Flavius Frasincar,
Geert-Jan Houben,
Philippe Thiran (Eds.)

WISM'07
Fourth International Workshop on Web Information Systems Modeling

Workshop at

CAiSE'07

The 19th International Conference on
Advanced Information Systems Engineering
Trondheim, 11-15 June, 2007

Organization

Program committee

Djamal Benslimane	France
Tommaso di Noia	Italy
Flavius Frasincar	The Netherlands
Martin Gaedke	Germany
Jaime Gomez	Spain
Volker Gruhn	Germany
Geert-Jan Houben	Belgium & the Netherlands
Ivan Jelinek	Czech Republic
Zakaria Maamar	UAE
Philippe Thiran	Belgium
Riccardo Torlone	Italy
Lorna Uden	UK

Workshop organizers

Flavius Frasincar	The Netherland
Geert-Jan Houben	Belgium & the Netherlands
Philippe Thiran	Belgium
Peter Barna	The Netherlands

Preface

Web Information Systems (WIS) are data-intensive systems that use the Web platform to present information. Due to the growing number of requirements that WIS need to fulfil, there is an increasing demand for methodologies to design these systems. Many of the WIS design methodologies use models to tackle the different aspects of these systems and thus manage their complexity.

The Web Information Systems Modeling (WISM) workshop aims to discuss some of the current issues involved in WIS modeling. This is the fourth edition of this international workshop after three successful editions organized in Luxembourg (2006), Sydney (2005), and Riga (2004). Due to the advantages that the Semantic Web has to offer for modeling, sharing, and reusing data, we decided to focus in this edition on how one can make use of Semantic Web technologies for WIS modeling.

Since the Web introduction there is an increased need to flexibly and efficiently integrate heterogeneous distributed data sources. One way to achieve integration can be realized by automatically mapping queries with the local schemas of the data sources. In case that the data model of the data sources is unknown one can build wrappers by exploiting the layout and ontological models specific to a certain domain.

In addition to the need to integrate data, there is a high need to integrate services in open P2P environments. One approach to service discovery is based on building communities of peers by matching their service descriptions. Such a matching procedure can make use of ontologies for describing the functionality and the input/output messages of services, and a thesaurus in order to allow more advanced service matching facilities.

The ability to integrate services should be complemented with flexible ways of querying. One of the most used query languages is SQL, and the standard language for Web service description is WSDL. Due to the popularity of these languages there is a need to build relational views on top of WSDL that can be subsequently queried using SQL. An object-oriented query language that supports Web services data types can be used to define relational views. These relational views can be subsequently nested yielding high level views.

With the advent of the Semantic Web there is a growing need of having Web pages annotated with metadata. Many of these Web pages are automatically generated by WIS. One way to support the Semantic Web is to build WIS that can recognize the end-user of a Web page: a human agent or software agent. For a software agent the WIS will generate the metadata describing the information that is conveyed in the current Web page.

Most of us did experience the long and tedious process of discovering articles on the Web that we need for our current research tasks. Such a process can be considerably improved by a recommender system that makes use of our research interests. In order to avoid the manual registration of user interests these recommender systems can automatically build user profiles by tracking researchers' academic activities and publications.

Most of the ontologies proposed for the Semantic Web are used for modeling static domains. For modeling dynamic domains there is a need to represent time and time-dependent aspects in ontologies. In order to be able to build such representations the current ontology language on the Web, i.e., OWL, needs to be extended with temporal primitives, and concepts able to change. The temporal primitives can be defined using a concrete domain, and the changing concepts can be elegantly specified by means of perdurants.

A lot of the information on the Semantic Web is stored as XML representations. As these representations tend to be rather large and complex, it is important to be able to efficiently query them. Given an XML query, a query processor needs to locate the results set that satisfies the query constraints. By making use of index structures and query processing strategies, one can improve the performance of the matching operation.

We do hope that some of the above issues sufficiently raised the reader's appetite to have a closer look at the articles gathered in these proceedings. We would like to thank all the authors, reviewers, and participants for their contributions making thus the organization of this workshop possible.

Flavius Frasincar
Geert-Jan Houben
Philippe Thiran
(June, 2007)

Table of Contents

F. Frasincar, G.J. Houben and Ph. Thiran (Eds.)

A Semantic Matching Approach for Making Heterogeneous Sources Interoperable

Michel Schneider, Damien Thevenet

LIMOS, Complexe des Cézeaux, 63173 AUBIERE Cedex
{michel.schneider@isima.fr, thevenetdamien@free.fr}

Abstract. Approaches to make multiple sources interoperable were essentially investigated when one are able to resolve a priori the heterogeneity problems. This requires that a global schema must be elaborated or that mappings between local schemas must be established before any request can be posed. The object of this paper is to study to what extend a mediation approach can be envisaged when none of these features are a priori available. Our solution consists in matching a query with each of the local schema. Such a solution is particularly suitable when sources are liable to evolve all the time. We are investigating this solution by considering the interoperability of heterogeneous XML sources. Local schemas are represented in the OWL language. Queries are formulated using an XQUERY-like language. Matching of names is solved by using an ontology of the domain. We have developed a prototype and conducted a number of experiments to evaluate the capacity of the approach.

Keywords : Interoperability, Heterogeneity, Mediation, Matching.

1 Introduction

The interoperability of multiple heterogeneous sources represents an important challenge considering the proliferation of numerous information sources both in private networks (intranet) and in public networks (internet). Heterogeneity is the consequence of the autonomy: sources are designed, implemented and used independently. Heterogeneity can appear for different reasons: different types of data, different representations of data, different management software packages. The interoperability consists in allowing the simultaneous manipulation of these sources so as to link the data which they contain. It is necessary to make different sources interoperable in numerous domains such as electronic business, environment, economy, medicine, genomics.

Interoperability problems occur in very different ways depending on whether sources are structured (data bases), semi-structured (HTML or XML pages), non-structured (any

file). The access interfaces also influences the possibilities of interoperability. For example two data bases can be difficult to make interoperable when they are only accessible through specific web interfaces.

One interoperability approach which has been studied for several years is based on mediation [17], [4]. A mediator analyzes the query of a user, breaks it down into sub-queries for the various sources and re-assembles the results of sub-queries to present them in a homogeneous way. The majority of mediation systems operate in a closed world where one knows a priori the sources to make interoperable. There are several advantages to this. First it is possible to build an integrated schema which constitutes a reference frame for the users to formulate their queries. Then it is possible to supply the mediator with various information which are necessary for the interoperability and particularly to resolve heterogeneity problems. The different kinds of heterogeneity to be resolved are now clearly identified: heterogeneity of concepts or intentional semantic heterogeneity; heterogeneity of data structures or structural semantic heterogeneity; heterogeneity of values or extensional semantic heterogeneity. Different solutions have been studied and experimented on to solve these problems. For example we can cite the work of [6] and [8]. From these initial investigations, very numerous works intervened to propose automatic approaches of integration of schemas. An approach was particularly investigated: the mapping of schemas. It led to the elaboration of several systems such as SEMINT, LSD, SKAT, DIKE, COMA, GLUE, CUPID. One will find analyses and comparisons of such systems in [14] or [5] or [12]. The practical aspects of the application of such systems are discussed in [1]. The role of ontologies was also investigated. In [2] and [11], the interest of ontologies for the semantic interoperability is underlined. Several approaches of integration of information based on ontologies were suggested. One will find a synthesis of it in [16]. It is necessary also to quote the work of [9] suggesting a logical frame for the integration of data. In all these works, the objective is to build a global schema which integrates all the local schemas.

When one operates in an evolutionary world where sources can evolve all the time, the elaboration of a global schema is a difficult task. It would be necessary to be able to reconstruct the integrated schema each time a new source is considered or each time an actual source makes a number of changes. To overcome these drawbacks an another approach has also been investigated : the query based approach. In this approach no global schema is required. The integration problems are solved during querying. Three main systems can be classified in this category: Information Manifold system, InfoSleuth system and Singapore system. Information Manifold [10] uses sources capabilities to determine how a query can be solved. InfoSleuth [13] is an agent-based system using ontologies for performing information gathering and analysis tasks. Singapore system [3] proposes an object language to formulate exact or fuzzy queries.

In this paper we suggest an approach which can also be considered as query-oriented. It is based on a semantic matching between the user query and each source schema. The user formulates its query by using its knowledge of the domain. Only the sources whose schemas match with the query are considered. The user query is rewritten for each of these sources according to its information capacity. These sources are then interrogated. Results are formatted and integrated. This approach offers several advantages. The rewriting process is simpler. A new source can be inserted at any time. The only obligation is to provide an adequate representation of this source.

The paper is organised as follows. In section 2 we give an overall presentation of our approach. Section 3 is devoted to the query language and section 4 to the OWL

representation of sources. In section 5 we explain the main features of our matching algorithm. Section 6 is relative to the rewriting of a query. Section 7 is devoted to some experiments with a prototype of the system. Section 8 presents a number of conclusions and perspectives.

2 Presentation of the approach

The approach which we propose does not use a global schema. The user thus formulates his query by using his implicit knowledge of the domain or by making an explicit reference to an ontology of the domain.

The matcher is the central element of the system. It receives the user query, and has the task to determine if this query can be applied to a data source (figure 1). To achieve this processing, it possesses a representation of each data source in a common formalism (we propose OWL to support this formalism, cf section 3). It must search for a correspondence between the query and each source by taking into account the terms and the structure of the query. Intuitively, so that a source can answer a query, the terms of the query must correspond to those of the source and the structure of the query must correspond to that of the source.

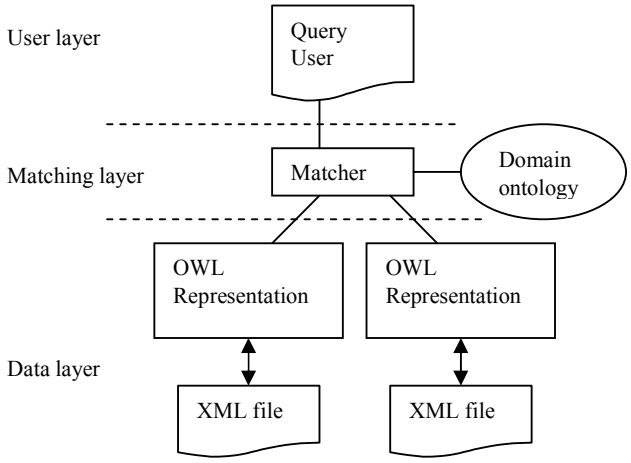

Figure 1: The architecture of our system

We propose a query language based on a simplified version of XQUERY. The structure of a query is thus defined through the various paths which appear in the clauses FOR, LET, WHERE. A correspondence is established with a source if each of these paths has a correspondent in the OWL representation of the source. More exactly, let E_1, E_2, …, E_k be a path. There is correspondence if one can find in the OWL representation classes C_1, C_2, …, C_k such that C_j is a synonym or hyponym of E_j for $j \in [1, k]$ and such that every couple of classes C_i, C_{i+1} for $i \in [1, k-1]$ is connected by a composition of properties in the OWL representation. In other words, it is necessary to find a subset of the OWL

representation which is subsumed by the path. The notions of synonym and hyponym are defined through the ontology of the domain.

For example consider the following query specified with our simplified XQUERY language :

```
Q :   for  $a in supplier, $b in customer
      where $b/name = "Ronald" and  $a/region = $b/region
      return <b> {$b}</b>
```

It looks for customers with name “Ronald” and living in the same region as a supplier. The user supposes that a customer has a name and that both a customer and a supplier have a region.

Consider the two semi-structured sources of figure 2.

It is straightforward to infer that the query matches with the first source since the supplier element and the customer element both have a son element the name of which is region. The matcher must then check that this son element occurs only once. A matching for the second source cannot be inferred so easily. First the matcher must discover that buyer is an hyponym of customer. Then it must scan the hierarchy upward in order to establish that a supplier and a buyer are both connected to a unique region. So this second source is also a candidate for a rewriting of query Q.

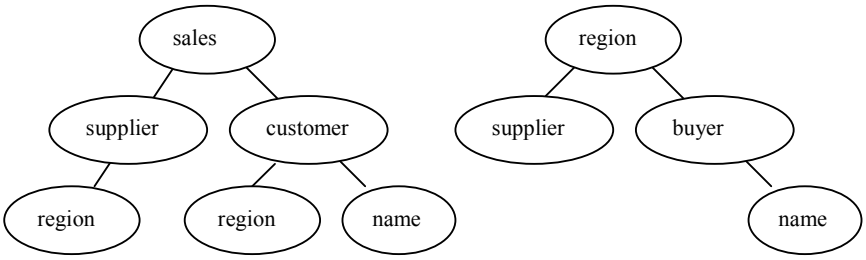

Figure 2 : Two semi-structured sources

We are now able to summarise the working of our system.

In the first phase the system initializes the connection with the ontology and gets back the names of the classes and the properties in the OWL representation. The system is then ready to handle queries.

In the second phase, when the system receives a query, it first interrogates the ontology to retrieve the synonyms and the hyponyms of the terms of the request. It then initiates the operation of matching for each of the paths of the query. Several rewriting possibilities can be proposed. To avoid inconvenient rewritings, we consider only the hyponyms of levels 1 to 3.

The third phase corresponds to the execution of one or several of these rewritings on the sources concerned. It may be necessary to transform the rewritings. This operation is performed with a wrapper associated to each source.

3 Query language

Our query language is a simplified version of XQuery. The user will have the possibility of using the FLWR construct of XQUERY with limitations indicated below.

- Since the user does not know the documents which can provide an answer, names of documents are omitted. So the root of a path is the name of an element. The system will make searches in all the documents having the root names in their description.
- Since the user does not know the structure of the data sources, it is not possible for him to decide if a term corresponds to an element or to an attribute. However he has the possibility of using the symbol '@' to indicate that he wants to search for an attribute. The system will first look for a correspondence with an attribute, but if this is not possible, it will continue its search on elements. If the symbol '@' is not present, the search will be made at the same moment on elements and attributes.
- Also, it is impossible for the user to know whether two elements are directly connected or if there are one or several intermediate elements. So it is not possible to differentiate the descent of one level "/" and the descent of several levels " // ". So the system will be responsible for testing the descent at several levels.
- The functions of XPATH are not implemented in the simplified version.
- No difference is made in the query user between lower case and upper case letters. The system will make sure the exact writing of a term is retrieved for the rewriting of the request.

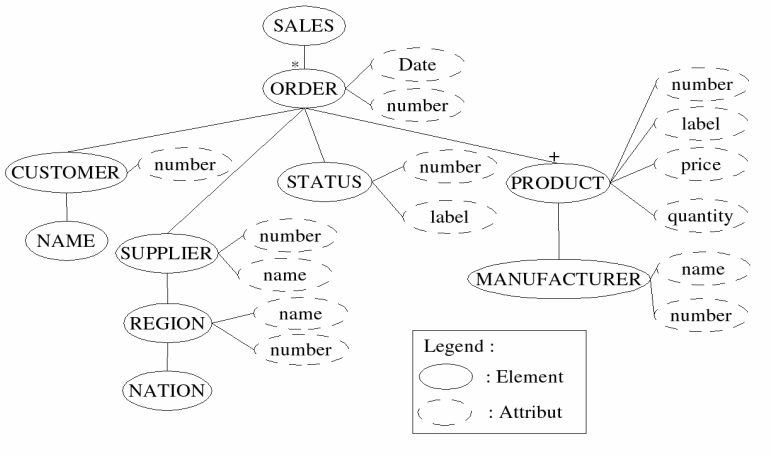

Figure 3: The semi-structured source A

4 OWL representation of the sources

To make the matching, the system could directly work on the DTD or the XML schema of the sources. We preferred to describe every source in OWL for various reasons. At first, as we want to operate with several types of sources, the matching will be implemented

with a single kind of description and will be so independent from the types of the sources. Then we foresee improved versions of the matcher, in particular by placing semantic annotations in the description of sources. The management of these annotations will be made more rationally with an OWL representation. Finally it is possible to take advantage of OWL's properties to implement the matching.

We elaborated an algorithm with which a DTD can be mapped into an OWL representation. This mapping is bijective: from the OWL representation, it is possible to regenerate the DTD. In the following we give the principles of the mapping.

The main idea is to represent every element of the DTD by an OWL class. Every father-son link between two elements is then represented by an OWL property. An attribute is also represented by a property. When a father element has only a single son element, the cardinality of this son is represented by creating a restriction on the property connecting the two elements. When the father element is a complex element, we add an intermediate class to be able to express correctly all the cardinalities.

```
<owl:Class rdf:ID="ORDER"><rdfs:subClassOf> <owl:Restriction>
<owl:onProperty rdf:resource="#ORDER.&complex1;"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:cardinality>
</owl:Restriction>  </rdfs:subClassOf> </owl:Class>
<rdf:Property rdf:ID="ORDER.&complex1;">
<rdfs:domain rdf:resource="#ORDER"/> <rdfs : range rdf:resource="#&complex1;"/>
</rdf:Property>
<owl:Class rdf:ID="&complex1;">
<rdfs:subClassOf> <owl:Restriction>
<owl:OnProperty rdf:resource="#&complex1;.CUSTOMER"/> <owl:cardinality
    rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:cardinality> </owl:Restriction> </rdfs:subClassOf>
…….
<rdfs:subClassOf> <owl:Restriction>
<owl:OnProperty rdf:resource="#&complex1;.PRODUCT"/> <owl:minCardinality
     rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:minCardinality> </owl:Restriction> </rdfs:subClassOf>
</owl:Class>
<rdf:Property rdf :ID="&complex1;.CUSTOMER">
<rdfs : domain rdf:resource="#&complex1;"/>
<rdfs:range rdf:resource="#CUSTOMER"/>
</rdf:Property>
```

Figure 4: OWL representation for the ORDER element and its sons in source A

Agreements for the names of classes and properties are as following. The class representing an element will be named with the name of the element. For an intermediate class (associated to a complex element), the name of the class will contain the names of elements with their separator, quite in brackets. When this name is long, an entity can be used. A property between two classes will carry the two names separated by a point. For attributes, the symbol '@' is used to separate the name of the class and the name of the attribute.

As an example let us consider the element ORDER of the source A, the schema of which is shown in figure 3.

In the DTD, the definition of this element is:

<!ELEMENT ORDER(CUSTOMER, STATUS, SUPPLIER, PRODUCT+)>

In order to obtain its OWL representation, a class ORDER is created and also an intermediate class the name of which is (CUSTOMER, STATUS, SUPPLIER, PRODUCT+). For clearer understanding, the entity &complexe1 is introduced to replace this name in the OWL file. Then a property connecting ORDER with the complex class is created, and the cardinality in the class ORDER is restricted. In the definition of the complex class the limitations of cardinalities are introduced for each of the elements. For CUSTOMER, STATUS and SUPPLIER, the cardinality is forced to be 1. Then properties are created to connect the complex class with each of the classes CUSTOMER, STATUS, SUPPLIER and PRODUCT (figure 4).

Using the same principles, it is possible to design an algorithm which maps a relational schema into a similar OWL representation. So our approach can be extended to deal also with relational sources.

5 Matching algorithm

We have to find a matching for each of the paths of the query.

To make the matching we represent a path as a tree having normal nodes and condition nodes. For example the path

order[customer/name="Pierre"] /product[price>15]

is represented by the tree in figure 5. In the tree, to every node is associated a simple path composed only of a succession of terms separated by the symbol "/".

The matching of a path is then made through two main functions matchSimplePath(SP) and matchPath(P).

