Automatically Grounding Semantically-enriched Conceptual Models to Concrete Web Services

Eran Toch, Avigdor Gal and Dov Dori

Technion – Israel Institute of Technology
Technion City, Haifa 32000, Israel

Abstract. The paper provides a conceptual framework for designing and executing business processes using semantic Web services. We envision a world in which a designer defines a "virtual" Web service as part of a business process, while requiring the system to seek actual Web services that match the specifications of the designer and can be invoked whenever the virtual Web service is activated. Taking a conceptual modeling approach, the relationships between ontology concepts and syntactic Web services are identified. We then propose a generic algorithm for ranking top-K Web services in a decreasing order of their benefit vis-á-vis the semantic Web service. We conclude with an extention of the framework to handle uncertainty as a result of concept mismatch and the desired properties of a schema matching algorithm to support Web service identification.

1 Introduction

Web services allow universal connectivity and interoperability of applications and services, using well-accepted standards as UDDI, WSDL, and SOAP. Current Web service standards focus on syntactic, operational details for implementation and execution rather than semantic capabilities description. A recent development enables the specification of semantic Web services. The semantic Web [5] aims to extend the World-Wide-Web by representing data on the Web in a meaningful and machine-interpretable form. The semantic Web is based on a set of languages that provide well-defined semantics and enable the markup of complex taxonomic and relations between entities on the Web. Ontologies, commonly defined as specifications of a conceptualization, [20] serve as the key mechanism for the semantic Web by allowing concepts to be globally defined and referenced. A leading language for ontology modeling for the semantic Web is the *Web Ontology language (OWL)* [9], providing a semantic markup for the definition of concept classes, relationships among them, and their instances. Several methods for annotating Web services with semantic metadata have been proposed. One of the most prominent methods is OWL-S [3]. Based on OWL, it provides an ontology for Web services, enabling a description of the service's profile, process model and its grounding - a mapping to the syntactic definition of the concrete Web service.

We envision a world in which some Web services have semantic descriptions, while others are only syntactically defined (using WSDL, for example). In particular, designers can define a "virtual" Web service as part of their business processes. An execution engine is required to look for actual Web services that match the specifications of the

designer and can be invoked whenever the virtual Web service is activated. The design of semantic Web services can be an iterative process, starting from rough design, and gradually refine the design based on feedback from some mechanism that grounds the semantic Web service to some existing Web services.

It is the aim of this paper to provide a framework for a model-driven design using semantic Web services. Taking a conceptual modeling approach, relationships between ontology concepts and syntactic Web services are identified. We then propose a generic algorithm for ranking top-K Web services in a decreasing order of their benefit vis-á-vis the semantic Web service. We conclude with a discussion on extending the framework to handle uncertainty that stems from concept mismatch and the desired properties of a schema matching algorithm to support Web service identification.

The main contribution of this work is twofold. At the conceptual level, we introduce a method for designing business processes as a composite set of Web services. At the algorithmic level, we provide a generic algorithm for ranking concrete Web services with respect to their suitability in fitting a semantic Web service description, in effect offering a model-driven approach for service-oriented computing. The semantic Web service serves as a conceptual model. Rather than generating a code out of the model, the model is implemented by locating and invoking existing services. It is worth noting that the concrete Web services are not necessarily annotated with semantic meta-data, and may be described as WSDL documents, reflecting the current state of affairs. Finally, we discuss the characterization of requirements for a schema matching algorithm should satisfy to qualify for interfacing with the Semantic Web.

The rest of the paper is organized as follows. Section 2 presents the model and formally defines the problem. The use of ontologies in ranking Web services is given in Section 3, followed by an algorithm for the matching process (Section 4). Section 5 discusses an extension to support semantic heterogeneity. Section 6 contains a related work. The paper concludes with a summary and future work (Section 7).

2 Model and Problem Definition

In this section, we provide a formal definition of the two main elements of our model, namely Web services (Section 2.1) and Semantic Web services (Section 2.2). We conclude with a formal introduction of the problem at hand (Section 2.3).

2.1 Web Services

Web services are loosely coupled software components, published and invoked across the Web. Several XML-based standards ensure the regulation of discovery and the interaction of Web services. In particular, UDDI allows Web services to be discovered through a keywords search. A Web Services Description Language (WSDL) document describes the interface and communication protocol of Web services. In this paper, we use restricted WSDL definition, ignoring namespaces, faults handling, and communication issues. Therefore, a Web service is a quadruple, $WS = (T, M, O, A)$, where:

- T is a finite set of types. A type can be primitive (*e.g.*, integer) or complex, described by an XML schema.

- M is a finite set of messages. Each message is defined by a name and a type, $t \in T$.
- O is a finite set of operations provided by the service.
- $A : M, R \rightarrow O$ is a finite set of assignments, each of which assigns a set of messages in $\{m_1, m_2, ..., m_n\} \in M$ and $R = \{input, output\}$ to an operation $o \in O$. Each message can serve as either an input or an output of the operation.

Current Web service architecture suffers from several limitations. In particular, although Web services are designed to provide distributed interoperability among applications, lack of semantic definition of these applications make the automatic integration and discovery of Web services a difficult task.

