Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

HIGH LEVEL DESIGN OF A SEMANTIC SEARCH ENGINE

CRISTESCU, S[orin] - A[lexandru]

Abstract: *The Internet consists of an increasing number of web pages and web services. The vast majority are described by human-readable HTML and WSDL, respectively, which makes them suitable to be found in syntactic searches (textual matching). Annotating them with semantic information unleashes a huge potential, providing more suitable results in the search process, which benefits both humans (in the case of web pages), but also companies (in the case of web services). This paper presents the key requirements and design decisions for a semantic search engine dealing with web services annotated with semantic information using SAWSDL (w3.org)*

Key words: *semantic search engine, semantic web*

1. INTRODUCTION

Web services are traditionally registered in a UDDI-like repository. A client needs to discover such a web service and then invoke it. In the real world this process is only used in a very limited way, mostly inside companies that control both the registration and the discovery process. This is due to the limitations of both phases, which are based on syntactic matching, rather than semantic.

We propose a registration and discovery process that would open up the usability of web services to real-world applications over the Internet. The idea is basically to have a search engine for web services, similar to a traditional search engine for web pages. However, in order for this to alleviate the limitations of the traditional search engines, we propose one based on semantic information. Our precondition is that web service descriptions (WSDL) need to be annotated with semantic concepts according to the SAWSDL standard (SAWSDL, 2007).

A traditional search engine needs to perform three basic steps in order to provide results to their consumers: crawling the web, indexing the web pages and actually searching for textual matches. A semantic search engine needs to provide the same functionality, with the notable difference that the indexing and searching are based on concepts (semantics), rather than on pure text (syntax). In this paper we present an algorithm for semantic matching of web services annotated using SAWSDL. We also present a high-level design of the semantic search engine.

In the area of semantic matching for web services, most of the research focuses on matching based on a semantic distance between the concepts. (Pukkasenung et al., 2010) proposes a matching algorithm based on the *semantic distance* for each of inputs, outputs, preconditions and effects (IOPEs). We also believe that using IOPEs gives accurate results, since we maximize the semantic information to base the matching on. At the same time, if matching of inputs and outputs is scalable to many users, extending the algorithm with preconditions and effects introduces only a small performance penalty, since the matching of the latter has less complexity than the matching of inputs or outputs. However, we propose matching IOPEs based on Tversky's model (Cardoso et al., 2008) and only use the semantic distance as a first filter, as explained in section 2. This provides more accurate results and at the same time it allows evaluating similarity of concepts defined in different ontologies.

Regarding registration, virtually all research in the area of semantic web services mentions registration of these services to UDDI by creating plug-ins that allow adding semantic capabilities and inference power to UDDI. However, the very creators of the standard – Microsoft, IBM and SAP – dropped their support for UDDI in 2007. We propose a replacement for UDDI that needs to address the main limitations of this standard: overwhelming complexity, lack of security, poor scalability, lack of good tooling, etc. Such a replacement is in fact the indexing step of a semantic search engine. Thus, semantic web services would be indexed based on their IOPEs annotated with semantic concepts. For example, if an input of a web service is annotated with the concept OrderRequest: http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest, then this concept is used in the index, much as a traditional search engine would index words. The advantage of approaching web service registration as indexing in a search engine is that we can take advantage of the work already done in the indexing area of classic search engines.

The search step then would mean matching the IOPEs of a requested web service to the advertised services based on Tversky's semantic matching model, which compares the properties of concepts.

A major difference with respect to traditional search is that in case of semantic web search we don't need all matching results, but only the best match, which is the web service to be invoked by the client who's looking for a match.

2. HIGH LEVEL DESIGN

The first important requirement to the data our search engine works with is that the WSDL descriptions of the services be annotated with semantic information. Our semantic search engine deals with services whose descriptions (WSDL files) contain IOPEs annotated with ontological concepts using SAWSDL. Our crawlers are fed with URLs pointing to semantically annotated WSDL documents. They fetch the documents and store them compressed in a repository. Each document is assigned a unique docID, derived from its URL. Importantly, the crawlers also fetch the ontology files used to annotate the WSDLs.

The indexer parses each WSDL document and creates a forward index, which keeps the relation between each docID and the hits, i.e. I, O, P, E concepts. At the same time, for each ontology referred by the IOPE concepts, the indexer involves an inference engine (Jena) to calculate the Tversky's matching values between each two concepts of that ontology. This list of matches is sorted in descending order by the matching values and is stored together with the corresponding ontology file. As we'll see later, this greatly improves the performance of the search.

Then the forward index is converted into four inverted indexes, each corresponding to I, O, P and E respectively. For each concept from IOPE, a unique wordID is generated. The indexes are sorted by this wordID.

With these preparations in place, the search process needs to return the best matching web service for a given profile. By profile we mean a set of IOPEs representing the desired characteristics of a service. The preconditions and effects are not mandatory, but the inputs and outputs are.