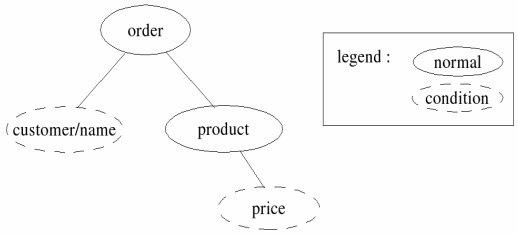

Figure 5: The tree corresponding to a path

Let SP_i be a simple path associated to node N_i. The function matchSimplePath(SP_i) looks in the OWL representation for the simple paths SP_{ij} which have a matching with SP_i. For example, let $SP_i=E_1/E_2/E_3$. A path SP_{i1} matches with SP_i if E_1, E_2, E_3 have correspondents C_1, C_2, C_3 in the OWL representation and if C_1 is connected to C_2 and C_2 is connected to C_3. E_j corresponds to C_j if E_j or an E_j's synonym or an E_j's hyponym is identical to C_j. C_1 is connected to C_2 if C_1 and C_2 are connected either by a direct or inverse property or either by a composition of direct and inverse properties.

The set of paths SP_{ij} $j \in [1, k]$ which have a matching with SP_i is associated to every node $N_i(SP_i)$. A node will thus be represented by $N_i(SP_i, SP_{ij}\ j \in [1, k])$.

The function matchPath(P) tests whether if a correct assembly of simple paths can be found which corresponds to the tree of P. Let N_i be a node of the tree and N_{i+1} one of its sons. One says that the assembly between the simple path SP_{im} of N_i and and the simple path $SP_{i+1,n}$ of N_{i+1} is correct if the last element of SP_{im} is connected with the first element of $SP_{i+1,n}$. The function matchPath(P) supplies all the possible correct assemblies for P. Each of these assemblies represents a path in the source which has a matching with P.

6 Rewritings of a query

To rewrite a query with regard to a source one looks for a rewriting of each of its paths. The rewriting of a path P_i then consists in replacing it in the query by one of the paths P_{ik} which matches with P_i and in inserting the navigation operators between the elements. When in P_{ik} one moves from a class C_1 to a class C_2 by a direct property, we only insert the descent operator // between the corresponding elements into P_i. If one moves from C_1 to C_2 by an inverse property, we insert the ascent operator.

At the end of this stage one can obtain several rewritings for a query.

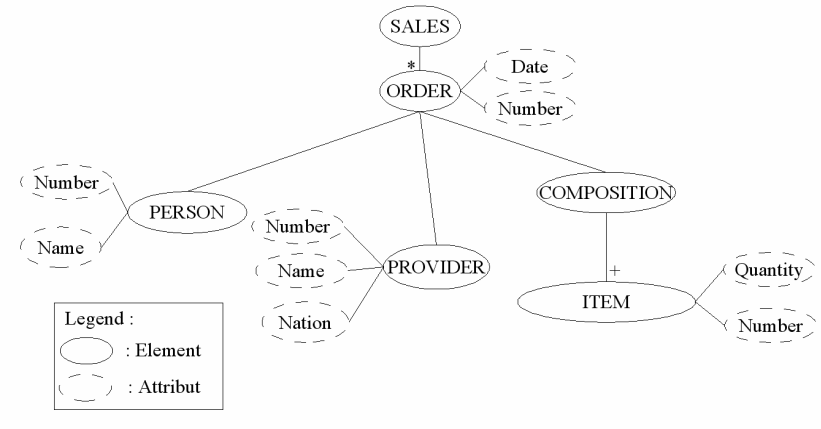

Figure 6: The semi-structured source B

7 Prototype and experiments

The prototype which we built implements the architecture presented in figure 1. We incorporated the tool SAXON-B [15] to access the OWL representations. We used the ontology WORDNET as the domain ontology. Since WORDNET is in fact a general ontology, we shall use sources for our experiments which do not contain highly

specialized terms. WORDNET is thus used only to provide synonyms and hyponyms of terms and to control that a succession of terms along a path is acceptable. Access to WORDNET is made through the JAVA API Java WordNet Library [7]. The body of the matcher is written in JAVA. We have implemented the two matching functions described in section 5.

Our experiments were conducted on source A already presented in figure 3 and on source B presented in figure 6.

We have submitted different queries to the prototype. We show the results obtained with the two sources A and B.

Query 1 : {order[customer/name="Pierre"]/product[price>15]}

Rewritings for source A:
1: {//ORDER[.//CUSTOMER//NAME = "Pierre"] //PRODUCT[./@price>15]}

Rewritings for source B:
1:

For this query the matcher proposes a correct rewriting for source A. It does not use any synonym or hyponym. No rewriting is proposed for source B.

Query 2 : {for $a in supplier where $a/nation="FRANCE" return $a}

Rewritings for source A:
1: {for $a in //SUPPLIER where $a//NATION = "FRANCE" return $a}

Rewritings for source B:
1: {for $a in //PROVIDER where $a/@Nation = "FRANCE" return $a}

In source A, NATION is an element and in source B, it is an attribute. In both cases, the matcher provides the correct rewriting.

Query 3 :
{for $a in supplier, $b in manufacturer where $a/name=$b/name and $a/nation = "FRANCE" return $a}

Rewritings for source A:
1: {for $a in //SUPPLIER, $b in //MANUFACTURER where $a/@name = $b/@name and $a//NATION = "FRANCE" return $a}

Rewritings for source B:
1:

For source B, the matcher does not provide any rewriting. For source A, it proposes a unique pertinent rewriting.

A rewriting such as:

{for $a in //SUPPLIER, $b in //MANUFACTURER where $a//REGION/@name = $b/@name and $a//NATION = "FRANCE" return $a} is also provided by our matching algorithm. This rewriting comes from the fact that there exists another attribute “name” of element REGION which can be reached from SUPPLIER. This rewriting contains strictly rewriting 1 and is not pertinent for the user. It is filtered in an additional step by using the following rule: “if a rewriting path P1 is a sub-path of another rewriting path P2 with the same starting node, delete P2 from the set of solutions”.

Query 4 : {for $a in person, $b in supplier where $a/name=$b/name return $a}

Rewritings for source A:
1: {for $a in //MANUFACTURER, $b in //SUPPLIER where $a/@name = $b/@name return $a}
2: {for $a in //CUSTOMER, $b in //SUPPLIER where $a//NAME = $b/@name return $a}
3: {for $a in //NAME, $b in //SUPPLIER where $a = $b/@name return $a}

Rewritings for source B:
1: {for $a in //PERSON, $b in //PROVIDER where $a/@Name = $b/@Name return $a}

The matcher provides three rewritings for source A since SUPPLIER, MANUFACTURER, NAME are hyponyms of PERSON (at level 3). The rewritings 1 and 2 are both pertinent and have immediate interpretation for a user. Rewriting 3 can surprise a user. It comes from the ambiguity of using in the schema of source A an element such NAME which is a hyponym of PERSON. In fact this rewriting is redundant with rewriting 2: its path is a sub-path of rewriting 2 and both terminate at the same element. So, rewritings 2 and 3 give the same result when executed on the source. We can use another filtering rule based on sub-paths. In this case we eliminate the rewriting 2 and keep only the rewriting 3.

Rewritings such as:
{$a in //MANUFACTURER, $b in //SUPPLIER where $a/@name = $b//REGION/@name return $a}
are provided by our matching algorithm. Like for query 3, there are filtered in the additional step.

Our matcher filters also rewritings such as: {for $b in //SUPPLIER, $a in //SUPPLIER where $a/@name = $b/@name return $a} which are correct but which correspond to a truth assertion and do not give pertinent results.

If we use only the hyponyms of level 1 for this query, the matcher give no answer for source A. This example shows clearly the interest of hyponyms, but also the problems which they can pose when confronting it to ambiguous schemas.

For source B the matcher provides a unique rewriting which is pertinent.

8 Conclusion and perspectives

Through the results obtained, it appears that our system is able to find data from an intuition of the user, intuition expressed through an implicit vision of the domain compatible with the ontology.

The main limitation comes from the fact that the system can propose irrelevant rewritings for a query. Most of them can be eliminated by adequate filtering rules. One can also act on the exploration depth in the ontology.

The quality of the ontology is highly important to avoid irrelevant rewritings. Ontology WORDNET used for our experiments is too general and contributes to sending back too many solutions. General terms such as name, number, … contribute to increase the number of irrelevant solutions. It would so be necessary to minimize their use in the sources schemas and to resort to more precise terms.

More elaborated solutions exist to deal with this problem. A solution which we are investigating at present consists in placing annotations in the OWL representation at the level of classes or properties. These annotations will be exploited by the matcher to take into account semantic features (sense of a term, meaning of a property). These annotations could be installed manually by the administrator of the source or automatically by the system by seeking the opinion of the users when several rewritings are possible. To help the matching one can ask the user to clarify his query if the system detects some ambiguities.

We think that these improvements could result in an efficient system.

The system can be extended to deal with other types of sources (relational, object).

The main advantage of our approach is its robustness with regard to the evolution of sources. When a new source is inserted, it is sufficient to elaborate its OWL representation so that it can be exploited by the system. When a source evolves, it is sufficient to reshape its OWL representation.

We are also engaged in another improvement of our prototype in order to allow the join of results coming from different sources. In that case a query is rewritten in several sub-queries, each sub-queries being relative to a different source. Our matching algorithm can be easily adapted for this more general situation. It is necessary to look for sub-paths in different sources and to impose a join condition between sub-paths (the terminal node of a sub-path must correspond with the start node of another sub-path).

Another important point concerns the performances of our approach. Searching in a source the compatible paths of a query is the critical stage. We investigate different solutions in order to permit the system to scale up.

Such a system can be very useful for different applications. Incorporated into an intranet system, it would allow a user to reach the data sources without knowing their schemas, by being based only on the domain ontology. In a P2P system, it could be installed on some peers or on the super-peers to facilitate access to data by their semantics. The only obligation for a peer would be to publish its data by using the OWL representation.

References

1. Bernstein P. A., Melnik S., Petropoulos M., and Quix C.: Industrial-strength schema matching. SIGMOD Record, Vol. 33, No 4, pp 38-43, 2004.
2. Cui Z., Jones D., O'Brien P.: Issues in Ontology-based Information Integration. IJCAI, Seattle, 2001.
3. Domenig R., Dittrich K.R.: A Query based Approach for Integrating Heterogeneous Data Sources. CIKM 2000, pp. 453-460.

4. Garcia-Molina H., Papakonstantinou Y., Quass D., Rajaraman A., Sagiv Y., Ullman J., Vassalos V. and Widom J.: The Tsimmis approach to mediation: Data models and languages. Journal of Intelligent Information Systems , Vol. 8, No. 2, pp. 117-132, 1997.
5. Hai Do H., Melnik S., Rahm E.: Comparison of Schema Matching Evaluations. Web, Web-Services, and Database Systems. pp 221-237, 2002.
6. Hull R.: Managing semantic heterogeneity in databases: A theoretical perspective. Proc. of the Symposium on Principles of Database Systems (PODS), Tucson, Arizona, pp. 51-61, 1997.
7. JWNL. Java WordNet Library. http ://sourceforge.net/projects/jwordnet.
8. Kedad Z., Métais E.: Dealing with Semantic Heterogeneity During Data Integration. Proc of the International Entity Relationship Conference, pp. 325-339, 1999.
9. Lenzerini M.: Logical Foundations for Data Integration. SOFSEM 2005. pp 38-40, 2005.
10. Levy A.L., Rajaraman A., Ordille J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. VLDB 1996, pp. 251-262
11. Missikoff M., Taglino F., 2004. An Ontology-based Platform for Semantic Interoperability. Handbook on Ontologies. pp 617-634, 2004.
12. Mohsenzadeh M., Shams F., Teshnehlab M.: Comparison of Schema Matching Systems. WEC (2), pp 141-147, 2005.
13. Nodine M.H., Fowler J., Perry B.: Active Information Gathering in InfoSleuth. CODAS 1999, pp. 15-26.
14. Rahm E ., Bernstein P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), pp 334-350, 2001.
15. Saxon. SAXON: The XSLT and XQuery Processor. http://saxon.sourceforge.net/.
16. Wache H., Vogele T., Visser U., Stuckenschmidt H., Schuster G., Neumann H. and Hubner S.: Ontology-based integration of information - a survey of existing approaches. In Stuckenschmidt, H., ed., IJCAI-01 Workshop: Ontologies and Information Sharing, pp 108-117, 2001.
17. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer, Vol. 25, No 3, pp.38-49, 1992.

Modelling Patterns for Deep Web Wrapper Generation

Thomas Kabisch and Susanne Busse

University of Technology Berlin
{tkabisch and sbusse}@cs.tu-berlin.de

Abstract. Interfaces of web information systems are highly heterogeneous, both in their schemas and their presentation layers. Web interface wrappers need to understand these interfaces in order to enable interoperation among web information systems.
But in a particular application domain hetergeneity is limited: Web interfaces of a domain share a common vocabulary and use a limited set of layout variants. We propose web interface patterns which are characterized by these two aspects. These patterns can be derived from a domain model which is structured into an ontological model and a layout model. A clustering approach allows to automatically generate patterns from a sample set of web interfaces.

1 Introduction

Web data sources offer a huge spectrum of information. Dynamic web data sources are mostly accessible through two interfaces: a form-based query interface and a result interface. Both offer an interface schema which does not need to be similar to the underlying database schema.

In contrast to classical databases, web source interfaces are designed for human usage only. Thus each interface schema is hidden behind a representation layer which tends to be unstructured and changes frequently. In order to make these sources available for higher order applications or to integrate them into federated information systems, a wrapping of these sources needs to be applied.

However, the understanding of heterogeneity of web interfaces is a challenging problem and nearly impossible to generalize independently of the domain. Inside an application domain the heterogeneity is limited.

Fig. 1. Examples for query- and result page fragments of airline web interfaces.

Consider the fragments of airline web interfaces in Figure 1. Intuitively these fragments represent common patterns for web interfaces in this domain. They are typical in two aspects: first in the used layout and second in terms of the vocabulary used to describe elements. The three fields `From, To, Date` in the given query interface fragment are ordered vertically and layouted closely connected. This layout usually is significant for semantically grouped elements. Similarly the presentation of the results is typical for the domain. Moreover the terms of the interfaces will reappear in many other source interfaces of the domain.

This paper proposes an approach to model patterns describing web interfaces of a domain such that the wrapper generation problem can be reduced to a pattern matching problem. A method how to learn these patterns automatically is also presented.

The remaining of the paper is structured as follows. Section 2 introduces domain metamodels to describe ontological and layout aspects of source interfaces in a domain. Section 3 describes the model driven learning process from a sample set of interfaces. Section 4 shows a clustering-based approach to derive patterns. Finally in section 5 and 6 we discuss related work, conclude and give an outlook.

2 Metamodels for Web Interfaces

This section introduces a model-based approach to describe interfaces of web information systems in a given application domain. The model corresponds to the observation that information on a web interface is carried by two paradigms: firstly by the used vocabulary and secondly by a specific layout. Human interactors use both aspects to understand the semantic of an interface. First a user interprets terms in their context to get knowledge about objects of interest, second the arrangement of layout elements helps the user to understand the structure of information. As an example again consider a typical flight booking interface as depicted in Figure 1. There are labels to understand the semantic of adjacent fields (e.g. `Date`). Other pages miss labels, the arrangement of elements is sufficient for human understanding of the semantic of a field (e.g. the query interface for a date field may be separated into three subfields (year, month, day) with no detail labels).

Our metamodel reflects both aspects of web interface design. We define a *Layout Metamodel* which generalizes possible layout variants occurring at web information systems of an application domain and an *Ontological Metamodel* describing ontological structures by terms and semantic relationships between them.

2.1 Ontological Metamodel

The *Ontological Metamodel* allows for building simple ontologies in the domain of interest. These ontologies represent terms occurring on web interfaces in form of labels, names or instances.

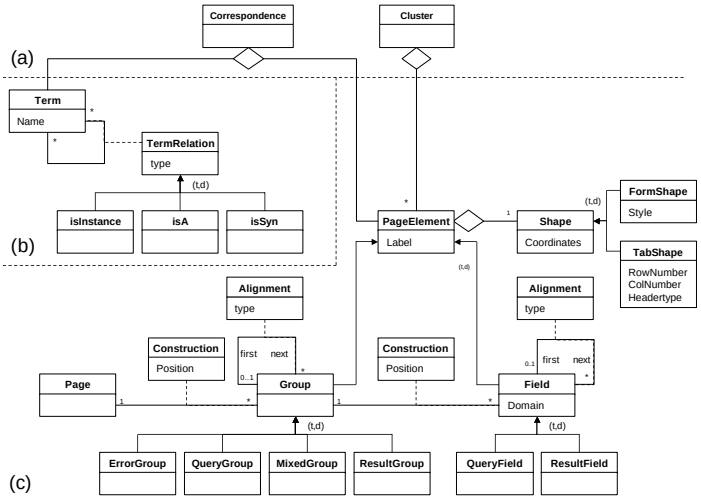

Fig. 2. Web Interface Metamodel: (a) inter-Model relations (b) Ontological Metamodel (c) Layout Metamodel

We only consider relation types that are needed to understand web interfaces: `isA, isSynonymTo, isInstance`. The `isA`-relation describes semantic hierarchies among terms respectively terms and instances (if exact distinction to `isInstance` can not be performed). The `isInstance` is more precise to assign instance terms to concept terms. Instance terms are obtained from field domains (e.g. predefined value lists). Finally the `isSynonym` relationship is used to group semantically similar terms.

2.2 Layout Metamodel

The Layout Metamodel is based on logical groups which strongly correspond to specific structures of web interface presentations. It reflects the hierarchical structure of web interfaces as described by [15] and others for the query interfaces. We generalize the model to all kinds of pages which are relevant for web information systems. We identify three levels of structural layout building blocks in web interfaces: *Page Level*, *Group Level* and *Field Level*, the latter two will be denoted as *PageElement Level*. Elements of a lower level compose elements of the next higher level. This is denoted by the AssociationClasss `Composition`. As each composition is ordered each subelement has a `Position` attribute. In the following we briefly describe the elements of each level.

Page Level A page is the upper most logical unit. We do not distinguish query and result pages because there also exist intermixed forms of pages which share

properties and behavior of query and result pages (e.g. in the flight domain an additional, dynamically created page for further selections may be used to refine the first query step results).

PageElement Level PageElements are building blocks of pages. Each of them is characterized by two main properties: `Shape` and `Label`. A `Shape` has a bounding box. It can be a (`TabShape`) or a (`FormShape`) representing a tabular or a form-style layout, respectively.

A `Label` refers to a text element of a page. So, it can be logical assigned to that `PageElement`. Moreover each `PageElement` is aligned to its neighboring `PageElements` (comp. `Alignment`). Possible alignment types are horizontal or vertical.

Group Level `Groups` represent containers of fields building a higher semantic concept. The table in Figure 1 presents a group that represents the semantic concept of a flight description. It consists of all fields which are grouped together. Groups can be recognized by identifying layout structures because grouped elements are usually visualized closely together.
The distinction between query interface and result representation is accomplished at the group level. Thus groups are specialized into the corresponding group types `QueryGroup` and `ResultGroup`. Moreover an error detection is important for page recognition, thus an additional specialization is `ErrorGroup`. Finally sometimes an exact distinction of query or result is not possible, thus we define `MixedGroups` to carry out this issue.