2.2 Semantic Web Services

Applying the advances of the Semantic Web to Web services, resulted in OWL-S [3]. OWL-S is a language for specifying Web service ontology, based on OWL, which augments current Web services architecture with semantic metadata. It provides a set of markup language constructs for describing the properties and capabilities of Web services, facilitating the automation of Web service tasks, including automated discovery, execution, composition and interoperation. An OWL-S ontology includes three sections, namely a profile ontology (what the service does), a process-model ontology (how it works) and a grounding ontology (how it can be used). The profile ontology extends the UDDI language, providing semantic annotation for the parameters the service accepts and provides, as well as general information describing the service.

We use as a case study, a semantic Web service named *Book Price*. The service receives a book title and a currency, locates the book's information, retrieves a price quote for it and convert it into a desired currency.[1] Figure 1 provides a visual illustration of the service using OPM/S [15], which serves as a modeling and visualization method for semantic Web services. OPM/S is an extension of Object-Process Methodology (OPM) - a conceptual object-oriented and process-oriented modeling language that supports the semantic Web [13, 14]. OPM/S models are composed of two entity types, namely services (represented as ellipses), and parameters that pass between (and possibly modified by) services, represented as rectangles. Semantics is annotated by tagging the entities with their ontological concepts in the upper-left corner of the entity.

The *Book Price* service returns the price of a book given its name and a desired currency. The service is composed of three atomic services, namely *Book Finder*, *Price Finder,* and *Currency Converter*. *Book Finder* receives a book name and produces the book information, if the book was found. *Price Finder* returns the book price in dollars, and *Currency Converter* converts the price to the desired currency. The example illustrates our notion of designing semantic Web services as an iterative process. In particular, note that the output of *Book Finder* is a rather fuzzy term of *Book Info*. As we will show later, this term is grounded in an ontology, yet it leaves the designer some maneuvering space. The designer has no particular preference at this time as to the exact form of *Book Info*, as long as it can serve as an appropriate input to *Price Finder*.

The process-model is defined as a workflow of processes, each being a quadruple $PR = (IN, OUT, E, P)$, where IN is a set of input parameters, OUT is a set of output

[1] Available at: http://www.mindswap.org/2004/owl-s/services.shtml

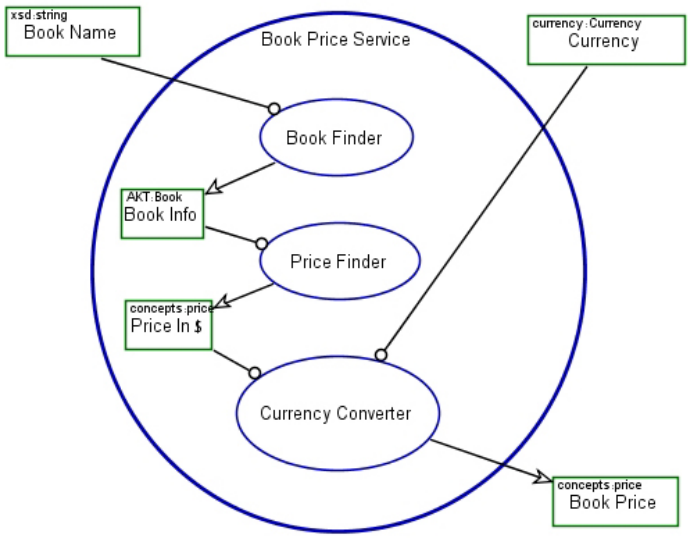

Fig. 1. Book-Price Service

parameters, E is a set of the process effects and P is a set of preconditions. Processes can be atomic or composite. Atomic processes are invoked in a single step and can be mapped directly to a WSDL operation. Composite processes, in contrast, represent a complex structure of processes. Formally, a composite process augments the process structure described above with a set of subprocesses (either atomic or composite), executed according to a certain control construct (such as parallel, sequential, conditioned, *etc*). For instance, the three subprocesses of the *Book Price Service* are executed sequentially starting from the top process. The last section of the OWL-S ontology is the grounding ontology. It provides a mapping between the atomic processes to the WSDL definition of the concrete Web service.

Elements of the profile and process-model sections, such as input and output elements, can be mapped to concepts in accompanying ontologies or to primitive XML datatypes. To illustrate this mapping, we use the AKT portal ontology [1], visualized in Figure 2 using OPM The *Book Info* parameter object in the semantic Web service (Figure 1) is mapped to a *Book* concept, described in the ontology. Given an ontology with a set of concepts C, the function $tag : IN \cup OUT \cup E \cup P \rightarrow C^*$, maps a parameter to its underlying set of concepts. A closer look at the portal ontology reveals the relationships between the *Book* concept to other concepts. The *Book* concept is a specialization of the *Publication* concept. *Publication* is characterized by *Location*, *Date* and *Title*, and is a specialization of *Information Bearing Object*.

