003. URL `http://citeseer.ist.psu.edu/671310.html`.

[78] F. Yergeau. UTF-8 - a transformation format of unicode and iso 10646, 1998.

[79] A. Zaremski and J. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.