Field Level `Fields` are the smallest building blocks of pages. Most properties of fields are similar to those of groups. Fields can be aligned to each other, composed to groups and do have a label. We distinguish query interface fields (e.g. HTML input element) and result fields (e.g. table cells). Moreover fields are associated to a bounding box and have an assigned shape. In contrast to groups fields can have information about the domain of possible values.

2.3 Integrated Metamodel

The integration step of both submodels allows a unified view to a domain of deep web sources and is the basis for the later derivation of patterns which are used to generate wrappers for the extraction of information. The integration of both submodels is characterized by two aspects: *Intermodel Correspondences* and *PageElement Clusters*.

Inter-Model Correspondences The Ontological Model and the Layout Model correspond by their used terms. The Layout Model contains terms mainly in the form of labels. All `PageElement`-instances do have a label which is known in the Ontological Model of the domain. Navigating these correspondence relation brings together layout and ontological information. It also allows to address synonymous terms, specialised terms (more precise terms) or instances.

Considering the query interface example in Figure 1 we get the labels `From, To, Date`. The correspondence to the Ontological Metamodel relate these terms to synonymous and related terms, in this case e.g. `Departure Airport`.

PageElement Cluster Semantically similar `PageElements` will be aggregated to a cluster. This idea is inspired from [15] where such clusters are generated for source query interfaces at field level. Consider the examples from Figure 1. Suppose the query interface is an initial centroid for a clustering. We obtain three clusters c_{From}, c_{To} and c_{Date}. Fields at other query interfaces are clustered relatively to these three initial clusters. In a next step groups will be clustered aggregating information of field clusters. As clusters sum up variants of layout and ontological extensions of web interfaces, the derivation of the desired web interface patterns can be performed directly from them.

3 Model Instantiation from Sample Sets of Interfaces

The overall process of pattern learning is structured as follows:

1. Query Interface Analysis
2. Enrichment of the Ontological Model
3. Processing of Result Pages
4. Clustering of PageElements
5. Derivation of Patterns

This section describes the first three points, the next section investigates the clustering and pattern derivation.

3.1 Query Interface Analysis

Starting point of source analysis are query interfaces of a sample set of web interfaces. The analysis consists of some pre-processing steps, particularly a tokenization and a logical parsing, and a model initialization step for the layout and the ontological model.

Tokenization Similarly to [16] we propose a layout driven tokenization of HTML at the beginning. A token contains of a HTML source fragment and is enriched by layout information which is hold in an attached bounding box. The coordinates of bounding boxes are obtained from rendering by a HTML rendering engine (cf. Figure 3).

Logical Parsing by Visual Grammar The tokenized presentation of a page can be transformed into a logical representation which is compatible to the given Layout Metamodel. The transformation is achieved by using a visual grammar as introduced by [16]. Visual grammars do not only recognize strings but also analyse layout structures. This part of parsing is domain independent because grammar Productions analyse layout structures only.

Initialization of Layout Model Having transformed interfaces they can be inserted into layout model. At first corresponding Elements are not grouped together. Thus after initialization the Layout Model is a collection of pages.

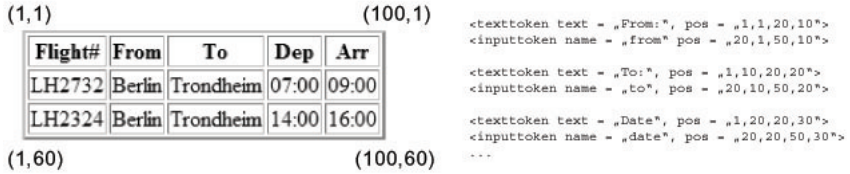

Fig. 3. Simple query interface and result of tokenization

Initialization of Ontological Model The Ontological Domain Model consists of concepts/terms which constitute the domain of interest. Initially terms of interest are labels of PageElements, Names of Elements or Instances (e.g. obtained from predefined Selection lists). They are normalized (stemming, stop-word removal, sorting). A heuristic divides uncommon and common labels (if a label occurs only once in learning set it is not representative and thus omitted in the Ontological Model). Common terms are saved and uncommon are pruned.

3.2 Enrichment of the Ontological Model

Acquiring Semantic Information from the Web

Application of Language Patterns Specific linguistic patterns in the English language guide to semantic relationships among terms. This idea was first stated out by Hearst [9] for the case of hyponymy. According to Hearst the following patterns describe hyponymy among noun phrases (NP):

1. NP such as $\{NP,\ NP,\ldots$ (and | or) $\}\ NP$
2. such NP as $\{NP,\}^*$ {(or|and)} NP
3. NP $\{,NP\}^*$ $\{,\}$ (and|or) other NP
4. NP $\{,\}$ (including|especially) {NP,}*{or|and} NP

Hyponymy can be interpreted as instance-relationship. Thus the above given patterns can be used to extract instances from the Web or to validate instance candidates. The approach is based on [13]. If new instances are searched, a query containing the hyponym and an extraction pattern is issued. In the case of the attribute `airport code` the query `"airport codes such as"` would be issued to look up potential instances. The result snippets will contain sentences which contain instance candidates near the pattern. In this example the first result contains the phrase `airportcodes, such as LAX` where `LAX` can be extracted as potential instance.

The second option to use Google and linguistic patterns is the validation of instance candidates. In this case a whole phrase containing a linguistic pattern is issued (Example query: `"airport codes such as LAX"`) The approach applies Pointwise Mutual Information (PMI) which is a measure for co-occurrences of two terms in information theory. It is formally defined

$$PMI(x,y) = \log_2 \frac{Pr(x,y)}{Pr(x) * Pr(y)} \tag{1}$$

In our context the probabilities will be approximated by the number of results returned by Google.

Wikipedia The online encyclopedia Wikipedia provides a huge corpus of texts defining concept terms. In the case Wikipedia contains an article that describes the concept term, there is a high probability that instances are contained in the article. The structure of Wikipedia can be exploited to find that instances. Many articles contain lists encounting related instances explicitly. Consider again the example for `airport code`. The related Wikipedia article contains a list of all known airports and their codes.
This issue was subject of a prior publication [10]

The Process of Semantic Enrichment In the process of semantic enrichment the technologies summarized above are exploited in different ways. We use Web knowledge to construct the Ontological Model as described in the following subsection.

Mining of Relationships First relationships between terms already in the initial Ontological Model will be mined. Here the language pattern based approach is used in combination with Google and Wikipedia (comp. [2], [10]). Queries containing the above described linguistic patterns are issued against Google in order to validate instance relationships, the PMI measure as described above is used to validate the assumed semantic relations.

Finding additional Synonyms Synonyms are essential to enable powerful patterns. Thus the initial model is enriched by synonyms of initial terms from web. Synonyms are acquired from WordNet.

Finding Instances The most important process is the enrichment of the model with new instances. Here we combine both strategies described above. If Wikipedia articles to the retrieved concept exist, first instances from associated lists are extracted. If no sufficient list of instances can be extracted from Wikipedia, the linguistic pattern approach against Google is facilitated.

3.3 Result Processing

Having enriched the Ontological Model in the way described the analysis of result pages can be executed.

Generating Queries Prior to analysing the result pages need to be obtained. Thus queries against existing query interfaces need to be performed. If instances in the Ontological Metamodel or domain information at query interface level allow an automatic query generation the process is done automatically, otherwise a user is requested to provide useful query parameters.

Page Analysis The result analysis is different to the analysis of query interfaces because the structure of result pages is of a higher degree of heterogeneity. Here a comparison-based approach such as [4] will support the tokenization and visual grammar parsing process.

Having parsed result interfaces the model instance can be updated by layout representation and new terms found while analysing result pages. The model now is ready for the pattern derivation process, which is based on Clustering of semantically close PageElements.

4 Pattern Generation

So far, we have instantiated the layout and ontological model with a sample set but do not have analyzed relationships and similarities between these instances. Now, patterns are identified that show similarities between different web sources (i.e. instances). The analysis consists of two steps: firstly, model instances are clustered to identify similar elements, secondly, patterns are derived from these clusters.

4.1 Clustering

After importing all sources of the learning set, the clustering can be accomplished. The clustering algorithm identifies clusters on field level, on group level and on page level using a bottom-up approach.

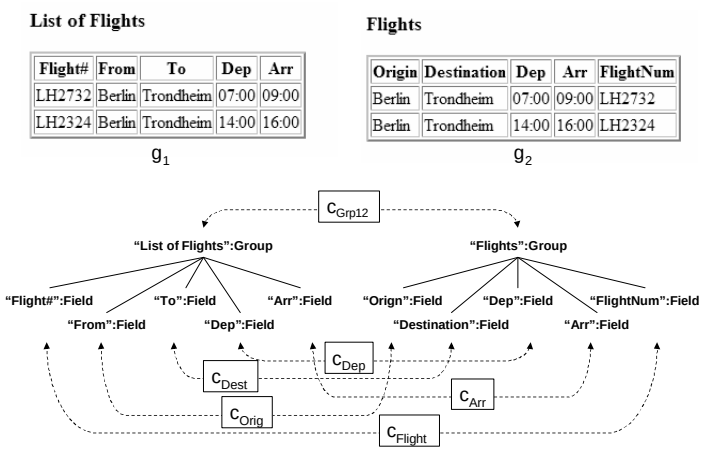

Fig. 4. Clustering between two Interfaces

Figure 4 shows an example of identifying a cluster on group level. Using a bottom-up approach field level clusters are identified at first. In a second step, an aggregation to higher levels (group and page-level) is processed.

Field-Level Clusters Field-level clustering is based on ideas in [15] and [5]: two fields belong to the same cluster if they share similar names, labels or domains. Similarity is defined on synonyms and on linguistic methods.

Group-Level Clusters Two groups belong to the same cluster if they share all field clusters or at least if the set of fields is a real subset of the other.

Page-Level Clustering Two pages belong to the same cluster if they share all group clusters or at least if the set of group clusters of the one page is a subset of the set of the other page.

Group- and Page-Level Clustering Consider the example in Figure 4. Supposing all fields are clustered correctly as shown in the figure we can build up the group level clusters. Initially, the following clusters exist at field level:
$clusters = \{c_{Flight} = \{"Flight\#","FlightNum"\}, c_{Orig} = \{"From","Origin"\}, c_{Dest} = \{"To","Destination"\}, c_{Dep} = \{"Dep"\}, c_{Arr} = \{"Arr"'\}\}$
We choose one page and try to build up higher order clusters based on groups existing on that page. Suppose we start with g_1 = (Field("#Flight"),Field("From"), Field("To"), Field("Dep"), Field("Arr")). In this case our initial field clusters are all that which appear in g_1.

Now, we check other groups whether their fields belong to one of the clusters which build up the initial group and other way round. If there exist no fields that belong to clusters of fields of other groups, a group-level cluster is found. In the given example we would check whether the fields of g_2 are in a cluster of g_1 and whether all clusters in g_1 have corresponding elements in g_2.

The algorithm in Figure 4.1 gives a detailed description on how to identify group-level clusters. It consists of the following steps:

1. Take the groups of one page to initialize group-level clusters.
2. For each group of other pages check if it could be assigned to one of the initial group-level clusters using the condition on the groups' fields.
3. For the remaining groups that could not assigned yet, repeat step one and two to find clusters in the remaining set.

4.2 Patterns Extraction

We can now derive a grammar representing the identified patterns. Again, we construct the grammar bottom-up. Considering our example in Figure 4 we derive the following productions representing semantical patterns. The grammar extends the initial visual grammar toward domain specific tokens.

```
//Field Level Clusters
C_Flight ::= Field("#Flight") | Field("FlightNum")
C_Orig   ::= Field("From") | Field("Origin")
C_Dest   ::= Field("To") | Field("Destination")
C_Dep    ::= Field("Dep")
C_Arr    ::= Field("Arr")

//Group Level Cluster
C_Group12 ::= Group(C_Flight, C_Orig, C_Dest, C_Dep, C_Arr) |
  Group(C_Orig, C_Dest, C_Dep, C_Arr, C_Flight)
```

$groupClusters := \emptyset$
$restGroups := \{g| \in Group\}$
// if there are remaining groups (step 3)
while $restGroups \neq \emptyset$
  // initialise group-level clusters:
  // all (remaining) groups of one selected page build group clusters (step 1)
  choose p_i from $\{p_i|p_i \in Page \wedge \exists g \in restGroups : g$ part of $(p_i)\}$
  $curCentroids := \{cg|cg \in restGroups \wedge cg$ part of $p_i\}$
  for each $cg \in curCentroids$
    $groupClusters := groupClusters \cup \{\{cg\}\}$
  $restGroups := restGroups - curCentroids$
  // for each other group check whether it can be associated to a group cluster (step 2)
  for each $rg \in restGroups$
    for each $cg \in curCentroids$
      // $curGroupCl$ is the set of all groups part of the cluster of cg
      $curGroupCl := \{g \in gCl$ — $gCl \in groupClusters \wedge cg \in gCl\}$
      // it can be associated if no field belongs to another cluster
      if $((\forall f_i \in rg, cl_{fi} \in \{cl_{fi}|cl_{fi}$ part of cluster$(f_i) \wedge cl_{fi}$ part of $p_i\} : cl_{fi}$ part of $cg)$
        and $(\forall f_j \in cg, cl_{fj} \in \{cl_{fj}|cl_{fj}$ part of cluster$(f_j) \wedge cl_{fj}$ part of $p_i\} : cl_{fj}$ part of $rg))$
      then $curGroupCl := curGroupCl \cup \{rg\}$
        $restGroups := restGroups - \{rg\}$
return $groupClusters$

Fig. 5. Algorithm for group level clustering

4.3 Discussion

The derived grammar gives a detailed description of query interfaces and result pages down to the level of tokens. By analyzing a set of web sources of a domain and by clustering, we identify similarities in layout and vocabulary. Thus, the grammar consists of many variants that can be used to generate wrappers for other, unknown web sources.

But this is also a problem: The grammar also covers uncommon variants of patterns. This leads to an undesirable complexity of the grammar causing an exhausting computation time when analyzing a new page. Therefore, we make experiments on reducing the complexity on the basis of the clustering results. There are two possibilities for reduction:

1. a brute-force pruning of uncommon production rules: small-sized clusters indicate uncommon patterns. Thus, these rules are removed.
2. enhancing the grammar by prioritization of production rules: rules representing large-sized clusters are checked at first when analyzing a new page, rules determined from small-sized clusters only if no rule could be found before.

This process is accomplished in a top-down manner: A pruning of higher level productions allows for a pruning on lower levels. As already said, the size of the clusters controls the reduction process. Thereby, a fixed number can be used as the threshold of a small cluster or a relative distinction can be done eliminating rules determined from the k smallest clusters.

5 Related Work

Early work in the field of web wrapper generation investigates the extracting structures [1], [3] and others. The main focus of these works is to recognize regular structures at result pages. Other approaches use domain specific extraction ontologies [6]. Another direction of research concentrates on query interfaces with focus of source integration [7], [15]. Whereas [15] develops a measure how to integrate fields by clustering them to semantically close groups. [5] investigates the special aspect of choosing a right label for fields in integrated query interfaces. [12] brings both interfaces together, reuses information which reappears in both interfaces in order to infer schemes. An interesting aspect is the supposing of an "hidden" visual grammar, which was first published by [16]. The idea to use language patterns to mine semantic relations was first published by [9], [2] first uses these ideas in the web context, finally [13] analyses query interfaces for integration by language patterns. Related to these approaches are ontology generation tools, like the Ontobuilder [11] or the DeepMiner-System [14], which takes query interfaces as basis to generate domain ontologies. This part is related to the semantic enrichment in our approach.

All of related work delivers solutions for aspects of the wrapping of web information systems. They all concentrate to specific detail problems. Some of the solutions are part of bigger systems such as the WISE-Integrator [8], but the general focus of the work is different to our approach. To the best of our knowledge no one presents a model based solution and a clustering based pattern generation approach.

6 Conclusion

Even if interfaces of web sources in general are highly heterogeneous, the interfaces and result pages of deep web sources of a specific domain share both common layouts and vocabulary. This paper has shown how we can use this fact to identify patterns at layout and ontological level to make web source wrapping easier. We presented a model-based approach for the recognition of patterns for web wrapper generation. The main contributions are

- an integrated view to ontological and layout-based aspects of web query interfaces and result pages
- a clustering-based approach to learn semantic web patterns from a given set of web sources

The patterns allow for generation of a grammar useful for the wrapping of unknown sources by pattern matching. Moreover, it becomes easier to handle changes of web interfaces due to design or schema adoptions using pattern transformation. Thus, the re-adjustment of a wrapper is no big effort.

Currently, the pattern grammar is directly discovered from clusters. As the clustering algorithm produces clusters for all fields of the web sources, the grammar can contain many alternative rules, even if some of these rules only represent

a small amount of web sources. We proposed to improve the grammar complexity by pruning strategies, that identifies rules representing such outliers.

The next steps of work will contain more experiments with different pruning strategies for grammar reduction and the identification of patterns on a higher abstraction level than fields or groups, e.g. patterns on a dialog level.

References

1. Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *SIGMOD '03*, pages 337–348. ACM Press, 2003.
2. Philipp Cimiano and Steffen Staab. Learning by googling. *SIGKDD Explorations*, 6(2):24–34, DEC 2004.
3. Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, 2004.
4. Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB '01*, pages 109–118. Morgan Kaufmann, 2001.
5. Eduard C. Dragut, Clement Yu, and Weiyi Meng. Meaningful labeling of integrated query interfaces. In *VLDB'2006*, pages 679–690. VLDB Endowment, 2006.
6. David W. Embley, Douglas M. Campbell, Randy D. Smith, and Stephen W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *CIKM '98*, pages 52–59. ACM Press, 1998.
7. Bin He, Kevin Chen-Chuan Chang, and Jiawei Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *ACM SIG KDD '04*, pages 148–157. ACM Press, 2004.
8. Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Automatic integration of web search interfaces with wise-integrator. *The VLDB Journal*, 13(3):256–273, 2004.
9. Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conf. on Computational linguistics*, pages 539–545, 1992.
10. Thomas Kabisch, Ronald Padur, and Dirk Rother. Using web knowledge to improve the wrapping of web sources. In *ICDE Workshops*, page 4, 2006.
11. Haggai Roitman and Avigdor Gal. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources using sequence semantics. In *Proc. of the EDBT 2006*, pages 573–576, 2006.
12. Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. Instance-based schema matching for web databases by domain-specific query probing. In *Proc. VLDB '04*, 2004.
13. Wensheng Wu, AnHai Doan, and Clement T. Yu. Webiq: Learning from the web to match deep-web query interfaces. In *Proc. ICDE 2006*, page 44, 2006.
14. Wensheng Wu, AnHai Doan, Clement T. Yu, and Weiyi Meng. Bootstrapping domain ontology for semantic web services from source web sites. In *TES 2005, Revised Selected Papers*, pages 11–22, 2005.
15. Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*, pages 95–106. ACM Press, 2004.
16. Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD '04*, pages 107–118. ACM Press, 2004.

Semantic Driven Service Discovery for Interoperability in Web Information Systems *

Devis Bianchini, Valeria De Antonellis, Michele Melchiori and Denise Salvi

Università di Brescia
Dip. Elettronica per l'Automazione
Via Branze, 38
25123 Brescia - Italy
{bianchin|deantone|melchior|salvi}@ing.unibs.it

Abstract. Semantic integration and interoperability in Web Information Systems is a major crucial issue of the recent years. Semantics is particularly important to share and integrate information and services in open P2P environments, where the lack of a common understanding of the world generates the need for explicit guidance in discovering available resources. Nowadays, ontologies supply a common basis for various research areas, wherever semantics is involved. Their use as means to share descriptions of available resources is being investigated for content discovery also in P2P systems and, in particular, ontology-based techniques are at the core of many proposals related to service discovery. In this paper we propose a semantic-driven service discovery approach in a P2P scenario, where peers are organized in semantic communities for integration and interoperability purposes. Specific ontologies are defined to add semantics to service descriptions and to guide service discovery among peers.