2.3 Problem Definition

Web service discovery is a process, in which a Web service is matched based on given specifications. In this work we focus on specifications that are given as semantic Web

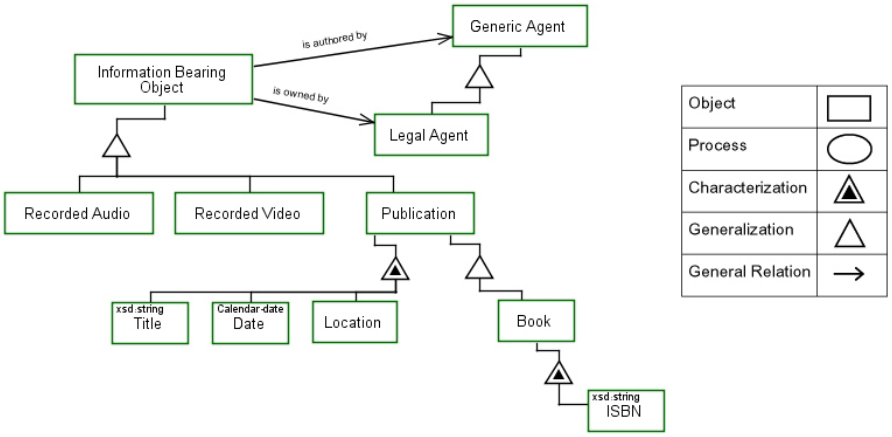

Fig. 2. The AKT Portal Ontology, visualized in OPM

services. Given an *atomic* process PR and a set $O\mathcal{P} = \{OP_1, OP_2, ...OP_p\}$ of operations within WSDL-described Web services, let $\rho_{PR} = (\preceq_{PR}, O\mathcal{P})$ be a partial order of operations, representing their relative fit for *implementing* PR. Therefore, if $OP_i \preceq_{PR} OP_j$ then OP_j is better suited to be executed as an implementation of PR than OP_i. Typically, ρ_{PR} may not be known in advance and currently a manual intervention on a grand scale may be required to ensure a selection of a suitable Web service.

In an attempt to automate the process and avoid gross errors in the discovery process, we propose the ranking of the best top-K suitable Web services, rather than providing a single Web service. Formally, given a process PR, a domain ontology ON, and a set $O\mathcal{P} = \{OP_1, OP_2, ..., OP_p\}$ of available Web services, we wish to generate a ranked mapping $O\mathcal{P}' = \{OP_{(1)}, OP_{(2)}, ..., OP_{(k)}\}$ of K Web services such that:

- $\forall i < j \leq k\ OP_j \preceq_{PR} OP_i$, and
- $\forall k < l\ OP_l \preceq_{PR} OP_k$.

3 Web Service Ranking using Ontologies

This section provides a conceptual ontology-based model for Web service ranking. Section 3.1 describes context classes and mark vectors, which are the primary tools for conceptual analysis of semantic Web services. Section 3.2 describes the method of ordering results of queries for Web services. Section 3.3 provides a detailed example.

3.1 Classification into Context Classes

We start our conceptual analysis by observing that ontologies provide a natural ranking mechanism that can be derived from the semantics of ontological constructs. Given a concept $c \in C$, annotated as the "anchor" concept, all concepts in C can be classified into one of four *context classes*, as follows:

Table 1. Classification of Semantic Relations to context classes

Context Class	OWL relations
Exact	direct mapping, owl:equivalentClass, owl:equivilantProperty, owl:sameIndividualAs, owl:ObjectProperty, owl:DatatypeProperty
Specific	owl:subClass, owl:intersectionOf, owl:oneOf, individual, $\{x \mid \forall c \in Specific, x = ObjectProperty(c) \vee x = DatatypeProperty(c)\}$
General	owl:unionOf, super-class, inverse(property), class-of $\{x \mid \forall c \in General, x = ObjectProperty(c) \vee x = DatatypeProperty(c)\}$
Negation	owl:complementOf, owl:disjointWith

Exact Concepts that have identical semantic meaning,including c itself and its properties. OWL provides relations such as *equivalentClass* to define concept equivalence.

General Concepts that supply higher-level context. For instance, the *Publication* concept is a super-class of the *Book* concept and therefore falls under the category of General with respect to *Book*.

Specific Concepts that provide a more specific context. *Book* belongs to the Specific class of *Publication*.

Negation Concepts which have explicit contradicting meaning. For instance, in OWL, if class c_1 *disjointWith* class c_2, then an instance of c_1 cannot be an instance of c_2.

While this classification is not new, careful attention should be given to properties, to which actual instance values are attached. Recall that our goal is to rank Web services according to their adequacy for the task at hand. Therefore, parameters from the semantic Web service description should be mapped to input and output messages of operations. Whenever a parameter is not grounded in a property element of an ontology, we should translate this grounding in terms of properties. Therefore, a class is represented by a subset of its properties, while a relation is represented by a subset of the properties of the class(es) with which it is associated. We should emphasize that this is not generally true, and such simplification is needed due to the simple format of WSDL. If a concept is mapped to a certain context class, its properties (both object properties and datatype properties) are added to the corresponding context class.

Given an ontology with a set of concepts C, there are $2^n - 1$ interesting combinations of concepts from C, ranging from individual concepts to a set of all concepts. Any such combination $C' \subseteq C$ is a possible CNF query to a Web service search engine. Let the response to such query be all Web services for which these concepts are considered relevant by the search engine. For example, given a query $C' \subseteq C$, Woogle [12] returns all Web services for which all concepts in C' appear in their WSDL description.