The search algorithm is structured on three levels:

1. First a syntactic (textual) matching is done with the IOPEs of the given profile; this is the same as what a traditional search engine would do, but it only returns the most relevant match (preferably one that covers all IOPEs);
2. If there was no result, a first kind of semantic matching is done: for each output concept in the given profile, we take each subclass in its ontology in increasing order of the semantic distance, as defined in (Pukkasenung et al., 2010); we compute the wordID for the current subclass and look it up in the corresponding index; thus, we try to find the best matches for all outputs according to the semantic distance; if at least one output can't be matched, the algorithm moves to step 3; then we do the same for inputs, but now we take the superclasses of each given input; if we have at least one input matched, we find the matches common for the outputs and the inputs; finally, we check which of these also match the preconditions and effects of the given profile, if they exist;
3. If there was no result from the previous step, more complex semantic matching is performed: for each output concept in the given profile, we take all concepts from the same ontology (except the subclasses covered at step 2) in decreasing order of Tversky's model value pre-calculated for the current output; we compute the wordID for each such concept and look it up in the corresponding index; if at least one output can't be matched, the algorithm stops and no match is returned; then we do the same for inputs; if we have at least one input matched, we find the matches common to the outputs and the inputs; finally, we check which of these also match the preconditions and effects of the given profile, if they exist.

Note that in step 1 we do a standard, textual match. This helps when an advertised web service is annotated with a concept such as:
http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest, which is also part of the required profile and where the namespace and the name of the concept match textually. This alleviates the need for a semantic match, since we can assume both the required profile and the advertised service mean the same thing.

Step 2 performs a first semantic match. Note that we need all outputs of a required profile to be matched. If any output is not matched by an advertised web service, we'd remain with a partial answer, which is not acceptable, since the outputs are used to compose and further invoke web services. As for the inputs, the requirement is less strict: we need at least an input to match, not necessarily all. In order to match an input, we use the covariance principle: if the required profile's input is a subclass of the advertised service's input, they match. Intuitively this means that if an advertised service can deal with a certain concept, it can also deal with its derived concepts, since they are just restrictions of the original concept. However, for the outputs we have the dual principle (contravariance): if an advertised service outputs a certain concept, we can never require a more restrictive version (i.e. a subclass) of that. Only a superclass of that concept will match.

Finally, if there are no matches so far, Tversky model is used, which provides a matching value for any two concepts of an ontology. We start with the best Tversky match (e.g. two concepts which differ by just a property) and then, if no match is found, we proceed with lower matches, until a certain threshold. The threshold is necessary to eliminate really bad matches and it is a parameter of the search process.

3. NON-FUNCTIONAL DESIGN

Just like a traditional search engine, a semantic one must fulfill certain performance criteria. Naturally, we can base our work on the experience of traditional search engines.

Among the non-functional requirements, availability and scalability score high. Typically they are dealt with redundancy, where data is duplicated and distributed across the nodes of a distributed hash table (DHT).

By design, disk seeks need to be avoided as much as possible, as they are expensive operations. Indexes are typically implemented as B+trees, which helps in minimizing the number of disk seeks needed to find an item.

A search engine such as Google (Brin & Page, 1998) uses a custom-made storage system for indexing its data, called Bigtable. It is a distributed storage system, designed to scale to petabytes of data across thousands of servers. For our prototype, we use Apache Hadoop Distributed File System (HDFS, 2007), an open source implementation in Java of a distributed file system designed to hold huge amounts of information and provide fast access to it.

Compression techniques should be chosen as a tradeoff between compression/decompression speed and compression ratio. The original Google implementation chose zlib to compress the crawled HTML pages. Our prototype also uses zlib to compress the ontology files, which tend to be large.

4. LIMITATIONS AND FUTURE WORK

In order to open the semantic web to real-world applications, tools are needed that facilitate matching the web services by their capabilities, composing and invoking them. For this to happen, we need a proper search engine for semantic web services. In this paper we present the high-level design of such a search engine, with the precondition that the WSDL files describing the web services be semantically annotated according to SAWSDL. Our proposal is a combination between traditional, textual (syntactic) matching and semantic matching, the latter involving semantic distance and Tversky's model to provide accurate results. While the first results are encouraging, more work needs to be done especially in matching ontologies: what happens if two concepts come from different ontologies but mean the same thing ? We need to be able to match them.

Also, note that we limit our algorithm to SAWSDL annotated descriptions. This excludes e.g. any annotations following the WSDL-S standard. It also excludes annotated RESTful services, which use their WADL descriptions. These limitations need to be addressed in a real-world semantic search engine. They are the next steps in our research.

5. REFERENCES

Brin S. & Page L. (1998). The anatomy of a large-scale hypertextual Web search engine, *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, doi:10.1016/S0169-7552(98)00110-X,pp.107-117

Cardoso J, Miller J. & Emani S. (2008), Web Services Discovery Utilizing Semantically Annotated WSDL, In: *Reasoning Web*, Baroglio et al., (Ed.), pp. 240-268, Springer-Verlag ISBN 978-3-540-85656-6, Berlin

Pukkasenung P., Sophatsathi P. & Lursinsap C. (2010). An Efficient Semantic Web Service Discovery Using Hybrid Matching, *Proceedings of the Second International Conference on Knowledge and Smart Technologies*, Chonburi,Thailand, pp. 49-53

*** (2007) http://hadoop.apache.org/hdfs/ - Hadoop Distributed File System, *Accessed on: 2011-08-11*

*** (2007) http://www.w3.org/TR/sawsdl/ - SAWSDL, *Accessed on*: 2011-08-11