1 Introduction

Semantic integration and interoperability in Web Information Systems (WIS) is a major crucial issue of the recent years. Distributed provisioning and invocation of WIS functionalities is addressed through the use of services, whose integration must be obtained by solving semantic heterogeneity of their functional interface. Moreover, each networked enterprise provides different functionalities/services, among which the required one should be semi-automatically detected by solving semantic heterogeneity in their functional interfaces. Semantics is particularly important to share and integrate information and services in open P2P environments, where the lack of a common understanding of the world generates the need for explicit guidance in discovering available resources. Nowadays, ontologies supply a common basis for various research areas, wherever semantics is involved. Their use as means to share descriptions of available resources is

* This work has been partially supported by the ESTEEM (Emergent Semantics and cooperaTion in multi-knowledge EnvironmEnt [7]) PRIN Project funded by the Italian Ministry of Education, University and Research.

being investigated for content discovery also in P2P systems and, in particular, ontology-based techniques are at the core of many proposals related to service discovery. In fact, ontology-based approaches constitute a step towards semantic service discovery, offering the benefits of formal specifications and inferencing capabilities. In open P2P systems, difficulties mainly arise due to the highly dynamic nature of peer interoperability, the lack of any agreed-upon global ontology, as well as the necessity of distributing the computation among peers when processing queries and searching services. Hence, effective service discovery methods and techniques under highly dynamic and context-dependent requirements are primary needs for a unified framework supporting semantic service discovery in a flexible fashion, exploiting service semantic description and flexible ontology-based matchmaking.

In this paper we propose the Semantic Driven Service Discovery approach in an open P2P networked scenario, P2P-SDSD, where peers are organized in a semantic community for integration and interoperability purposes. Ontologies over the Web are introduced to express domain knowledge related to service descriptions and to guide service discovery among peers. For community constitution, we assume that a peer called *promoter* spreads out a *manifesto* containing a suitable portion of its peer ontology, expressing a core domain knowledge for possible member aggregation. Each peer that aims at joining the community matches its own peer ontology against the manifesto and replies to the promoter. Otherwise, if a peer is not interested, it forwards the manifesto to the other peers of P2P network. Once the semantic community is established, services can be searched and exchanged between the members of the community by means of ontology-based techniques.

The paper is organized as follows: Section 2 gives an overview of the proposed architecture for the peer community, underlining the role of semantics; Section 3 explains the startup procedure of the community through the manifesto sharing; Section 4 shows how to interoperate in the community to perform service discovery and publishing; Section 5 compares the proposed approach with related work; finally, in Section 6 some final considerations and future work are discussed.

2 Network architecture

The semantic community is organized like a P2P network and it is constituted by n peers, each of them exporting its own WIS functionalities by means of services. Each peer can play different roles: (i) to search for a given service (*requester*); (ii) to propose a set of suitable services when a service request is given, through the application of matchmaking techniques (*broker*); (iii) to provide a selected service for its invocation and to publish a new service (*provider*). In an evolving collaborative P2P community, a peer can contain the description of a required service, while a different peer acts as a provider for that service, or a peer can be both a requester and a broker. According to this general view, each peer presents the architecture shown in Figure 1. In the following we focus on the Semantic Peer Registry and the Service MatchMaker, without details on the Application Program Interface, the Service Invoker and the P2P module handling inter-peer communications.

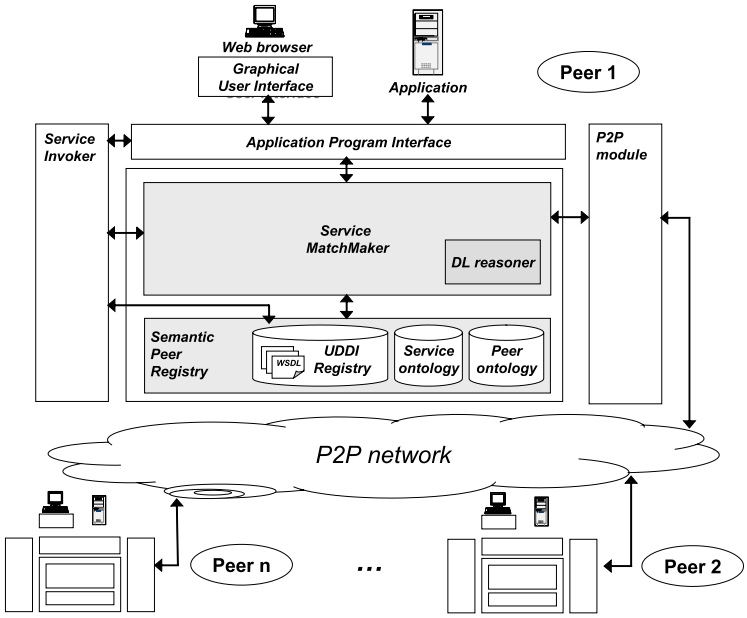

Fig. 1. Peer architecture.

2.1 Semantic Peer Registry

Service descriptions represent functional aspects of a service, based on the WSDL standard for service representation, in terms of service category, service functionalities (operations) and their corresponding input/output messages (parameters). Services are stored in an extended UDDI Registry, called *Semantic Peer Registry*, where besides the UDDI registry and WSDL descriptions, a peer ontology provides semantic knowledge related to service descriptions and a service ontology contains semantic service descriptions with reference to the peer ontology.

The peer ontology is constituted by:

- a *Service Functionality Ontology (SFO)*, that provides knowledge on the concepts used to express service functionalities (operations);
- a *Service Message Ontology (SMO)*, that provides knowledge on the concepts used to express input and output messages (parameters) of services.

Concepts in the peer ontology are organized according to `subclass-of` and `equivalent-to` semantic relationships. Furthermore, the peer ontology is extended by a *thesaurus* providing terms and terminological relationships (as synonymy, hypernymy and so on) with reference to names of concepts in the ontology. In this way, it is possible to extend matchmaking capabilities when looking

for correspondences between elements in service descriptions and concepts in the ontology.

In the service ontology, services are semantically represented by DL logic expressions [4], whose elements (service category, operation names, input/output parameter names) are properly mapped to the peer ontology. Both peer ontology and service ontology are expressed in OWL-DL.

2.2 Service MatchMaker

The Service MatchMaker is in charge of comparing service descriptions, combining together different matchmaking models [5]: (i) a deductive model, exploiting deduction algorithms for reasoning on service descriptions, (ii) a similarity-based model, where retrieval metrics are applied to measure the degree of match between services. In this paper we will combine these matchmaking strategies in the context of P2P semantic communities to improve service discovery.

Both the matchmaking models are based on jointed use of the peer ontology and the thesaurus to classify the type of match between services (exact, partial and mismatch) and to quantify it by means of suitable similarity coefficients, that is, by evaluating the service similarity. Terminological relationships considered in the thesaurus are SYN for synonymy, BT (resp., NT) for broader term (resp., narrower term) and RT for related term. The thesaurus is exploited to compute the *Name Affinity* coefficient between names of input/output parameters and operations.

Definition 1 (Name Affinity coefficient). *Given the thesaurus $\mathcal{TH}$, the Name Affinity coefficient between two terms $t, t' \in \mathcal{TH}$, denoted by $NA(t,t')$, is: (i) 1.0 if $t = t'$; (ii) $max_l(\tau(t \rightarrow^l t'))$ if $t \neq t' \wedge t \rightarrow^l t', l \geq 1$, where $t \rightarrow^l t'$ denotes a path of terminological relationships from t to t'; (iii) 0.0 otherwise. A weight $\sigma_{tr} \in [0,1]$ is associated to each kind of terminological relationship tr, in order to evaluate its implication for name affinity; in our experimentation, $\sigma_{SYN} = 1$, $\sigma_{BT/NT} = 0.8$ and $\sigma_{RT} = 0.5$. The function $\tau(t \rightarrow^l t') = \prod_{k=1}^{l}(\sigma_{tr_k}) \in [0,1]$ defines the strength of $t \rightarrow^l t'$ as the product of the weights of all terminological relationships in the path. Since between two terms in the thesaurus there can exist more than one path, the one with the highest strength is chosen.*

We say that t and t' have *name affinity* ($t \sim t'$) if and only if $NA(t,t') \geq \alpha$, where $\alpha > 0$ is a threshold given by experimental results to select only terms with high values of the *Name Affinity* coefficient. The choice of the actual value of α is done during a training phase where α is set initially to a given value (i.e., 0.5) then this value is increased or decreased until a satisfactory trade-off between recall and precision is obtained. That is, increasing α leads to be more selective by identifying a name affinity between two terms only if they are very similar according to the thesaurus. Viceversa, by decreasing α, name affinities are established also between pairs of terms that are related by a weaker path of terminological relationships. Name Affinity coefficient is used to extend

traditional Description Logic subsumption test (denoted by $\sqsubseteq$) between two generic terms, even if they do not belong to the peer ontology.

Definition 2 (Affinity-based subsumption test). *Given an atomic concept C in the peer ontology $\mathcal{PO}$, we define the set of terms in the thesaurus that have name affinity with the concept C as $C_{\mathcal{TH}} = \{T \in \mathcal{TH} \mid T \sim C\}$. Analogously, we define the set of concepts of $\mathcal{PO}$ that have name affinity with a term T in $\mathcal{TH}$ as $T_{\mathcal{PO}} = \{C \in \mathcal{PO} \mid T \in C_{\mathcal{TH}}\}$.*
Given the peer ontology $\mathcal{PO}$, the thesaurus $\mathcal{TH}$ and a pair of terms T^1 and T^2 used in service descriptions to denote service elements, T^1 is subsumed by T^2 with respect to $\mathcal{TH}$, denoted by $T^1 \sqsubseteq_{\mathcal{TH}} T^2$, if and only if there exists $C \in {T^1}_{\mathcal{PO}}$ and $D \in {T^2}_{\mathcal{PO}}$ such that $C \sqsubseteq D$ is satisfied in $\mathcal{PO}$.
Note that we pose $T^1 \equiv_{\mathcal{TH}} T^2$ if both $T^1 \sqsubseteq_{\mathcal{TH}} T^2$ and $T^2 \sqsubseteq_{\mathcal{TH}} T^1$ hold.

The deductive matchmaking model applies the affinity-based subsumption test to service description elements considered separately (categories, operations, I/O parameters) to classify the match between a service request $\mathcal{R}$ and each supplied service $\mathcal{S}$. In [5] a formal definition of the following kinds of matches is given:

Exact match, to denote that $\mathcal{S}$ and $\mathcal{R}$ have the same capabilities, that is, for each operation in $\mathcal{R}$ there exists an operation in $\mathcal{S}$ that has: (i) equivalent name; (ii) equivalent output parameters; (iii) equivalent input parameters.

Plug-in match, to denote that $\mathcal{S}$ offers at least the same capabilities of $\mathcal{R}$, that is, for each operation in $\mathcal{R}$ there exists an operation in $\mathcal{S}$ that has: (i) an equivalent or more specific operation name; (ii) an equivalent or more specific output parameter for each output parameter of the required operation; (iii) a set of input parameters, each of them is equivalent or more generic than an input parameter of the required operation; the inverse kind of match is denoted as `subsume`; the rationale behind the `plug-in` and `exact` matches is that $\mathcal{S}$ totally fulfills the request $\mathcal{R}$ if it provides all the required outputs, but, on the other hand, $\mathcal{R}$ must be able to provide all the inputs needed for the execution of $\mathcal{S}$.

Intersection match, to denote that $\mathcal{S}$ and $\mathcal{R}$ have some common capabilities, that is, there exist an operation in $\mathcal{S}$ and an operation in $\mathcal{R}$ such that: (i) their names are related in any generalization hierarchy; (ii) there exists a pair of I/O parameters, one from $\mathcal{R}$ and one from $\mathcal{S}$, that are related in any generalization hierarchy.

Mismatch, otherwise.

Service categories are initially exploited to filter out not suitable services: only supplied services whose categories are equivalent or more specific than the request category are selected. We consider a qualitative ranking among the kinds of matches, that is, `exact` > `plug-in` > `subsume` > `intersection` > `mismatch`.

Similarity analysis is applied to quantify the match between services and it is based on the Name Affinity coefficient. In particular, when `exact/plug-in` match occurs, similarity between services is set to 1 (full similarity); if `mismatch` occurs, the similarity value is set to zero; finally, when `subsume` and `intersection`

Entity-based similarity
$$ESim(\mathcal{R},\mathcal{S}) = \frac{2\cdot A_{tot}(IN_{\mathcal{R}},IN_{\mathcal{S}})}{\lvert IN_{\mathcal{R}}\rvert+\lvert IN_{\mathcal{S}}\rvert} + \frac{2\cdot A_{tot}(OUT_{\mathcal{R}},OUT_{\mathcal{S}})}{\lvert OUT_{\mathcal{R}}\rvert+\lvert OUT_{\mathcal{S}}\rvert} \in [0,2]$$ $IN_{\mathcal{R}}, IN_{\mathcal{S}}$ - sets of input parameter names of $\mathcal{R}$ and $\mathcal{S}$ $OUT_{\mathcal{R}}, OUT_{\mathcal{S}}$ - sets of output parameter names of $\mathcal{R}$ and $\mathcal{S}$ $A_{tot}(IN_{\mathcal{R}}, IN_{\mathcal{S}}) = \sum_{in^i_{\mathcal{R}}\in IN_{\mathcal{R}}, in^j_{\mathcal{S}}\in IN_{\mathcal{S}}} NA(in^i_{\mathcal{R}}, in^j_{\mathcal{S}})$ $A_{tot}(OUT_{\mathcal{R}}, OUT_{\mathcal{S}}) = \sum_{out^i_{\mathcal{R}}\in OUT_{\mathcal{R}}, out^j_{\mathcal{S}}\in OUT_{\mathcal{S}}} NA(out^i_{\mathcal{R}}, out^j_{\mathcal{S}})$
Operation similarity
$$OpSim(op^i_{\mathcal{R}}, op^j_{\mathcal{S}}) = NA(name\_op^i_{\mathcal{R}}, name\_op^j_{\mathcal{S}}) + \frac{2\cdot A_{tot}(IN^i_{\mathcal{R}},IN^j_{\mathcal{S}})}{\lvert IN^i_{\mathcal{R}}\rvert+\lvert IN^j_{\mathcal{S}}\rvert} + \frac{2\cdot A_{tot}(OUT^i_{\mathcal{R}},OUT^j_{\mathcal{S}})}{\lvert OUT^i_{\mathcal{R}}\rvert+\lvert OUT^j_{\mathcal{S}}\rvert} \in [0,3]$$ $IN^i_{\mathcal{R}}, IN^j_{\mathcal{S}}$ - sets of input parameter names of the i-th operation of $\mathcal{R}$ and the j-th operation of $\mathcal{S}$ $OUT^i_{\mathcal{R}}, OUT^j_{\mathcal{S}}$ - sets of output parameter names of the i-th operation of $\mathcal{R}$ and the j-th operation of $\mathcal{S}$
Functionality-based similarity
$$FSim(\mathcal{R},\mathcal{S}) = \frac{2\cdot\sum_{i,j} OpSim(op^i_{\mathcal{R}}, op^j_{\mathcal{S}})}{\lvert OP(\mathcal{R})\rvert+\lvert OP(\mathcal{S})\rvert} \in [0,3]$$ $OP(\mathcal{R}), OP(\mathcal{S})$ - sets of operation names of $\mathcal{R}$ and $\mathcal{S}$
Global similarity
$$GSim(\mathcal{R},\mathcal{S}) = w_1 \cdot NormESim(\mathcal{R},\mathcal{S}) + w_2 \cdot NormFSim(\mathcal{R},\mathcal{S}) \in [0,1]$$ w_1, w_2 - weights introduced to assess the relevance of each kind of similarity ($w_1 \in [0,1]$ and $w_2 = 1-w_1$) $NormESim(), NormFSim()$ - $ESim()$ and $FSim()$ normalized to the range $[0,1]$

Table 1. Similarity coefficients between service descriptions $\mathcal{R}$ (request) and $\mathcal{S}$ (supply).

match occur, similarity coefficients exposed in Table 1 are computed to evaluate similarity between $\mathcal{R}$ and $\mathcal{S}$.

Entity-based similarity coefficient $ESim$ evaluates the similarity of all the I/O parameters of the considered services to measure how much they are based on the same information; *Functionality-based similarity* coefficient $FSim$ compares pairs of operations together with their corresponding I/O parameters to measure how much the two services perform the same functionalities. Each component in $ESim()$ formula in Table 1 produces by construction a value belonging to the [0,1] range, so that $ESim() \in [0,2]$. Similarly, each component in $OpSim()$ formula produces by construction a value belonging to the [0,1] range, so that $OpSim() \in [0,3]$ and, accordingly, $FSim() \in [0,3]$. $ESim$ and $FSim$ are normalized into the [0,1] range and combined in the *Global similarity* coefficient $GSim$. A detailed description of similarity coefficients and their application is given in [6].

If the value of $GSim$ is equal or greater than a threshold δ, then $\mathcal{R}$ and $\mathcal{S}$ are considered *similar*. It is noteworthy to say that these coefficients are specifically oriented toward a comparison between service descriptions expressed in terms of operations and corresponding I/O parameters rather than pure vectors of terms. The result of this similarity evaluation depends on the choice of δ. Actual value of δ is experimentally set during a training phase. Since actual values of $ESim$ and $FSim$ depend on name affinity evaluation and therefore depend also on the α value, we first set this value to obtain a satisfactory name affinity evaluation then we vary the value of δ until an acceptable trade-off between precision and recall is obtained on a set of training services.

The semantic matchmaking techniques can also be applied to compare services stored on the same peer or on different peers, to identify semantic links. In particular, a semantic link between two services is established if the kind of match is not `mismatch` and the $GSim$ value is equal or greater than threshold δ. The semantic link is described by the kind of match and the $GSim$ coefficient and it is stored in the service ontology.

Two types of semantic links can be established: (i) semantic links between services belonging to the same peer (*intra-peer semantic links*); (ii) semantic links between services belonging to different peers (*inter-peer semantic links*); the related peers are referred as *semantic neighbors*. In the following, we briefly illustrate the process of semantic community constitution as defined in the ESTEEM Project [7], then focusing on inter-peer semantic link definition.

3 P2P semantic community setup

Figure 2 shows the process of semantic community constitution, where the information owned by the peers at each step is listed on the right. We denote as *Global Overlay (GO)* the generic P2P network which peers potentially aiming at joining semantic community belong to. In the P2P network, each peer knows its own IP address. The promoter of the semantic community builds a *startup manifesto*, containing a portion of its peer ontology manually selected, apt to express the core interests of the intended semantic community. The startup manifesto generally will contain the upper level concepts of promoter's Service Message Ontology, since it contains knowledge about the domain in which the sharable services operate. It is important that startup manifesto is limited, so it can be easily flooded over the P2P network starting the setup process. When a peer in *GO* receives the startup manifesto, it can accept or not to join the community. In case of acceptance, the peer sends back a message to the promoter with its IP address and becomes a candidate member. Otherwise, the peer simply ignores the startup manifesto and forwards it to the other peers in *GO*, except the one from which the startup manifesto came.