Given a concept $c \in C$, a query $C' \subseteq C$ can be mapped to a vector $mark = (e, g, s, n)$ of binary variables, representing the context classes Exact, General, Specific and Negation, respectively. A variable is assigned with a value of 1, if exists $c' \in C$ such that c' belongs to the relevant context class of c. We now consider the AKT portal ontology (Figure 2) for some examples. Assuming the anchor concept is *Book*, then the queries $\{Book\}$ and $\{Book \wedge ISBN\}$ are mapped to the mark vector $(1,0,0,0)$ since they contain concepts from the exact context class. The query $\{Book \wedge ISBN \wedge Title\}$ contains both

Table 2. Mark vectors and their classifications

Exact	General	Specific	Negation	Ranking Category
1	1	1	0	ACCURATE
1	1	0	0	
1	0	1	0	
1	0	0	0	CONTEXT-LESS
0	1	1	0	CONTEXT-ONLY
0	1	0	0	
0	0	1	0	
0	0	0	1	INVERSE
0	1	0	1	UNCERTAIN
0	0	1	1	
0	1	1	1	
1	0	0	1	
1	1	0	1	
1	0	1	1	
1	1	1	1	
0	0	0	0	EMPTY

Exact concepts (*Book* and *ISBN*) and a General concept (*Title*). Hence, it is mapped to the vector $(1,1,0,0)$.

Table 2 provides the complete mark vectors set. The right column groups the vectors into 5 ranking categories according to the matching pattern derived from the vector. For instance, the ACCURATE category contains results that originate from both direct mapping concepts and context concepts. Thus, it has the potential to produce results with high accuracy. The CONTEXT-LESS category contains results that originated from direct mapping concepts only, without a match to supporting context concepts.

3.2 Ranking of Mark Vectors

The ranking of Web services relies on ranking of mark vectors, based on the marginal benefit of the concepts that form the vector. Each context class has a different contribution to the vector. For instance, a concept that belongs to the General class provides context to the query and potentially increases its precision. On the other hand, concepts that belong to the Negation class may lead to erroneous results. A partial order between mark vectors is proposed in Figure 3. Mark vectors of the ACCURATE category are ranked higher than any other category, because their compatible queries contain both exact and contextual concepts. Specifically, it is considered more accurate than vectors of the CONTEXT-LESS and CONTEXT-ONLY categories. However, no order is determined between CONTEXT-LESS and CONTEXT-ONLY categories. Order between vectors within a category is relevant to the ACCURATE and CONTEXT-ONLY categories, and it is based on the weighted sum of matching concepts. The three top classes represent positive queries, those originating from direct and contextual concepts. The three bottom classes represent negative queries, which will retrieve empty, unfavorable or doubtful results. Queries which are assigned to the same mark vector may have an internal ranking between them, according to the number of concepts that form the query.

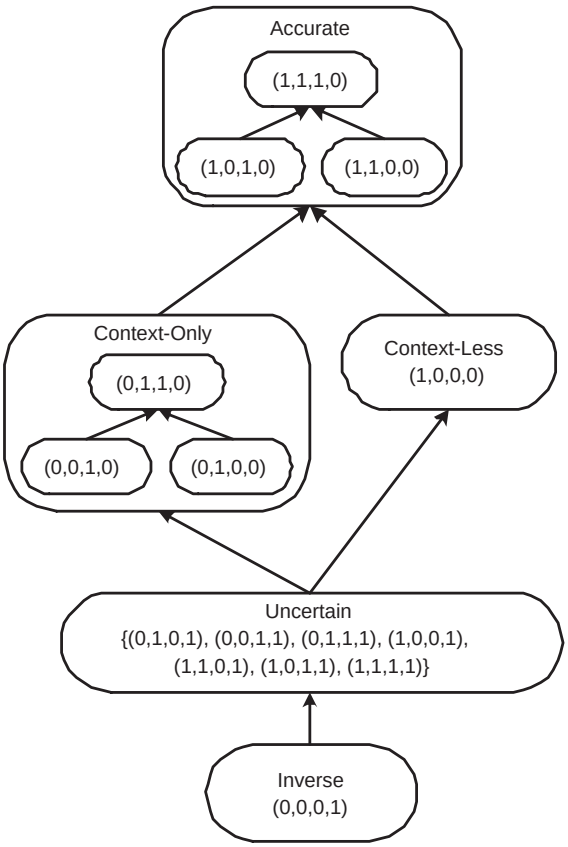

Fig. 3. Partial Order between Mark Vectors

If C_1 and C_2 are two queries, and the number of concepts in C_1 is higher than the number of concepts in C_2, then C_1 is considered more precise and therefore it is ranked higher.