Candidate members constitute a so-called *Semantic Overlay (SO)*, but the community is not established yet. They know their own IP address and the startup manifesto of the community. The promoter sends to all candidate members the complete version of *manifesto* together with the list of candidate members IP addresses. The manifesto contains both the Service Message Ontology

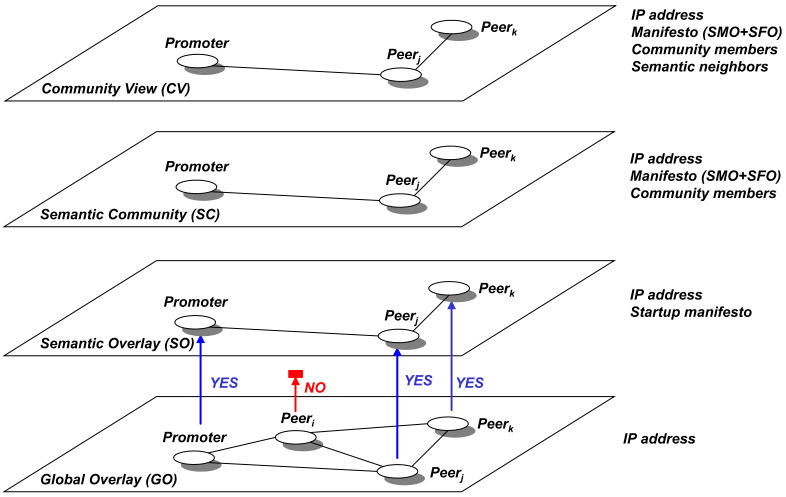

Fig. 2. Setup of the semantic community.

and the Service Functionality Ontology of the promoter and is sent only to candidate members to reduce network overload. At this point, the *Semantic Community (SC)* is established and candidate members become community members. Each peer knows its own IP address, the manifesto and the community member current list.

Once the semantic community is established, each peer searches for its semantic neighbors to establish the inter-peer semantic links. To do this, it sends a probe service request for each service it wants to make sharable; this probe service request contains the description of the service functional interface (categories, operations, I/O parameters). The probe service request is sent to all the other peers of the semantic community, according to the community member list. Each peer receiving the probe service request matches it against its own service descriptions by applying the matchmaking techniques explained in Section 2.2 and obtains for each comparison the kind of match mt and the similarity degree $GSim$. If mt is not `mismatch` and $GSim$ is equal or greater than threshold δ, they are enveloped in a message sent back to the peer from which the probe service request came. An *inter-peer semantic link* is established between the two peers, that become semantic neighbors with respect to the linked services.

In this way, each peer can build a map of its semantic neighbors, with similar services and semantic links with them. In this phase we say that peers belong

to the *Community View (CV)* and know their own IP address, the community manifesto, the community member current list and semantic neighbors.

Community can evolve when new peers join it or new services are published on community members. To collect the list of new peers that aim at joining the community, promoter sends again the startup manifesto on P2P network and receives the IP addresses of new candidate members, to which the promoter sends the community manifesto, while it sends an updated member list to all the peers in the community. A peer that receives a new member list can repeat the procedure to find semantic neighbors provided that the community changed. When a community member publishes a new service, it advertises the promoter, that triggers the semantic community; in this way, all members update their inter-peer semantic links by means of probe service request mechanism. Note that the semantic community and community view are established before service requests are propagated. As explained in the next section, when a peer receives a request, it applies matchmaking strategies to properly select a subset of its semantic neighbors that are more suitable to serve that particular request.

4 P2P semantic community interoperation

Once the semantic community is established and inter-peer semantic links are defined, services can be searched and exchanged between community members. A peer p of the community can receive a service request either directly from the Application Program Interface or from another community member. Given a service request $\mathcal{R}$, a peer p searches for suitable services in its own Semantic Peer Registry and retrieves a list $CS = \{\langle \mathcal{S}_1, GSim_1, mt_1 \rangle, \ldots \langle \mathcal{S}_n, GSim_n, mt_n \rangle\}$ of services with corresponding similarity values $GSim_i \geq \delta$ and match type mt_i different from `mismatch`.

If a service $\mathcal{S}_i \in CS$ presents an `exact` or a `plug-in` match with the request, then $\mathcal{S}_i$ satisfies completely the required functionalities and it is not necessary to forward the service request to semantic neighbors with respect to $\mathcal{S}_i$. Otherwise, if $\mathcal{S}_i$ presents a `subsume` or an `intersection` match with the request, the peer p forwards the request to those peers that are semantic neighbors with respect to $\mathcal{S}_i$. Peer p does not consider semantic neighbors that present a `subsume` or an `exact` match with $\mathcal{S}_i$, because this means that they provide services with the same functionalities or a subset of $\mathcal{S}_i$ functionalities and they cannot add further capabilities to those already provided by $\mathcal{S}_i$ on the peer p. This phase is repeated for every $\mathcal{S}_i \in CS$. Semantic neighbors which present inter-peer links with any service $\mathcal{S}_j$ stored on p, but not included in CS, are discarded since they are not relevant with respect to $\mathcal{R}$. Each selected semantic neighbor sn presents a set of k inter-peer semantic links with some services on p that are suitable for $\mathcal{R}$. It is described as $\langle sn, \{\langle \mathcal{S}_1, GSim_1, mt_1 \rangle, \ldots \langle \mathcal{S}_k, GSim_k, mt_k \rangle\}\rangle$, where $\mathcal{S}_1 \ldots \mathcal{S}_k \in CS$ and have a semantic link with some services stored on sn, featured by $GSim_1 \ldots GSim_k$ similarity degree and $mt_1 \ldots mt_k$ type of match, respectively. Note that this formulation holds even if sn has more than one service related to the same service $\mathcal{S}_i \in CS$.

Since the relevance of sn does not depend only on the similarity associated to the semantic links between p and sn, but also on the similarity degree between $\mathcal{S}_i \in CS$ and $\mathcal{R}$, the harmonic mean is used to combine these two aspects. Therefore, the relevance of a semantic neighbor sn is defined as:

$$r_{sn} = \frac{1}{k} \sum_{i=1}^{m_{sn}} \frac{2 * GSim_i * GSim(\mathcal{R}, \mathcal{S}_i)}{GSim_i + GSim(\mathcal{R}, \mathcal{S}_i)} \tag{1}$$

Relevance values are used to rank the set of semantic neighbors in order to filter out not relevant semantic neighbors (according to a threshold-based mechanism) and to further constrain the request forwarding (according to a token-based strategy).

Example. Let consider $\texttt{Peer}_A$ providing three services `findDisease`, `findDiagnosis` and `findLaboratory` for which the following semantic neighbors have been found (Figure 3):

$\langle \texttt{Peer}_B, \{\langle \texttt{findDisease}, 0.753, \texttt{intersection} \rangle, \langle \texttt{findDiagnosis}, 1.0, \texttt{exact} \rangle\}\rangle$

$\langle \texttt{Peer}_C, \{\langle \texttt{findLaboratory}, 0.7, \texttt{intersection} \rangle, \langle \texttt{findDisease}, 1.0, \texttt{plug-in} \rangle\}\rangle$

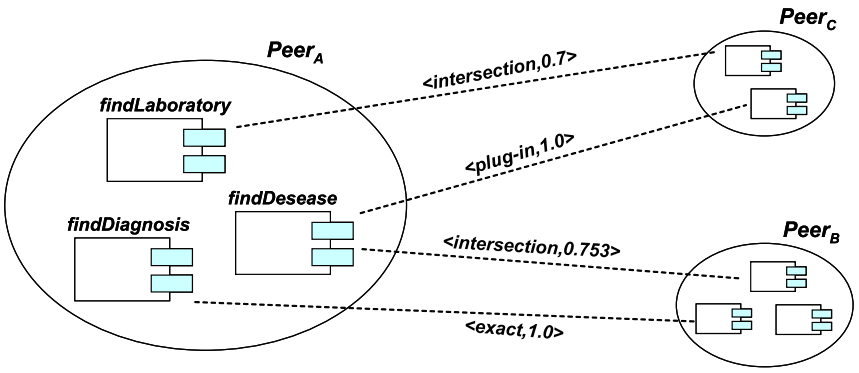

Fig. 3. An example of inter-peer semantic links in the P2P community.

Let suppose that for a given request $\mathcal{R}$ on the $\texttt{Peer}_A$, we obtain the following list of matching services: $\texttt{CS} = \{\langle \texttt{findDisease,0.9,intersection} \rangle, \langle \texttt{findDiagnosis, 0.7,subsume} \rangle\}$, while $\{\langle \texttt{findLaboratory,0.0,mismatch} \rangle\}$ is excluded from `CS`. For what concerns `findDisease`, both $\texttt{Peer}_B$ and $\texttt{Peer}_C$ must be considered as semantic neighbors, since they could provide some additional capabilities with respect to $\texttt{Peer}_A$. Moreover, for what concerns `findDiagnosis`, $\texttt{Peer}_C$ is not a semantic neighbor, while $\texttt{Peer}_B$ has a related service, that presents an `exact` match with `findDiagnosis`. This means that $\texttt{Peer}_B$ has no additional capabilities to offer with respect to those already provided by `findDiagnosis` in $\texttt{Peer}_A$. The resulting set of selected semantic neighbors with respect to $\mathcal{R}$ is $\{\langle\ \texttt{Peer}_C, \{\texttt{findDisease, 1.0, plug-in}\}\ \rangle, \langle\ \texttt{Peer}_B, \{\texttt{findDisease, 0.753, intersection}\}\ \rangle\}$. The relevance values for $\texttt{Peer}_C$ and $\texttt{Peer}_B$ with respect to the request $\mathcal{R}$ are then:

$$r_{\texttt{Peer}_\texttt{C}} = \frac{2 * 1.0 * 0.9}{1.0 + 0.9} = 0.947; \quad r_{\texttt{Peer}_\texttt{B}} = \frac{2 * 0.753 * 0.9}{0.753 + 0.9} = 0.82 \qquad (2)$$

5 Related work

Service semantic description in a P2P context and semantic links between peers based on provided services is a crucial aspect to improve effectiveness and performance of P2P service discovery. Several proposals have been made in literature to enable semantic-enhanced service discovery in a P2P context, not necessarily based on semantic communities. In METEOR-S [10] service descriptions are kept in UDDI Registries semantically enhanced with local *domain specific ontologies*, while a centralized *registries ontology* is used to classify peer registries. During the discovery process, *registries ontology* is browsed to find the proper registry to which submit the request. GloServ [1] defines a predefined *skeleton ontology*, represented as a taxonomy of concepts each of them representing a service category, that in turn is associated to a high level server. Each server represents a P2P network of nodes organized via a *Content Addressable Network (CAN)* [9]. These peers provide services belonging to the category represented by ontological concept associated to their own server. The *skeleton ontology* is a centralized structure, replicated and cached in each high level server. Service discovery is organized in two phases: (a) browsing concept taxonomy to find the proper high level server; (b) keyword-based search through a CAN lookup table. Our approach does not constrain peers to use a common ontology: each peer exploits its own peer ontology, eventually mapped to the community manifesto to enhance peer interoperability.

DAML-S for P2P [8] considers a P2P network, where each member stores a local DAML-S ontology to describe services; service semantic descriptions are kept in local UDDI registries extended with DAML-S ontologies. However, no semantic links are found between peers that provide similar services and when a peer does not satisfy a request, a flooding mechanism is used to find suitable services on the other peers of the network. ARTEMIS [2] defines a P2P network, where each peer has a coarse-grained *Service Functionality Ontology (SFO)* to classify services and a fine-grained *Service Message Ontology (SMO*) to annotate services with medical concepts, based on medical information standards. Peer store in a reference mediator super-peer the services they provide. A peer sends a request to its reference mediator expressed in terms of its own ontologies; mediator uses ontology mappings to find matching services in its local registries and also forwards the request to other mediators. No semantic links are exploited to prune the set of peers to which forward the request.

WSPDS [3] describes a P2P network where peers have local DAML-S ontologies to provide service semantics and semantic links with other peers based on similarity between services they provide. When a request is submitted to a peer, it searches for local matching results and forwards the request to all the semantic neighbors, independently of the current request or the local results of the query. Original contribution of our approach is related to the flexibility of the

matchmaking process based on both deductive and similarity-based approaches and on the definition and exploitation of inter-peer semantic links.

6 Concluding remarks

In this paper, we proposed a semantic-based approach for service discovery in a P2P scenario, where peers are organized in communities. Specific ontologies (called *peer ontologies*) are used to add semantics to service descriptions and are exploited during community setup to find inter-peer semantic links. These links are used to propagate a service request between the peers of the community in an efficient way. Implementation and experimentation of the proposed approach are being performed in the application scenario of scientific collaboration in medicine [7].

References

1. K. Arabshian and H. Schulzrinne. An Ontology-based Hierarchical Peer-to-Peer Global Service Discovery System. *Journal of Ubiquitous Computing and Intelligence (JUCI)*, 2006.
2. The ARTEMIS Project: A Semantic Web Service-based P2P Infrastructure for the Interoperability of Medical Information Systems. `http://www.srdc.metu.edu.tr/webpage/projects/artemis/`.
3. F. Banaei-Kashani, C. Chen, and C. Shahabi. WSPDS: Web Services Peer-to-Peer Discovery Service. In *Proc. of the Int. Conference on Internet Computing (IC'04)*, pages 733–743, Las Vegas, Nevada, USA, 2004.
4. D. Bianchini and V. De Antonellis. Ontology-based Semantic Interoperability Tools for Service Dynamic Discovery. In *IFIP Int. Conf. on Interoperability of Enterprise Software Applications, INTEROP-ESA'05*, Geneva, Switzerland, 2005.
5. D. Bianchini, V. De Antonellis, M. Melchiori, and D. Salvi. Semantic-enriched Service Discovery. In *IEEE ICDE Int. Workshop on Challenges in Web Information Retrieval and Integration, WIRI 2006*, Atlanta, Georgia, USA, 2006.
6. D. Bianchini, V. De Antonellis, B. Pernici, and P. Plebani. Ontology-based Methodology for e-Service discovery. *Journal of Information Systems, Special Issue on Semantic Web and Web Services*, 31(4-5):361–380, June-July 2006.
7. ESTEEM Project Home Page: Emergent Semantics and cooperaTion in multiknowledge EnvironmEnt. `http://www.dis.uniroma1.it/~esteem/index.html`.
8. M. Paolucci, K.P. Sycara, T. Nishimura, and N. Srinivasan. Using daml-s for p2p discovery. In *Proceedings of the Int. Conference on Web Services (ICWS2003)*, 2003.
9. S. Ratnasamy, P. Francis, M. Handley, R.M. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of the ACM SIGCOMM'01 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 161–172, San Diego, CA, USA, August 2001.
10. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, 6(1):17–39, 2005.

Web Service Mediation Through Multi-level Views

Manivasakan Sabesan and Tore Risch

Department of Information Technology, Uppsala University, Sweden
{msabesan, Tore.Risch}@it.uu.se

Abstract. The web Service MEDiator system (WSMED) provides general query capabilities over data accessible through web services by reading WSDL meta-data descriptions. Based on imported meta-data, the user can define views that extract data from the results of calls to web service operations. The views can be queried using SQL. The views are specified in terms of declarative queries that access different web service operations in different ways depending on what view attributes are known in a query. To enable efficient query execution over the views by automatic query transformations the user can provide semantic enrichments of the meta-data with key constraints. We evaluated the effectiveness of our approach over multi-level views of existing web services and show that the key constraint enrichments substantially improve query performance.

Keywords: web service views, query optimization, semantic enrichment

1. Introduction

Web services [4] provide an infrastructure for web applications by defining sets of operations that can be invoked over the web. Web service operations are described by meta-data descriptions of operation signatures, using the *Web Services Description Language* (WSDL) [5]. An important class of operations is to access data through web services, e.g. Google's web page search service [12] and the United States Department of Agriculture nutrition database of foods [27]. However, web services don't support general query or view capabilities; they define only operation signatures.

We have developed a system, WSMED – Web Service MEDiator, to facilitate efficient queries over web services. The view definitions called *WSMED views* are defined in terms of imported WSDL descriptions of web service operations. Furthermore, *multi-level* WSMED views can be defined in terms of other WSMED views. Web services return nested XML structures (i.e. records and collections), which have to be flattened into relational views before they can be queried with SQL. The knowledge how to extract and flatten relevant data from a web service call is defined by the user as queries called *capability definitions* using and object-oriented query language, *WSMED query language* (WQL), which has support for web service data types.

An important semantic enrichment is to allow for the user to associate with a given WSMED view different capability definitions depending on what view attributes are known in a query, the *binding pattern* of the capability definition. The WSMED query

optimizer automatically selects the optimal capability definition for a given query by analyzing its used binding patterns. These view definitions enrich the basic web service operations to support SQL data access queries.

A WSDL operation signature description does not provide any information about which parts of the signature is a key to the data accessed through the operation. As we show, this information is critical for efficient query execution of multi-level WSMED views. Therefore, we allow the user to declare to the system all (compound) keys of a given WSMED view, called *key constraints*.

This paper is organized as follows: Section two describes the architecture of WSMED. Section three gives examples of WSMED view definitions using an existing web service and explains the capability definitions. Section four analyzes the performance of a sample query to verify the effectiveness of query transformations based on the semantic enrichments compared to conventional relational algebra transformations. Section five describes the strategies of the query processor. Section six discusses related work. Finally section seven summarizes the results and indicates future work.

2. The WSMED System

Figure 1a, illustrates WSMED's system components. Imported WSDL meta-data is stored in the *web service meta-database* using a generic *web service schema* that can represent any WSDL definition. The *WSDL Importer* populates the web service meta-database, given the URL of a WSDL document. It reads the WSDL document using the WSDL parser toolkits *WSDL4J* [24] and *Castor* [23]. The retrieved WSDL document is parsed and automatically converted into the format used by the web service meta-database. In addition to the general web service meta-database, WSMED also keeps additional user-provided *WSMED enrichments* in its local store.

The *query processor* exploits the web service descriptions and WSMED enrichments to process queries. The query processor calls the *web service manager* which invokes web service calls using *Simple Object Access Protocol* (SOAP) [13] through the toolkit *SAAJ* [19] to retrieve the result for the user query.

Figure 1b illustrates architectural details of the query processor. The *calculus generator* produces from an SQL query an internal calculus expression in a Datalog dialect [18]. This expression is passed to the *query rewriter* for further processing to produce an equivalent but simpler and more efficient calculus expression.

The query rewriter calls the *view processor* to translate SQL query fragments over the WSMED view into relevant capability definitions that call web service operations. An important task for the query rewriter is to identify overlaps between different subqueries and views calling the same web service operation. This requires knowledge about the key constraints. We will show that such rewrites significantly improve the performance of queries to multi-level views of web services.

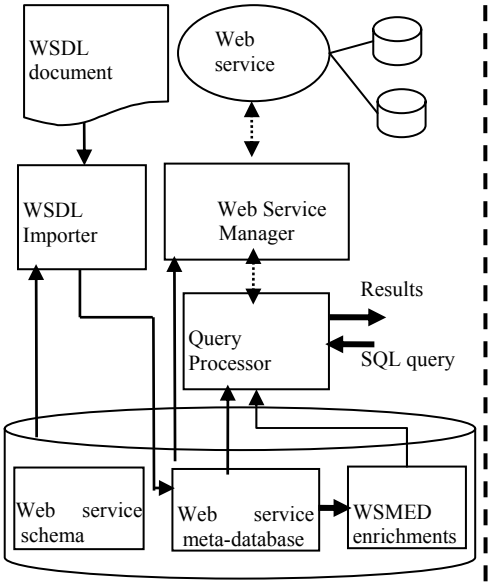

Figure 1a: WSMED components

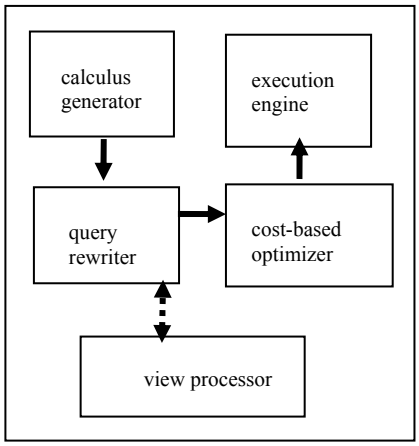

Figure 1b: Query Processor

The rewritten query is finally translated into an algebra expression by a *cost-based optimizer* that uses a generic web service cost model as default. The algebra has operators to invoke web services and to apply external functions implemented in WSDL (e.g. for extraction of data from web service results). The algebra expression is finally interpreted by the *execution engine*. It uses the web service meta-database to generate a SOAP message when a web service operation is called.

3. WSMED Views

To illustrate and evaluate our approach we use a publicly available web service to access and search the National Nutrient Database for US Department of Agriculture [28]. The database contains information about the nutrient content of over 6000 food items. It contains five different operations: *SearchFoodByDescriptions*, *CalculateNutrientValues*, *GetAllFoodGroupCodes*, *GetWeightMethods* and *GetRemainingHits*. We illustrate WSMED by the operation *SeachFoodByDescriptions* to search foods given a *FoodKeywords* or a *FoodGroupCode*. The operation returns *NDBNumber, LongDescription,* and *FoodGroupCode* as the results. The WSMED view named *food* in Table 1 allows SQL queries over this web service operation.

Table 1. **WSMED view *food***

ndb	keyword	descr	gpcode
19080	Sweet	Candies	1900
……	……	……	……

For example, the following SQL query to the view *food* retrieves the description of foods that have food group code equal to 1900 and keyword 'Sweet':

```
select descr
from   food
where  gpcode='1900' and keyword ='Sweet';
```

The view *food* is defined as follows:

```
create SQLview food (Charstring ndb,
   Charstring keyword,Charstring descr, Charstring gpcode)
as multidirectional
  ("ffff" select ndb, "",descr, gpcode
          where foodDescr("","")= <ndb,descr,gpcode>)
  ("fffb" select ndb, "",descr
          where foodDescr("",gpcode)= <ndb,descr,gpcode>)
  ("fbff" select ndb,descr,gpcode
          where foodDescr(keyword, "")= <ndb,descr,gpcode>)
  ("fbfb" select ndb, descr
           where foodDescr(keyword,gpcode)
                 = <ndb,descr,gpcode>)
```

Figure 2: WSMED view definition

A given WSMED view can access many different web service operations in different ways. When the user defines a WSMED view he can specify the view by several different declarative queries, called *capability definitions*, using an object oriented query language called WQL having special web service oriented data types. Each capability definition implements a different way of retrieving data through web service operations using WQL. Different capability definitions can be defined based on what view attributes are known or unknown in a query, called the *capability binding patterns*. The query optimizer automatically chooses the most promising capability definitions for a given query to a WSMED view. Each capability definition provides a different way of using the web service operations to retrieve food items. The capability binding patterns of the view *food* are:

1. *ffff*- all the attributes of the view are free in the query. That is, the query does not specify any attribute selection value. In this case the capability definition specifies that all food items should be returned.
2. *fffb*- a value is specified only for fourth attribute *gpcode*. This means that the capability definition returns all food items for a given food group code.
3. *fbff*- a value is specified in the query only for the second attribute *keyword*, i.e. all food items associated with the given keyword are retrieved.
4. *fbfb*- both the values *keyword* and *gpcode* are specified in the query, finding the relevant food items.

In our example query the binding pattern is *fbfb*. The capability definitions are defined as declarative WQL queries that all call a function *foodDescr* in different ways. The function *foodDescr* is defined as a WQL query that wraps the web service operation *SearchFoodByDescription* given two parameters *foodkeywords* and *foodgroupcode*. It selects relevant pieces of a call to the operation *SearchFoodByDescription* to extract the data from the data structure returned by the operation.

To simplify sub-queries and provide heuristics for estimating selectivities, it is important for the system to know what attributes in the view are (compound) keys.

Therefore, the user can specify *key constraints* for a given view and set of attributes by a system function *declare_key*, e.g.:

```
declare_key("food", {"ndb"});
```

Key constraints are not part of WSDL and require knowledge about the semantics of the web service. In our example web service the attribute *ndb* is the key. The attributes are specified as a set of attribute names for a given view *(e.g. {"ndb"})*. Several keys can be specified by several calls to *declare_key*.

The query optimizer may also need to estimate the cost to invoke a capability and the estimated size of its result, i.e. its *fanout*. Costs and fanouts can be specified explicitly by the user if such information is available. However, normally explicit cost information is not available and the cost is then estimated by a *default cost model* that uses available semantic information such as signatures, keys, and binding patterns to roughly estimate costs and fanouts. Key constraints will be shown to be the most important semantic enrichment in our example, and additional costing information is not needed.

3.1 Capability definition function

The function *foodDescr*, used in the capability definitions in Figure 2, has the following definition:

```
1.create function foodDescr (Charstring fkw,
2.                           Charstring fgc)
3.          ->Bag of <Charstring ndb,Charstring descr,
4.                    Charstring gpcode>
5. as select re["NDBNumber"],re["LongDescription"],
6.           re["FoodGroupCode"]
7.    from  Record out, Record re
8.    where out =
9.      cwo("http://ws.strikeiron.com/USDAData?WSDL",
10.         "USDAData",
11.         "SearchFoodByDescription",
12.         {fkw, fgc})
13.    and re in out["SearchFoodByDescriptionResult"];
```

Given a food keyword, *fkw*, and a group code, *fgc*, the function *foodDescr* returns a bag of result rows extracted from the result of calling the web service operation named *SearchFoodByDescription*. Any web service operation can be called by the built-in generic function *cwo* (line 9). Its arguments are the URI of WSDL document that describes the service (line 9), the name of the service (line 10), an operation name (line 11), and the input argument list for the operation (line 12). The result from *cwo* is bound to the query variable *out* (line 8). It holds the output from the web service operation temporarily stored in WSMED's local database. The system automatically converts the input and output messages from the operation into records and sequences where records are used to represent complex XML elements [7] and sequences represent ordered elements. In our example, the argument list holds the parameters *FoodKeywords* and *FoodGroupCode* (line 12). The result *out* is a record structure from which only the attribute *SearchFoodByDescriptionResult* is extracted (line 13). Extractions are specified using the notation *s[k]*, where *s* is a variable holding a record, and *k* is the name of an attribute.

The function *foodDescr* selects relevant parts of the result from the call to the operation. In our example, the relevant attributes are *NDBNumber, LongDescription,* and *FoodGroupCode,* which are all attributes of a record stored in the attribute *SearchFoodByDescriptionResult* of the result record. Our example web service operation *SearchFoodByDescription* returns descriptions of all available food items when both attributes *foodkeywords* and *foodgroupcode* are empty strings. On the other hand, if *foodkeywords* is empty but *foodgroupcode* is known, the web service operation will return all food with that group code. Similarly, if *foodgroupcode* is empty but *foodkeywords* is known, the web service operation will return all food with that keyword. If both *foodkeywords* and *foodgroupcode* are non-empty, the operation will return descriptions of all food items of the group code with matching keywords. This knowledge about the semantic of the web service operation *SearchFoodByDescription* is used to define the capability definition function in Figure 2.

4. Impact of key constraints

To illustrate the impact of key constraints we define two views in terms of the WSMED view *food*. The view *foodclasses* is used to classify food items while *fooddescriptions* describes each food item:

```
create view foodclasses(ndb, keyword, gpcode)
     as select ndb,keyword,gpcode from food;
create view fooddescriptions(ndb, descr)
     as select ndb, descr from food;
```

This scenario is natural for our example web service that treats *foodclasses* different from *fooddescriptions*. The following SQL query accesses these views.

```
select fd.descr
from   foodclasses fc, fooddescriptions fd
where  fc.ndb=fd.ndb and fc.gpcode='1900';
```

First the example query is translated by the calculus generator (Figure 1b) into a Datalog expression:

```
Query(l) :- foodclasses(ndb,keyword,gpcode) AND  fooddescrip-
tions (ndb,descr) AND descr=l AND gpcode='1900'
```

The definitions of the views *foodclasses* and *fooddescriptions* are defined in Datalog as[1]:

```
foodclasses(ndb, keyword, gpcode) :- food(ndb, keyword, *,
                gpcode).
fooddescriptions(ndb,descry) :- food(ndb, *, descr, *).
```

Given these view definitions the Datalog expression is transformed by the view processor (Figure 1b) into:

```
Query(l) :- food(ndb,*,*,'1900') AND food(ndb,*,l,*).
```

Here the predicate *food* represents our WSMED view. At this point the added semantics that *ndb* is the key of the view play its vital part. Two predicates *p(k,a)* and *p(k,b)* are equal if *k* is a key and it is then inferred that the other attributes are also

[1] '*' means don't care.

equal, i.e. *b=a* [9]. If a key constraint that *ndb* is the key is specified, this is used by a query rewriter to combine the two calls to *food*:

```
Query(l) :- food(*,*,l,'1900').
```

Without knowing that *ndb* is the key the transformation would not apply and the system would have to join the two references to the view *food* in the expanded query. The simplification is very important to attain a scalable query execution performance as shown in Section 5.

The next step is to select the best capability definition for the query. The heuristics is that if more than one capability definition is applicable, the system chooses the one with the most variables bound. Since *l* is the query output and *gpcode* is given, the binding patterns *ffff* and *fffb* both apply, and the system chooses *fffb* because it is considered cheaper. The call to *food* then becomes:

```
Query(l) :- l=foodDescr("","1900").
```

Similar to relational database optimizers, given the definition of *foodDescr*, a cost based optimizer generates the algebra expression in Figure 3a, which is interpreted by the execution engine. The apply operator (γ) calls a function producing one or several result tuples for a given input tuple and bound arguments [14]. By contrast, Figure 3b shows an execution plan for the non-transformed expression where the system does not know that *ndb* is key. It is using a nested loop join (NLJ) to join the capability definitions. An alternative possible better plan based on hash join (HJ) that materializes the inner web service call is shown in Section 5. In case no costing data is available about the capability definitions (which is the case here), the system uses built in heuristics, i.e. a default cost model. In our case the cost based optimizer produces the plan in Figure 3a, which is optimal for our query.

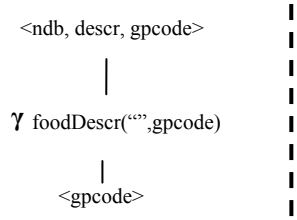

Figure 3a: Full semantic enrichment

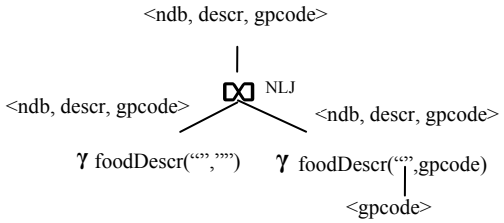

Figure 3b: Naïve execution

5. Query Performance

To determine the impact of semantic enrichments on query processing strategies, we have experimented with four different kinds of query execution strategies. They are:

1. The *naïve implementation* does not use any semantic enrichment at all and no binding pattern heuristics. That is, no *key* is specified for the *food* view definition and no default cost model was used. This makes the capability definition be regarded as a black box called iteratively in a nested loop join since the system does not know

that *foodDescr* returns a large result set when both arguments are empty. The execution plan in Figure 3b shows the naïve plan.

2. With the *default cost model* the system assumes that the view *food* is substantially more expensive to use when attribute *gpcode* is not known than when it is known, i.e. it is cheaper to execute a capability definition where more variables are bound. Still there is no key specified. Figure 5b illustrates the plan using nested loop join.
3. Figure 5a shows the execution plan with the default cost model and a *hash join strategy* where the results from web service operation calls are materialized by using hash join to avoid unnecessary web service calls. This can be done only when the smaller join operand can be materialized in main memory.
4. With *full semantic enrichment* the key of the view is specified. Figure 3a, shows the execution plan. It is clearly optimal.

As shown in Figure 4a, the naïve strategy was the slowest one, somewhat faster than using the default cost model with nested loop join. The default cost model with a hash join strategy scaled significantly better, but requires enough main memory to hold the inner call to *foodDescr*. Figure 4b compares the default cost model with hash join with the performance of full semantic enrichments. The hash join strategy was around five times slower. This clearly shows that semantic enrichment is critical for high performing queries over multi-level views of web services.

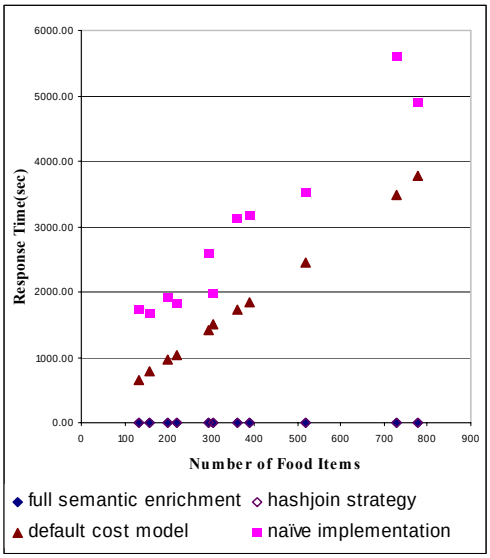

Figure 4a: Performance comparison of four query execution strategies

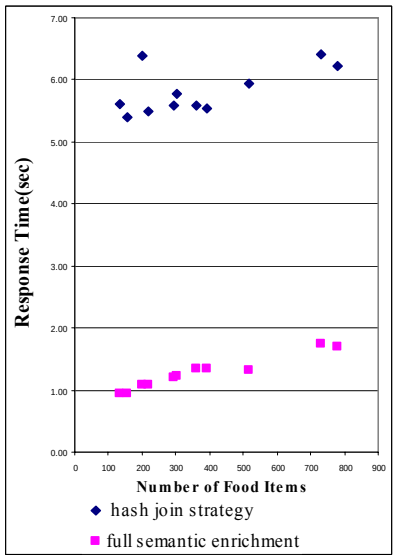

Figure 4b: Performance comparison of hash join and full semantic enrichment execution strategies

The diagrams are based on the experimental results in Table **2** and the experiment was made by using the real values to actually retrieve the results through web service operations. VG, NF, S1, S2, S3, and S4 denote the value used for parameter *gpcode*,

the number of food items (actual fanout), and the execution time in seconds for the four different strategies.

With the naive strategy the system does not use any binding pattern heuristics and will call *foodDescr* with empty strings ($\gamma_{\text{foodDescr}(\text{“”},\text{“”})}$) which produces a large costly result containing all food items in the outer loop. This is clearly very slow.

Table 2. Experimental results

VG	NF	S1	S2	S3	S4
0900	303	1985.14	1512.74	5.77	1.22
0600	390	3177.28	1848.28	5.55	1.33
1400	219	1831.05	1041.74	5.50	1.08
1100	779	4891.13	3785.30	6.22	1.69
2000	157	1655.48	777.31	5.41	0.94
0800	359	3114.28	1723.28	5.59	1.35
0400	201	1914.23	955.38	6.38	1.08
1800	517	3524.34	2452.22	5.93	1.33
2200	132	1741.51	645.03	5.62	0.93

With the default cost model strategy the system assumes that queries over the view *food* produce larger results when the attribute *gpcode* is unknown than when it is known. Based on this the call to *foodDescr* with a known *gpcode* value is placed in the outer loop of a nested loop join. This clearly is a better strategy than the naïve implementation.

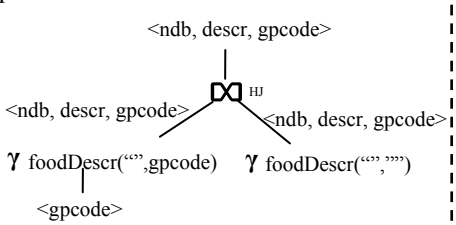

Figure 5a: Execution plan of hash join strategy

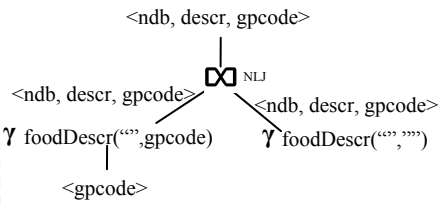

Figure 5b: Execution plan with default cost model

Finally by utilizing key constraints in the WSMED view definition the system will know that the two applications of *foodDescr* can be combined into one call. With this *full enrichment strategy* only one web service operation call is required for execution of the query and no hash join is needed. We notice that this is the fastest and most scalable plan and that it needs no costing knowledge.

6. Related Work

Preliminary results for our method of querying mediated web services were reported in [20].

SOAP [12] and WSDL [5] provide standardized basic interoperation protocols for web services but no query or view capabilities. The SQL 2003 standard [8][26] has facilitates to combine SQL with *XML Query language* (XQuery) [3] to access both ordinary SQL-data and XML documents stored in a relational database. By contrast, we optimize SQL queries to views over data returned by invoking web services and we use semantic query transformations to improve the performance.

The formal basis for using views to query heterogeneous data sources is reviewed in [10][15][25]. As some other information integration approaches, e.g. [11][29], we also use binding patterns as one of our semantic enrichments to access data sources with limited query capabilities. We define semantically enriched declarative views extracting data from the results of each web service operations in terms of an object-oriented query language. In [1] an approach is described for optimizing web service compositions by procedurally traversing ActiveXML documents to select embedded web service calls, without providing view capabilities.

WSMS [22] also provide queries to mediated web services. However, they concentrate on optimizing pipelined execution of web service queries while we utilize semantic enrichments for efficient query processing over multi-level views of web services. XLive [6] is a mediator for integrating heterogeneous sources including web service sources with specific wrappers based on XML standards. In contrast we deploy a generic wrapper that can call any web service.

In particular, unlike the other works, we show that key constraints significantly improve performance of queries to multi-level views of web services with different capabilities.

7. Conclusions and future work

We devised a general approach to query data accessible through web services by defining relational views of data extracted from the result SOAP messages returned by web service operations. Multi-level relational views of web service operations can be defined. The system allows SQL queries over these WSMED views. The view extractions are defined in terms of an object oriented query language. The query performance is heavily influenced by knowledge about the semantics of the specific web service operations invoked and all such information is not provided by standard web service descriptions. Therefore the user can complement a WSMED view with semantic enrichments for better query performance. Our experiments showed that *binding patterns* combined with *key constraints* are essential for scalable performance when other views are defined in terms of WSMED views.

Strategies for parallel pipelined execution strategies of web service operation calls as in WSMS [22] should be investigated. The pruning of superfluous web service operation calls is crucial for performance. The adaptive approaches in [2][17] should be investigated where useless results are dynamically pruned in the early stage of query execution. Currently the semantic enrichments are added manually. Future work could investigate when it is possible to automate this and how to efficiently verify that enrichment is valid. For example, determination of key constraints is currently added manually, and this could be automated by querying the source. Another issue is how to minimize the required semantic enrichments by self tuning cost modeling techniques [16] based on monitoring the behavior of web service calls.