3.3 Example

Consider the *Book Price* Service, as described in Figure 1, and the AKT portal ontology (Figure 2). To ground the first atomic process of the service - *Book Finder* - we describe the output parameter, *Book Info*, using a set of concepts from the portal ontology. The set C will be defined as all elements of the ontology (including properties), and the anchor concept is defined as *Book*. The context classes for *Book* are defined as follows:

- $Exact = \{Book, ISBN\}$
- $Specific = \phi$
- $General = \{Publication, Title, Date, Location, IsAuthoredBy, IsOwnedBy \ldots\}$
- $Negation = \phi$

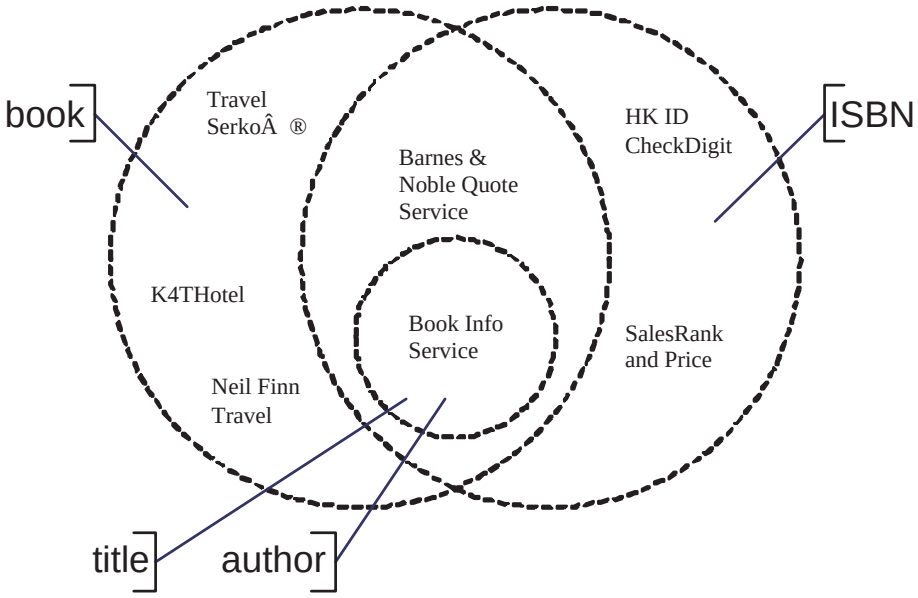

Fig. 4. Matching Results for the BookFinder Process

The next step is to build a set of queries $C' \subseteq C$. For instance, the following list contains a subset of the conjunctive normal form queries that can be constructed from concepts of C, and their compatible mark vector:

$$\{Book\}, \{Book \wedge ISBN\} \Rightarrow (1,0,0,0)$$
$$\{Book \wedge ISBN \wedge Publication\}, \{Book \wedge ISBN \wedge Publication \wedge Title\} \ldots \Rightarrow (1,1,0,0)$$

adding concepts to the query can potentially increase its precision (and decrease its recall). Therefore, each query in the list is ranked higher than the subsequent query on its left hand side. Furthermore, queries that are assigned to mark vectors of the ACCURATE class are ranked higher than ones of the CONTEXT-LESS class. Figure 4 describes the outcome of executing the queries through the Woogle search engine [12] on real Web services. The dashed circles represent sets of services that were retrieved using a single query. For instance, the following services were returned in response to the $\{Book\}$ query: *Travel SerkoA* , *K4THotel*, *Neil Finn Travel*, *Barnes & Noble Quote Service* and *Book Info Service*. These services belong to two different domains: the travel domain, where the word Book is used in the context of booking a flight, and the publication domain, which is the one we need. Intersection of this set with the set induced by $\{ISBN\}$, results in a subset of the previous set. It is worth noting that the *Book Info Service*, which is the only service that answers the requirements, is retrieved by the query $\{Book \wedge ISBN \wedge Publication \wedge Title\}$. This query is ranked higher than $\{Book \wedge ISBN\}$, so the final ranking will be:

Book Info Service $\prec_{PR}$ *Barnes & Noble Quote Service* $\prec_{PR}$ *Neil Finn Travel*...

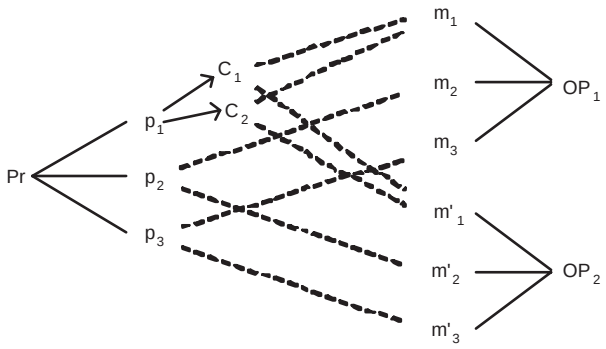

Fig. 5. Mapping between semantic processes and WSDL operations

4 Generic Ranking Algorithm

We now present a generic ranking algorithm that takes as input an atomic process *PR*, a domain ontology *ON*, and a set of WSDL-described operations $O\mathcal{P}$, and returns a set of operations $O\mathcal{P}'$, which are ranked according to their capability of implementing *PR*. The basic idea behind the algorithm is to infer affinity between a process and each operation, and use it as a measure for ranking. Affinity is derived from the mappings between process parameters $(p_1, p_2, ..., p_k)$ and operation messages $(m_1, m_2, ..., m_n)$. Such a mapping, as presented in Section 3, is provided through the ontological concepts, associated with parameters. Figure 5 visualizes this notion for parameter p_1. p_1 is represented by concepts C_1 and C_2, which are directly mapped to message m_1.

An implicit assumption in the ensuing discussion is that two operations that take the same parameters as input perform the same intended activity. Researchers have argued against this assumption [32]. Therefore, we provide a motivation for keeping it in this specific context. According to the model, the behavior of a semantic Web service is expressed using composite processes. Such flexibility allows a designer to use **atomic** processes for constructing composite ones. Therefore, one can assume that *PR* is sufficiently simple, or else it would be designed as a composite process and be further decomposed into atomic processes. While "sufficiently simple" is a vague metric, we assume that the output variance of such processes is limited, implying that the interface of the operation is sufficient for matching purposes. Moreover, Web service designers who wish external programs to use their proposed Web services are expected to develop simple, well-documented processes, again supporting our assumption.

The basic idea behind the algorithm is to use the underlying ontologies to produce a contextual matching of parameters, as defined in Section 3. The algorithm analyzes the semantic relationships between concepts and produces a ranking of the parameter matching results, which is reflected later in a ranking between the operations.

The algorithm iterates through all the parameters of a process *PR*, extracting the concepts related to that parameter. A mapping is established by procedure *MatchMessages* between each parameter and each operation message. The mapping is an assignment $Mapping : (message, parameter) \rightarrow [0, 1]$, assigning a *matching score* to each

Algorithm 1 Rank Operations

Input: PR, ON, $\mathcal{OP} = \{OP_1, OP_2, ..., OP_p\}$
Output: $\mathcal{OP}' = \left\{OP_{(1)}, OP_{(2)}, ..., OP_{(k)}\right\}$

$Mappings \leftarrow \phi$
$Messages \leftarrow \bigcup_{OP_i \in \mathcal{OP}} A^{-1}(OP_i)$
for all $paremeter \in PR$ **do**
  for all $c \in concept(parameter)$ **do**
    $Concepts \leftarrow (c, \text{“}direct\text{“})$
    $Concepts \leftarrow (related(c, ON), \text{rtype})$
  end for
  $ParameterMapping \leftarrow MatchMessages(Concepts, Messages)$
  $Mappings \leftarrow Mappings \cup (parameter, ParameterMapping)$
end for
for all $OP_i \in OP$ **do**
  $\mathcal{OP}' \leftarrow \mathcal{OP}' \cup OP_i$
  for all $OP_{(j)} \in \mathcal{OP}'$ **do**
    $OP_{(j)} \preceq_{PR} OP_{(i)} \Leftrightarrow score(OP_{(j)}) \geq score(OP_{(i)})$
  end for
end for

message-parameter pair. The overall ranking of the operation set is calculated by the function $score(OP)$, which is described below. Finally, a ranking between the operations is established according to the matching score.

An important aspect of the algorithm is the use of related concepts. The set $Concepts$ holds concepts, extracted from the ontology. Each concept is tagged with its relation to the original concept *(i.e.*, the concept to which the parameter is tagged). For example, in the *BookPrice* process definition (Figure 1), the parameter *Book Info* is mapped directly to the concept *Book* of the AKT portal ontology (Figure 2). We define the concept *Book* as a *semantic anchor* within the ontology, and it is tagged with the *direct* tag. The $related(c, ON)$ function retrieves a list of concepts which are related to the anchor concept in the ontology ON and tags each concept according to its semantic relation type ($rtype$) to the anchor object.

After characterizing the concepts according to their semantic relation, *MatchMessages* is called in order to compute a matching score between the parameter and each of the available messages. The core of *MatchMessages* is elaborated in Section 3. Basically, it produces a mapping between the set of concepts, representing the OWL-S process parameter set, and each set of messages, specified in the interface of the WSDL operations. A precondition of the existence of any semantic correspondence between the message and the parameter is data-type equivalence between the two. The function *MatchDataType* compares the types of the parameters of the corresponding concepts to be matched. For instance, if the primitive type of the direct-mapping concept of the parameter is *xsd:string* and the primitive type of the message is *xsd:float*, then the message cannot match the parameter.

If the message passed the data-type test, the procedure applies a virtual matching function, *GenericMatch*, in order to calculate the similarity between each concept and each message. The abstract function can be implemented using string matching, linguistic similarity or schema matching techniques.[2]

The last stage of the algorithm involves the ranking of the operations according to their overall matching score. An operation $OP_{(j)}$ is ranked higher than operation $OP_{(i)}$ if it has a higher or equal score. The calculation is defined as follows:

$$score(OP) = \frac{1}{|Parameters|} \sum_{pr,m \wedge A(m)=op} Mapping(pr,m) w_{pr} \cdot \prod_{pr} h(OP,pr)$$

$$h(OP,pr) = \begin{cases} 1 \text{ if } pr \text{ is mapped by at least one message } m \in OP, \\ \quad \text{such that } Mapping(pr,m) > 0 \\ 0 \text{ otherwise} \end{cases}$$

The function averages the matching scores (*Mapping*) for each message that participate in some assignment A and each of the process parameters ($pr \in Parameters$). Parameter importance is specified using a weight w_{pr}, which is associated with each parameter. To omit operations that have only a partial mapping to the process, we use h to assign a value of zero whenever the mapping of the operation does not supply a corresponding message for each of the process parameters.