The semantic web is an emerging prominent approach for the future data representations where WSDL working groups are proposing standards to incorporate semantic web representations [21]. It should be investigated how mediate of web services based on such semantic web representations.

Acknowledgements

This work is supported by Sida.

References

[1] S. Abiteboul et al., Lazy query evaluation for active XML, *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Managementof Data*, 227–238, 2004.

[2] R. Avnur, and J. M. Hellerstein, Eddies: Continuously adaptive query processing, *Proc. SIGMOD conference*, 2000.

[3] S.Boag, D.Chamberlin, M.F. Fernández, D.Florescu, J.Robie, and J.Siméon, XQuery 1.0: An XML Query LanguageW3C Candidate Recommendation, *published online at http://www.w3.org/TR/xquery/*, 2006

[4] D.Booth, H.Haas, F.McCabe, E.Newcomer, M.Champion, C.Ferris, and D.Orchard, Web Services Architecture,W3C Working Group Note, *published online at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/* , 2004

[5] E.Christensen, F.Curbera, G.Meredith, and S. Weerawarana, *Web services description language (WSDL) 1.1.*, W3C, http://www.w3.org/TR/wsdl, 2001.

[6] T.Dang Ngoc, C.Jamard, and N.Travers , XLive : An XML Light Integration Virtual Engine, *Proc. of BDA*, 2005

[7] D.C. Fallside, and P.Walmsley, XML Schema Part 0: Primer Second EditionW3C Recommendation, *published online at http://www.w3.org/TR /xmlschema-0/*, 2004

[8] A.Eisenberg, and J.Melton, SQL/XML is Making Good Progress, *ACM SIGMOD Record*, 31(2), June 2002

[9] G. Fahl, and T. Risch, Query Processing over Object Views of Relational Data, *The VLDB Journal* , 6(4), 261-281, 1997.

[10] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A.Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J.Widom, The TSIMMIS Approach to Mediation: Data Models and Languages, *In Journal of Intelligent Information Systems*, 8(2): 117-132, 1997

[11] H.Garcia-Molina, J.D Ullman, and J.Widom, *Database Systems: The Complete Book,* ISBN 0-13-098043-9, Prentice Hall, 1047-1069, 2002.

[12] Google SOAP Search API (Beta), *published online at http://code.google.com/apis /soapsearch/*

[13] M.Gudgin, M.Hadley, N.Mendelsohn, J.Moreau, and H.Frystyk Nielsen, SOAP Version 1.2 Part 1: Messaging Framework,W3C Recommendation, *published online at http://www.w3.org/TR/soap12-part1/* ,2003

[14] L.M.Haas, D. Kossmann, E. Wimmers, and J .Yang, Optimizing queries across diverse data sources, *Proc. Very Large Database Conference(23rd VLDB),* 1997

[15] A.L.Halevy, Answering queries using views: A survey, *VLDB Journal*, 4(10), 270-294, 2001.

[16] Z.He, B.S.Lee, and R.Snapp, Self-Tuning Cost Modeling of User-Defined Functions in an Object-Relational DBMS, *ACM Transactions on Database Systems*, 30(3), 812-853, 2005.

[17] Z.G.Ives, A.Y.Halvey, and D.S.Weld, Adapting to Source Properties in Processing Data Integration Queries, *Proc. SIGMOD conference*, 2004

[18] W. Litwin, and T. Risch, Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *Proc. IEEE Transactions on Knowledge and Data Engineering,* 4(6), pp. 517-528, 1992

[19] SAAJ Project, *published online at https://saaj.dev.java.net/*

[20] M.Sabesan, T.Risch, and G.Wikramanayake, Querying Mediated Web Services**,** *Proc. 8th International Information Technology Conference (IITC 2006*), 2006

[21] Semantic Web Activity, W3C Technology and Society domain, *published online at http://www.w3.org/2001/sw/*

[22] U.Srivastava, J.Widom, K.Munagala, and R.Motwani, Query Optimization over Web Services, *Proc Very Large Database Conference(VLDB 2006),* 2006

[23] The Castor Project, *published online at http://www.castor.org/index.html*

[24] The Web Services Description Language for Java Tool kit(WSDL4J), *published online http://sourceforge.net/projects/wsdl4j*

[25] J.D.Ullman, Information Integration Using Logical Views, *Proc. 6th International Conference on Database Theory (ICDT '97)*, 19-40, 1997.

[26] XML-Related specifications (SQL/XML), *published online at http://www.sqlx.org/SQL-XML-documents/5FCD-14-XML-2004-07.pdf*, 2005

[27] Web Service USDAData, *published online http://ws.strikeiron.com/USDAData?DOC&page=proxy*

[28] WSDL document for USDAData web service, *published online http://ws.strikeiron.com/USDAData?WSDL*

[29] V.Zadorozhny, L.Raschid, M.E.Vidal, T.Urban, and L.Bright, Efficient Evaluation of Queries in a Mediator for WebSources, *Proc. of the 2002 ACM SIGMOD international conference on Management of data*, 85-96, 2002.

S-FrameWeb: a Framework-Based Design Method for Web Engineering with Semantic Web Support

Vítor Estêvão Silva Souza[1], Thiago Wotikoski Lourenço[1],
Ricardo de Almeida Falbo[1], Giancarlo Guizzardi[1,2]

[1] Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514
29075-910 Vitória – ES, Brazil
[2] Laboratory for Applied Ontology, Polo Tecnologico, Via Solteri, 38
38100 Trento, Italy
{vitorsouza, twotikoski}@gmail.com, falbo@inf.ufes.br, guizzardi@loa-cnr.it

Abstract. The Web Engineering area is evolving fast. Many methods and frameworks to support Web Information Systems (WISs) development have already been proposed. Particularly, the use of frameworks and container-based architectures is state-of-the-practice. Motivated by this scenario, we have proposed a method for designing framework-based WISs called FrameWeb. However, we should consider that the Semantic Web has been gaining momentum in the last few years. The idea is that the information on the Web should be available in machine-processable formats so that software agents could reason with it. This paper presents an extension to FrameWeb, called S-FrameWeb, that aims to support the development of Semantic WISs.

Keywords: Web Engineering, Web Information Systems, Frameworks, Semantic Web.

1 Introduction

The Semantic Web is being considered the future of the World Wide Web (WWW). Coined by Berners-Lee [1], the term represents an evolution of the current WWW, referred by some as the "Syntactic Web". In the latter, information is presented in a way that is accessible only to human beings, whereas in the former data is presented both in human-readable and machine-processable formats, in order to promote the development of software agents that would help users carry their tasks on the Web.

However, for Berners-Lee's vision to become a reality, Web authors and developers must add semantic annotations to their Web Applications. This is neither an easy nor a small task and support from tools and methods is needed.

Methods already exist for the development of Web Information Systems (WISs), such as WAE [2], OOWS [3] and OOHDM [4]. In this context, we proposed a method for the design of WISs that are based on frameworks, called FrameWeb (Framework-Based Design Method for Web Engineering) [5]. FrameWeb proposes a basic

architecture for developing WISs, a set of activities and a UML profile for a set of design models that brings concepts used by some categories of frameworks. The idea is that the use of FrameWeb would further improve team productivity by using a modeling language that would allow designers to produce diagrams that represent framework concepts, and developers (maybe, in the future, CASE tools) to directly translate these diagrams to code [5].

To help developers build WISs with semantic annotations, we decided to work on an extension of FrameWeb, called S-FrameWeb. The idea is to incorporate into the method activities and guidelines that drive the developer in the definition of the semantics of the WISs, resulting in a "Semantic Web-enabled" application.

This paper presents S-FrameWeb, and it is organized as follows: section 2 discusses some issues concerning WebE and the Semantic Web and briefly presents FrameWeb. Section 3 presents S-FrameWeb and how it proposes to build "Semantic Web-enabled" applications. Section 4 discusses related work. Finally, section 5 presents our conclusions and potential future work.

2 Web Engineering and the Semantic Web

Web Engineering (WebE) has been defined as "the establishment and use of engineering principles and disciplined approaches to the development, deployment and maintenance of Web-based Applications" [6]. WebE was conceived at a time when Web Applications (WebApps) were developed in an ad-hoc manner, without a methodology or software process to support developers. Nowadays, however, there are many methods, such as WAE [2], OOWS [3] and OOHDM [4], that are being used.

Also, technologies for codifying WebApps have evolved. The use of frameworks to support the construction of complex Web Information Systems (WISs) is state-of-the-practice. Container-based architectures, such as the most recent version of the Java Enterprise Edition [7] standard, also borrow many concepts from these frameworks. Both frameworks and container-based architectures promote the reuse of a commonly used application infrastructure and improve productivity.

There are many different frameworks available for coding WISs. However, it is possible to separate them into few categories organized by purpose [5]. Table 1 lists four of these categories: Front Controller [8], Decorator, Object/Relational (O/R) Mapping [9] and Dependency Injection frameworks [10]. Other kinds of frameworks include: Aspect-Oriented Programming frameworks, Authentication & Authorization frameworks, Search engines, etc.

Table 1. Frameworks that form a commonly used infrastructure for Web Applications.

Framework	Purpose
Front Controller	Also known as MVC frameworks, defines an architecture that separates the functionality of a WebApp from its presentation based on the Model-View-Controller pattern [11].

Framework	Purpose
Decorator	Based on the Decorator design pattern [11], automates the task of making every web page of the site have the same layout (header, footer, navigation bar, colors, images, etc).
Object/Relational (O/R) Mapping	Provides automatic and transparent persistence of objects to tables of a RDBMS using meta-data that describe the mapping between both worlds [9].
Dependency Injection	Allows the developer to program to interfaces [10] and specify the concrete dependencies in a configuration file. The idea is that classes that depend on services from different tiers would declare an association with an interface instead of the concrete implementation. This facilitates, for instance, the replacement of the real service class with a mock object for unit testing.

These frameworks can substantially change the architecture and the components that must be developed for a WIS. That motivated the proposition of the Framework-based Design Method for Web Engineering (FrameWeb). The interested reader should refer to [5] and [12] for detailed information. FrameWeb proposes:

- A standard architecture for Web Applications that integrates with those frameworks by separating their concerns into different packages;
- A UML profile suited for the construction of four kinds of design models that represent framework components from different packages: Domain Model, Persistence Model, Navigation Model and Application Model [12];
- Although FrameWeb does not prescribe a software process, allowing organizations to use the process that suits them best, it suggests that use cases and class diagrams are used during requirement analysis.

FrameWeb's standard architecture divides the system into three main tiers, as shown in Figure 1. The Presentation Logic tier contains elements related to Web-based user interfaces. The **`Controller`** package gathers the action classes that integrate with the Front Controller framework, while the **`View`** package contains Web pages, style sheets, images and other files related with the exhibition of information.

The Business Logic tier also contains two packages: **`Domain`** and **`Application`**. The former includes classes that represent domain concepts modeled during requirement analysis. The latter comprises classes that implement functionalities represented as use cases during that same stage.

The last tier regards Data Access. The **`Persistence`** package contains classes that communicate with the Object/Relational (O/R) Mapping framework to create, retrieve, update and delete domain objects from the persistence store. FrameWeb suggest the use of the Data Access Object pattern [8] for this package.

The dependency associations in figure 1 show how these packages interact. User stimuli come from **`View`** components and reach the **`Controller`** classes by means of the MVC framework. The action classes in `Controller` call methods from **`Application`** classes, which manipulate **`Domain`** objects and also depends on the

`Persistence` of these objects. Associations stereotyped as `<<weak>>` represent loose coupling. For instance, the packages in the Presentation tier do not create and manipulate domain objects directly, but use them to display data or pass them around as parameters, using a domain-driven approach[1].

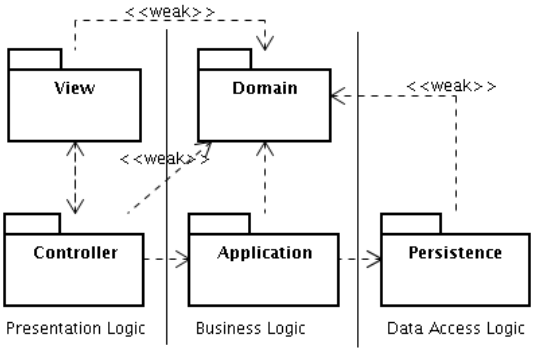

Fig. 1. FrameWeb's standard architecture for WIS [5].

To model classes and other components that belong to the different packages of the standard architecture, FrameWeb uses UML's lightweight extension mechanism to create a profile for designing four different kinds of diagrams [12] during system design, which are summarized in Table 1. All of them are based on UML's class diagram, but represent components from different packages that integrate with different frameworks. Interested readers should refer to [12] for further details.

Table 2. Diagrams built during the design of a WIS using FrameWeb.

Diagram	Purpose
Domain Model	Represents domain classes modeled during analysis, complemented with platform-dependent information (attribute types, association navigabilities, etc.) and O/R mappings (which are more easily represented in this model instead of the Persistence Model because the attributes are modeled here). Guides the implementation of classes from the `Domain` package and also the configuration of the O/R framework.
Persistence Model	Shows DAO classes that are responsible for the persistence of domain objects and the existence of specific queries to the database. Every domain class that requires persistence should have a DAO interface and an implementation for each persistence technology used. Guides the codification of the DAOs, which belong to the `Persistence` package, and the creation of specific database queries on the O/R framework.

1 In this context, the domain-driven approach (referred to as model-driven approach by the framework's documentation) consists of using an instance of a domain class as wrapper for its attributes when they are passed as parameters.

Diagram	Purpose
Navigation Model	Displays components from the presentation tier, such as web pages, HTML forms, templates, binary files and action classes, and their relationships among themselves. Guides the implementation of action classes (`Controller` package), other view components (`View` package) and the configuration of the Front Controller framework.
Application Model	Models the interfaces and classes that implement use case functionalities and the dependency chain from the action classes (which depend on them) until the DAOs (which they depend on). Guides the codification of classes from the `Application` package and the configuration of the Dependency Injection framework.

FrameWeb provides a way for modeling WIS that is suited for those based on frameworks. There is no indication, however, on how to provide semantic annotations that could make the WIS available for Semantic Web agents to reason with it. Reasoning means that software agents are able to understand the information presented by Web pages and take sensible actions according to a goal that was previously given. The most usual way for agents to understand the contents of a website is by semantically annotating the pages using formal knowledge representation structures, such as ontologies.

An ontology is an engineering artifact used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of its vocabulary words [13]. Along with ontology representation languages such as OWL [14], they are able to describe information from a website in formal structures with well-defined inference procedures that allow software agents to perform tasks such as consistency checking, establish relation between terms, systematic classification and infer information from explicitly defined information in this structure.

If the ontology is built (using one of many methodologies for their construction [15]) and given the availability of tools such as OILEd[2] and Protégé[3], the annotation of static Web pages with OWL has become a straightforward task.

However, few websites are composed strictly by static pages. What is commonly seen is Web pages being dynamically generated by software retrieving information on-the-fly from data repositories such as relational databases. These data-intensive websites have the advantage of separating data and layout, but also have limitations such as being invisible to search engines and not being comprehensible by software agents [16]. Thus, an automated way of annotating dynamic Web pages is needed.

One way to do that is, when a Web page is requested at the Web server, it must recognize if the request comes from a human agent or a software agent. In the latter case, instead of generating a HTML human-readable Web page, the server should return a document written in an ontology specification language (e.g. OWL) containing meta-data about the information that would be conveyed in the page.

2 http://oiled.man.ac.uk/

3 http://protege.stanford.edu/

Although the solution seems appropriate, many aspects still need to be addressed, such as: how are the agents supposed to find the Web page? How will they know the correct way to interact with it? For instance, how will they know how to fill in an input form to submit to a specific request?

Hepp [17] advocates that semantic annotation of static or dynamic data is not enough and that the original vision of the Semantic Web can only be achieved by the utilization of Semantic Web Services. A Web Service is "a software system designed to support interoperable machine-to-machine interaction over a network" [18]. Web Services provide a nice way for software agents to interact with other systems, requesting services and processing their results.

Many researchs are now directed to the use of Web Services on the Semantic Web. Semantic Web Services are formed by adding semantic annotations to Web Services so they become interpretable by software agents. Meta-data about the service are written in a markup language, describing its properties and capacities, the interface for its execution, its requirements and the consequences of its use [19]. Many tasks are expected to be automated with this, including service discovery, invocation, interoperation, selection, composition and monitoring [20].

3 Semantic FrameWeb

The main goal of S-FrameWeb is to make WISs "Semantic Web-enabled". This should be accomplished by the Front Controller framework, which identifies if requests come from human or software agents. In the former case, the usual Web page is presented, while in the latter, an OWL document is returned.

To fulfill its purpose, S-FrameWeb adds three new steps to FrameWeb's software process: domain analysis, ontology design and ontology implementation. A suggested software process is shown in figure 2. These steps are further discussed next.

3.1 Domain Analysis

To bring a WIS to the Semantic Web it is imperative to formally describe its domain. As stated in section 2, the most usual way of doing this is by constructing an ontology. S-FrameWeb indicates the inclusion of a Domain Analysis activity in the software process for the development of a domain ontology (we don't use the term "domain model" to avoid confusion with FrameWeb's Domain Model).

Domain Analysis is "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain" [21, 22]. When a software is built, the purpose is to solve a problem from a given domain of expertise, such as medicine, sales or car manufacturing. If the domain is analyzed prior to the analysis of the problem, the knowledge that is formalized about the domain can be reused when another problem from the same domain needs a software solution [22].

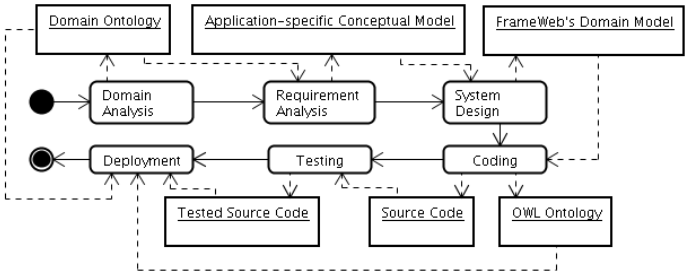

Fig. 2. The software process suggested by S-FrameWeb.

For a diagrammatic representation of the ontology, S-FrameWeb uses OMG's[4] Ontology Definition Metamodel (ODM) [23], "a language for modeling Semantic Web ontologies in the context of MDA" [24]. ODM defines an ontology UML profile that allows developers to represent ontologies in UML class diagrams.

The output of Domain Analysis is an ontology that represents concepts from the problem domain. The ontology's diagram can be reused in the Requirement Analysis phase to produce the application's conceptual model, which will later be refined and become FrameWeb's Domain Model (FDM) during system design.

Table 3 summarizes the evolution of the models throughout the software process.

Table 3. Models produced by the software process suggested by S-FrameWeb

Activity	Artifact	What the model represents
Domain Analysis	Domain Ontology	Concepts from the domain to which the software is being built. Modeled in ODM, but converted to OWL for deployment.
Requirement Analysis	Conceptual Model	Concepts that are specific to the problem being solved. Modeled in ODM.
System Design	FrameWeb's Domain Model (FDM)	Same as above plus OR mappings. Modeled using S-FrameWeb's UML profile.
Coding	OWL code	OWL representation of FDM, without OR mappings.