5 Web Services and Schema Matching

Schema matching is the task of matching between concepts describing the meaning of data in various data sources (*e.g.* database schemata, XML DTDs, HTML form tags, *etc.*). As such, schema matching is recognized as one of the basic operations required by the process of data integration [7]. Due to its cognitive complexity [8], schema matching has traditionally been performed by human experts [22]. As the automation level of data integration increases, the ambiguity inherent in concept interpretation is becoming a major obstacle to effective schema matching. For obvious reasons, manual concept reconciliation in dynamic environments (with or without computer-aided tools) is inefficient and at times close to impossible. Introduction of the Semantic Web vision [5] and shifts toward machine-understandable Web resources and Web services have underlined the pressing need for automatic schema matching.

Attempting to address these data integration needs, several heuristics for automatic schema matching have been proposed and evaluated in the database community (*e.g.*, see [4, 10, 21, 16, 26, 31, 18]). However, as one would expect, recent empirical analysis has shown that there is no single dominant schema matcher that performs best, regardless of the data model and application domain [17], and such schema matcher may never be found. Finally, due to the unlimited heterogeneity and ambiguity of data, none of the existing heuristics can find optimal mappings for many pairs of schemata.

Bearing these observations in mind and striving to some robustness in the matching process, an approach studied in [2, 17, 24] suggested to generate not one, but K

[2] See [18, 29, 25, 11] for examples of matching methods

top-ranked mappings, examining them (either iteratively or simultaneously) until a sufficiently effective mapping is found. In this work, we adopt this research direction and aim at extending Web service ranking to support imprecise matching. Observe that an operation that implements *PR* fully is expected to have corresponding input and output messages, which have the same semantics in spite of name differences. Therefore, matching an ontology concept with a Web service input and output parameter may carry with it a degree of uncertainty. As a simple example, consider the concept *Title*. This concept may be matched (by one or another matching algorithm) with a concept *Book Title*, and assigned a similarity of 0.5. *Title* belongs to its own Exact context class, and therefore the *e* variable in the *marks* vector should be assigned a non-negative number. A natural extension to the approach presented in this work, to support the uncertainty that stems from partial mappings, can be done by allowing each variable in the *marks* vector to accept values in $[0, 1]$, corresponding to the similarity measure as determined by the matching algorithm(s) of choice. Such a change entails a revision in the partial order among different *marks* vectors, as given in Figure 3. We defer this extension to an extended version of this work.

Even though ontology languages such as OWL provide a formal set of constructs and relations, the construction of semantic Web services and ontologies may vary significantly among designers. This difference, which may be due to the methodologies, conventions, purposes and even the "style" the designers exhibit, can greatly affect the results of the algorithm described in this work. Therefore, an analysis of the way ontological knowledge is modeled and understood by humans is necessary. Specifically, research of the implications of different relations in ontological languages such as OWL is relevant to our work. Emerging works in this issue include [6] and [30], but further research is needed in this field.

6 Related Work

Our work presents an approach for grounding descriptions of semantic (possibly virtual) Web services, which exhibit a rich conceptual model, with physical (possibly not semantic) Web services, through their flat WSDL description. In this section we discuss other efforts that tackle similar problems. Paolucci et al. [28] proposed a method for matching semantic Web services. The work uses the OWL-S profile ontology as a method for describing the capabilities of services, and proposes an algorithm that matches service requesters and advertisers. The work requires the availability of full semantic description of both service requester and service advertiser in order to perform the matching. Furthermore, it requires a common ontology, or at least two connected ontologies. Klein et al [23] have used process ontology in order to match between services. The work proposes an indexing schema for services in order to promote efficient matching of services. Similar to [28], [23] requires services to be semantically annotated and indexed before matching can be executed. Our work differs primarily in the problem definition - we are concerned with grounding services to WSDL descriptions, not with matching semantic Web services. We demonstrate that a major contribution to a matching process can be made even if ontology exists only for one side of the match.

A prominent example of an architecture that supports dynamic composition of Web services is Proteus [19]. Proteus suggests an information mediating approach towards building dynamic compositions of Web services. It exhibits a multi-level architecture, which includes wrapping existing data sources with Web services, locating Web services through attribute search, building dynamic integration plans and efficiently executing the services using compression techniques. However, this approach requires Web services to be manually annotated, using a specialized ontology, in order to be reused.

METEOR-S [29] is a framework for annotating WSDL descriptions with semantic metadata. It provides a framework for semi-automatic mapping between WSDL elements and ontological concepts. The matching algorithm is based on linguistic matching at the single concept level which is enhanced with schema-based matching between the XML schema specification of WSDL elements and the ontological structure. Our work differs in the direction of matching and in its nature. While the matching algorithm used in METEOR-S takes as input a WSDL document and matches it to an ontology, our proposed algorithm performs the opposite task: it takes an ontology-powered conceptual model and tries to match it with WSDL documents. This difference has considerable implications on the algorithm. While METEOR-S is basically a schema matching algorithm, our approach acknowledges the rarity of rich XML schemas in WSDL documents. The lack of such schemas is compensated for by projecting the ontological knowledge onto keyword queries.

Our work is also related to efforts for developing matching algorithms for non-semantic Web services. Woogle [12] is a search engine for Web services. A similar goal is shared by [27], which uses a different algorithm. Woogle accepts keyword queries and returns results according to information in WSDL documents, including message parameters. Our proposed framework is complementary to that of Woogle in the type of queries it accepts. Our algorithm decomposes conceptual models into keywords and then uses a generic algorithm in order to match the keywords. Woogle can be used as an implementation for the latter task.

7 Conclusion

This paper describes a conceptual framework for designing composite business processes using semantic Web services, grounding them with existing (either semantic or other) Web services. A designer defines a rough draft of a semantic Web service to be searched. The system then searches for existing Web services that match the specifications and ranks them according to their fit with the proposed process design. Modeling languages such as OPM/S can be used for rapid specification of semantic Web service.

We use ontologies as the main vehicle for conveying semantics and utilize ontological constructs in the ranking process. We then propose a possible extension of the framework to handle poor Web service specifications and semantic heterogeneity. An extended version of this work will include a fully specified methodology for designing composite business processes in an environment with varying levels of semantic specifications. Other extensions of this work include efficient algorithmic solutions to the ranking problem, using pruning and indexing.

Acknowledgement

The work of Gal and Dori was partially supported by TerreGov, a European Commission 6^{th} Framework IST project and the Technion Fund for the Promotion of Research.

References

1. The akt reference ontology. http://www.aktors.org/publications/ontology/, 2002.
2. A. Anaby-Tavor. Enhancing the formal similarity based matching model. Master's thesis, Technion-Israel Institute of Technology, May 2003.
3. A. Ankolekar, D.L. Martin, Z. Zeng, J.R. Hobbs, K. Sycara, B. Burstein, M. Paolucci, O. Lassila, S.A. Mcilraith, S. Narayanan, and P. Payne. DAML-S: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop (SWWS)*, pages 411–430, July 2001.
4. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, May 2001.
6. Abraham Bernstein, Esther Kaufmann, Christoph Bu"rki, and Mark Klein. How similar is it? towards personalized similarity measures in ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*, February 2005.
7. P.A. Bernstein and S. Melnik. Meta data management. In *Proceedings of the IEEE CS International Conference on Data Engineering*. IEEE Computer Society, 2004.
8. B. Convent. Unsolvable problems related to the view integration approach. In *Proceedings of the International Conference on Database Theory (ICDT)*, Rome, Italy, September 1986. In *Computer Science*, Vol. 243, G. Goos and J. Hartmanis, Eds. Springer-Verlag, New York, pp. 141-156.
9. M. Dean, G. Schreiber, F. van Harmelen, J. Hendler, I. Horrocks, M. McGuinness, P.F. Patel-Schneider, and S. Stein. OWL web ontology language reference. Working draft, W3C, March 2003.
10. A. Doan, P. Domingos, and A.Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In Walid G. Aref, editor, *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, Santa Barbara, California, May 2001. ACM Press.
11. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the eleventh international conference on World Wide Web*, pages 662–673. ACM Press, 2002.
12. Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004.
13. D. Dori. *Object-Process Methodology - A Holistic Systems Paradigm*. Springer Verlag, 2002.
14. D. Dori. Visweb - the visual semantic web: unifying human and machine knowledge representations with object-process methodology. *VLDB*, 13(2):120–147, 2004.
15. D. Dori, E. Toch, and I. Reinhartz-Berger. Modeling semantic web services with opm/s a human and machine-interpretable language. In *Third International Workshop on Web Dynamics, WWW 2004*, New York, 2004.

16. N. Fridman Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, Austin, TX, 2000.
17. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 2004. to appear.
18. A. Gal, G. Modica, H.M. Jamil, and A. Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 2004. to appear.
19. Shahram Ghandeharizadeh, Craig A. Knoblock, Christos Papadopoulos, Cyrus Shahabi, Esam Alwagait, José Luis Ambite, Min Cai, Ching-Chien Chen, Parikshit Pol, Rolfe R. Schmidt, Saihong Song, Snehal Thakkar, and Runfang Zhou. Proteus: A system for dynamically composing and intelligently executing web services. In *ICWS*, pages 17–21, 2003.
20. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
21. B. He and K. Chen-Chuan Chang. Statistical schema matching across Web query interfaces. In *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 217–228, San Diego, California, United States, 2003. ACM Press.
22. R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 51–61. ACM Press, 1997.
23. Mark Klein and Abraham Bernstein. Towards high-precision service retrieval. *IEEE Internet Computing*.
24. G. Koifman, A. Gal, and O. Shehory. Schema mapping verification. In H. Davulcu and N. Kushmerick, editors, *Proceedings of the VLDB-04 Workshop on Information Integration on the Web*, pages 52–57, Toronto, Canada, August 2004.
25. J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, September 2001.
26. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the IEEE CS International Conference on Data Engineering*, pages 117–140, 2002.
27. Mourad Ouzzani and Athman Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004.
28. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002.
29. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Meteor-s web service annotation framework. In *Proceedings of WWW 2004*, pages 553–562, New York, NY, May 2004.
30. M. Andrea Rodríguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. Knowl. Data Eng.*, 15(2):442–456, 2003.
31. P. Rodriguez-Gianolli and J. Mylopoulos. A semantic approach to XML-based data integration. In *Proc. of the International Conference on Conceptual Modelling (ER'01)*, pages 117–132, Yokohama, Japan, 2001. Lecture Notes in Computer Science, Springer-Verlag.
32. A.M. Zaremski and J.M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.