Figure 3 shows the conceptual model for a very simple culinary recipes WIS called "Cookbook". This application includes the registry of recipes and a simple search feature. After the domain of culinary was analyzed and an ontology modeled, the conceptual model was built in ODM using only the classes that were required for this particular application.

The stereotype `<<OntClass>>` indicates domain classes, `<<ObjectProperty>>` models associations between domain classes, `<<DataType>>` represents XML data types and `<<DatatypeProperty>>` models associations between classes and data types.

4 Object Management Group – http://www.omg.org/ontology/

The reader accustomed with UML conceptual models may notice that associations are represented as classes in ODM. This is because in OWL, associations are independent from classes and, for instance, can form their own subsumption hierarchy. More on ODM's semantics can be found at [23].

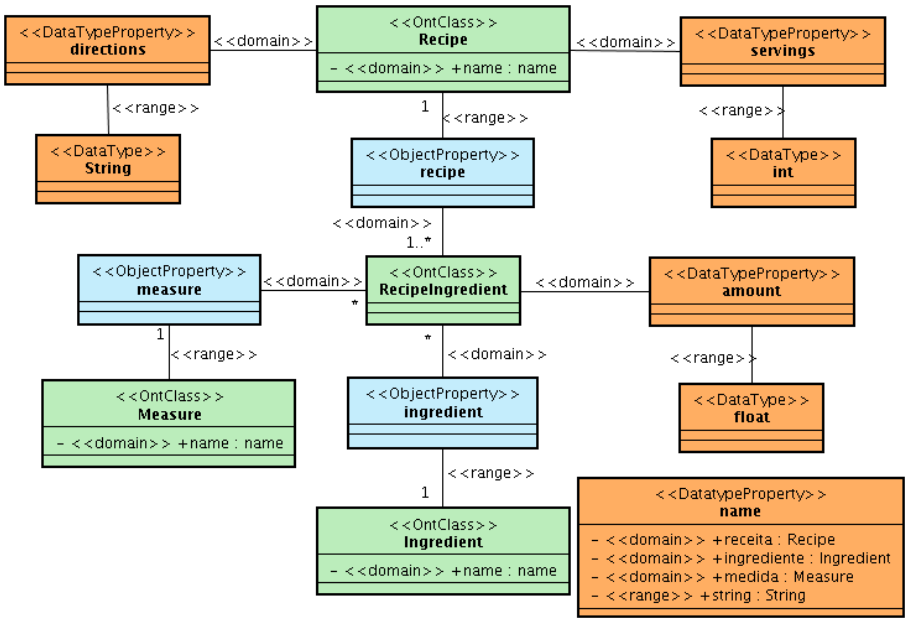

Fig. 3. The conceptual model for the Cookbook application.

3.2 Ontology Design

During the design phase of the software process, FrameWeb prescribes the construction of a Domain Model (referred to as FDM), which shows classes that represent concepts from the problem domain and their object/relational mappings.

S-FrameWeb proposes an extension to this diagram, mixing both FrameWeb's and ODM's UML profiles to build FDM. Based on the conceptual model, the designer should simplify ODM's syntax and add the OR mappings. Figure 4 shows the FDM for the Cookbook application. We can see that **`<<DataType>>`** elements were replaced by class attributes (this should happen only in simple cases – for instance, when they do not participate in a subsumption hierarchy or when the datatype is not structured) and that some mappings were included (`{not null}` and `{cascade=all}` [12]).

3.3 Ontology Implementation

Finally, at the coding phase, the domain ontology and the FDM should be coded in OWL and deployed into the WIS. In this context, “deploy” means placing the OWL

file in a predetermined location so the Front Controller framework can read it. Because the models are represented in ODM, their codification in OWL are straightforward (ODM proposes graphical representations for OWL constructs) and, in the near future, probably this can be automated by CASE tools.

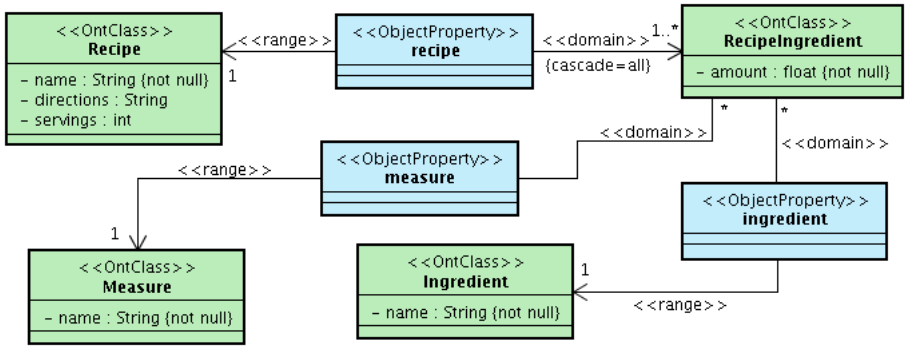

Fig. 4. S-FrameWeb's Domain Model for the Cookbook application.

During the execution of the WIS, the Front Controller provides an infrastructure that identifies when the request comes from a software agent, reads both ontology's and FDM's OWL files and responds to the request based on the execution of an action and reasoning over the ontologies. Since existing Front Controller frameworks do not have this infrastructure, a prototype of it was developed and is detailed next.

3.4 Front Controller Infrastructure

To experiment S-FrameWeb in practice, we have extended the Struts[2] framework[5] to recognize software agents requests and respond with an OWL document, containing the same information it would be returned by that page, but codified as OWL instances. Together with the OWL files for the domain ontology and the FDM, software agents can reason about the data that resulted from the request.

Figure 5 shows this extension and how it integrates with the framework. The client's web browser issues a request for an action to the framework. Before the action gets executed, the controller automatically dispatches the request through a stack of interceptors, following the pipes and filters architectural style. This is an expected behavior of Struts[2] and most of the framework's features are implemented as interceptors (e.g., to have it manage a file upload, use the `fileUpload` interceptor).

An “OWL Interceptor” was developed and configured as first interceptor of the stack. When the request is passing through the stack, the OWL Interceptor verifies if a specific parameter was sent by the agent in the request (e.g. `owl=true`), indicating that the action should return an “OWL Result”. If so, it creates a pre-result listener that will deviate successful requests to another custom-made component that is responsible for producing this result, which we call the “OWL Result Class”. Since

5 http://struts.apache.org/2.x/

this result should be based on the application ontology, it was necessary to use an ontology parser. For this purpose, we chose the Jena Ontology API, a framework that provides a programmatic environment for many ontology languages, including OWL.

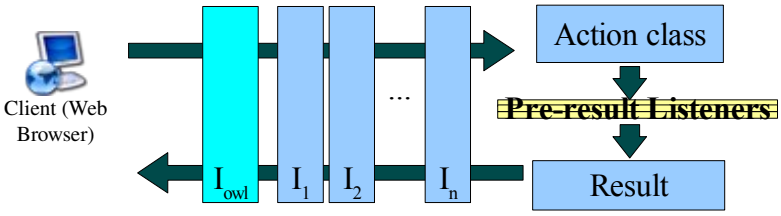

Fig. 5. S-FrameWeb's Front Controller framework extension for the Semantic Web.

Using Jena and Java's reflection API, the OWL Result Class obtains all accessor methods (JavaBeans-standardized get*Property*() methods) of the Action class that return a domain object or a collection of domain objects. These methods represent domain information that is available to the client (they are called by the result page to display information to the user on human-readable interfaces). They are called and their result is translated into OWL instances, which are returned to the client in the form of an OWL document.

4 Related Work

As the acceptance of the Semantic Web idea grows, more methods for the development of "Semantic Web-enabled" WebApps are proposed.

The approach which is more in-line with our objectives is the Semantic Hypermedia Design Method (SHDM) [25]. SHDM is a model-driven approach for the design of Semantic WebApps based on OOHDM [4]. SHDM proposes five steps: Requirement Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation.

Requirements are gathered in the form of scenarios, user interaction diagrams and design patterns. The next phase produces a UML-based conceptual model, which is enriched with navigational constructs in the following step. The last two steps concern interface design and codification, respectively.

SHDM is a comprehensive approach, integrating the conceptual model with the data storage and user-defined templates at the implementation level to provide a model-driven solution to WebApps development. Being model-driven facilitates the task of displaying agent-oriented information, since the conceptual model is easily represented in OWL.

While the approach is very well suited to content-based WebApps, WIS which are more centered in providing functionalities (services) are not as well represented by SHDM. The proposal of FrameWeb was strongly motivated by the current scenario where developers are more and more choosing framework or container-based

architectures to create applications for the Web. S-FrameWeb builds on top of FrameWeb to provide semantics to these functionality-based WebApps.

5 Conclusions and Future Work

S-FrameWeb suggests a software process that facilitates the development of Semantic WISs by automating certain tasks concerning the generation of semantic annotations on dynamic Web pages. However, the following limitations have already been identified and are bound to future work:

- Software agents must know how to find the Web pages. Pages that are linked by others can be found by search engines, but that is not the case with the ones that represent the request for a service. The research on Web Service discovery could provide some insight on this issue;
- Agents must speak a common language to understand Web pages. If an instance of "table" is returned, how will the agent know if it's a "piece of furniture", "a systematic arrangement of data usually in rows and columns"[6] or any other meaning? The use of top-level ontologies, such as Dolce[7], should be considered for this matter;
- Works in the area of Semantic Web Services [19, 20, 26] suggest another way to deal with the issue of annotation of WISs. S-FrameWeb should be implemented to use Web Services in the future to compare both approaches;
- The infrastructure prototype was developed for the Struts[2] framework and currently has some limitations that have to be addressed. Other frameworks should be extended so S-FrameWeb can be used in different platforms.

Acknowledgments. This work was accomplished with the financial aid of CAPES, an entity of the Brazilian Gov't dedicated to scientific and technological development.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001) n. 284, p. 34-43
2. Conallen, J.: Building Web Applications with UML. 2nd edn. Addison-Wesley (2002)
3. Fons, J.; Valderas, P.; Ruiz, M.; Rojas, G.; Pastor, O: OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, USA (2003)
4. Schwabe, D., Rossi, G.: An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems 4 (4). Wiley and Sons (1998)
5. Souza, V. E. S., Falbo, R. A.: FrameWeb: A Framework-based Design Method for Web Engineering. Proceedings of the Euro American Conference on Telematics and Information Systems, Faro, Algarve, Portugal (2007)

6 Merriam-Webster Online Dictionary (http://www.m-w.com)

7 More about Dolce at http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf

6. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web Engineering: A New Discipline for Development of Web-based Systems. Proceedings of the First ICSE Workshop on Web Engineering. IEEE, Australia (1999)
7. Shannon, B.: Java™ Platform, Enterprise Edition (Java EE) Specification, v5. Sun Microsystems (2006)
8. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall / Sun Microsystems Press (2001)
9. Bauer, C., King, G.: Hibernate in Action. 1st edn. Manning (2004)
10. Fowler, M.: Inversion of Control Containers and the Dependency Injection Pattern (http://www.martinfowler.com/articles/injection.html). Captured on July 19th (2006)
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
12. Souza, V. E. S., Falbo, R. A.: A Language for Modeling Framework-based Web Information Systems. Proceedings of the 12th International Workshop on Exploring Modeling Methods in System Analysis and Design. Trondheim, Norway (2007)
13. Guarino, N.: Formal Ontology and Information Systems. Proceedings of the 1st International Conference on Formal Ontologies in Information Systems. IOS Press. Trento, Italy (1998) p. 3-15.
14. W3C: OWL Web Ontology Language Guide, fev. 2004 (http://www.w3.org/TR/owl-guide/). Captured on: November 13th (2006)
15. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering. Springer (2005)
16. Stojanovic, L., Stojanovic, N., Volz, R.: Migrating data-intensive Web Sites into the Semantic Web. Proceedings of the 2002 ACM symposium on Applied computing. ACM. Madrid, Spain (2002) p. 1100-1107
17. Hepp, M.: Semantic Web and semantic Web services - Father and Son or Indivisible Twins? IEEE Internet Computing. IEEE (2006) v. 10, n. 2, p. 85-88
18. W3C: W3C Glossary and Dictionary (http://www.w3.org/2003/glossary/). Captured on: January 23rd (2007)
19. McIlraith, S. A., Son, T. C., Zeng, H.: Semantic Web Services. Intelligent Systems. IEEE (2001) v. 16, n. 2, p. 46-53
20. Narayanan, S., McIlraith, S. A.: Simulation, Verification and Automated Composition of Web Services. Proceedings of the 11th international conference on World Wide Web. ACM. Hawaii, USA (2002) p. 77-88
21. Neighbors, J. M.: Software Construction Using Components. Ph.D. Thesis. Department of Information and Computer Science, University of California, Irvine (1981)
22. Falbo R. A., Guizzardi, G., Duarte, K. C. : An Ontological Approach to Domain Engineering. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002). Ischia, Italy (2002). pp. 351- 358
23. OMG: Ontology Definition Metamodel Specification (http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf). Captured on: January 29th (2007)
24. Đurić, D.: MDA-based Ontology Infrastructure. Computer Science and Information Systems. ComSIS Consortium (2004) vol. 1, issue 1
25. Lima, F., Schwabe, D.: Application Modeling for the Semantic Web. First Latin American Web Conference (LA-Web). IEEE-CS Press. Santiago, Chile (2003)
26. Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. Proceedings of the 11th International World Wide Web Conference (WWW 2002). Hawaii, USA (2002)

Personalizing Bibliographic Recommendation under Semantic Web Perspective[1]

Giseli Rabello Lopes, Maria Aparecida Martins Souto,
Leandro Krug Wives, José Palazzo Moreira de Oliveira

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil
{grlopes, souto, wives, palazzo}@inf.ufrgs.br

Abstract. This paper describes a Recommender System for scientific articles in digital libraries for the Computer Science researchers' community. The system employs the Dublin Core metadata standard for the documents description, the XML standard for describing user profile, which is based on the user's *Curriculum*, and on service and data providers to generate recommendations. The main contribution of this work is to provide a recommendation mechanism based on the user academic curriculum reducing the human effort spent on the profile generation. In addition, this article presents and discusses some experiments that are based on quantitative and qualitative evaluations.

Keywords: Recommender System, User profile, Digital library, Semantic Web.

1 Introduction

Today, the scientific publications can be electronically accessed as soon as they are published on the Web. The main advantage of open publications is the minimization of the time and the space barriers inherent to the traditional publication process. In this context, Digital Libraries (DLs) have emerged as the main repositories of digital documents, links and associated metadata. This is a change in the publication process and has encouraged the development of automatic systems to rapidly explore and obtain required information. EPrints [10], DSpace [22], Kepler [16], CITIDEL [4] and BDBComp [12] are examples, among others. Usually, users with different knowledge levels, experiences and interests receive the same information as the answer to their queries. Aiming to avoid these problems, Recommender Systems in DLs have been proposed and developed (e.g., ARIADNE, ResearchIndex, CyberStacks and ARP).

The Recommender Systems involve information personalization. The personalization is related to the ways in which information and services can be tailored to match the specific needs of a user or a community [3]. The human-centered demand specification is not an easy task. One experiences this difficulty when trying to find scientific papers in a good indexing and retrieval system such Scholar Google.

[1] This work was partially supported by the project Pronex FAPERGS, grant 0408933, and Project PerXML CNPq, grant 475743/2004-0. The first and the last authors are partially supported by CNPq.

The query formulation is complex and the fine tuning of the user requirements is a time-consuming task. Few researchers have enough time to spend some hours a week searching for, eventually, new papers in theirs specific research area. This functionality, the query specification, may be reached by the analysis of the user activities, history, information demands, etc.

This article presents a Recommender System to Computer Science researchers and academics. The information and service provided by the system are based on the *Lattes Curriculum Vitae* (*Lattes* CV) [13], a system that registers all the researcher's academic activities and publications with a XML output. The main contribution of this work is to provide a recommendation mechanism based on the user academic curriculum reducing the human effort spent on the profile generation.

The article is organized as follows. We start giving an overview of the background literature and concepts, then the recommender system and detail its architecture and techniques. Finally, we present some quantitative and qualitative experiments to evaluate and validate our system and discuss the results and conclusions of our work.

2 Background

The semantic Web technologies promote an efficient and intelligent access to the digital documents on the Web. The standards based on metadata to describe information objects have two main advantages: computational efficiency during the information harvesting process and interoperability among DLs. The first is a consequence of the increasing use of Dublin Core (DC) metadata standard [8]; the latter has been obtained as a result of the OAI initiative (Open Archives Initiative) [17]. DC metadata standard was conceived with the objective of defining a minimal metadata set that could be used to describe the available resources of a DL. This standard defines a set of 15 metadata (Dublin Core Metadata Element Set - DCMES). Table 1 shows these elements and their associated descriptions.

Table 1. Dublin Core Metadata Element Set, adapted from [8].

Element Name	Description
dc:title	A name given to the resource.
dc:creator	An entity primarily responsible for making the content of the resource.
dc:subject	A topic of the content of the resource.
dc:description	An account of the content of the resource (e.g., abstract).
dc:publisher	An entity responsible for making the resource available.
dc:contributor	An entity responsible for making contributions to the content of the resource.
dc:date	A date of an event in the lifecycle of the resource (typically, *dc:date* will be associated with the creation or availability of the resource).
dc:type	The nature or genre of the content of the resource.
dc:format	The physical or digital manifestation of the resource.
dc:identifier	An unambiguous reference to the resource within a given context (e.g., URL).
dc:source	A reference to a resource from which the present resource is derived.
dc:language	A language of the intellectual content of the resource.
dc:relation	A reference to a related resource.
dc:coverage	The extent or scope of the content of the resource (typically, *dc:coverage* will include spatial location).
dc:rights	Information about rights held in and over the resource.

The main goal of OAI is to create a standard communication way, allowing DLs around the world to interoperate as a federation [21]. The DL metadata harvesting process is accomplished by the OAI-PMH protocol (Open Archives Initiative Protocol for Metadata Harvesting) [18], which defines how the metadata transference between two entities, *data* and *service providers*, is performed. The *data provider* acts by searching the metadata in databases and making them available to a *service provider*, which uses the gathered data to provide a specific service.

Considering that a Recommender System concerns with information personalization, it is essential that it copes with user profile. In our work, the user profile is obtained from the user's *curriculum vitae*, i.e., *Lattes* CV. The *Lattes* CV is a Brazilian Research Council (CNPq) initiative and offers a standard database of researchers and academics curricula. The platform is used: (i) to evaluate the competency of researchers and academics for grant concession; (ii) to select committees' members, consulting people and counselors; and (iii) to assist the evaluation processes of research and post-graduate courses. Thus, all the research personnel must have an updated CV in order to submit research projects or to receive any kind of support from the agencies. It is the main instrument to support the researcher evaluation, as the CV is publicly accessible at the CNPq site the data may be verified by the research community. As a consequence this is the best source for the user profile creation.

Table 2. Lattes CV Metadata Element Subset, adapted from [13].

Metadata Category	Description
Personal information	This category contains general information about the user. Some metadata are: - *cv:name* - *cv:personal-address* - *cv:professional-address*
University degrees	This category contains user's information about his/her academic degrees. Some metadata are: - *cv:graduation-level* (Undergraduate, Master graduate, and PhD. graduate) - *cv:graduation-year* - *cv:monograph-title* - *cv:monograph-keywords* - *cv:monograph-area* - *cv:monograph-advisor*
Language proficiency	This category contains information about the languages that user has any proficiency. Some metadata are: - *cv:language* - *cv:language-skill* (reading, writing, speaking, comprehension) - *cv:language-skill-level* (good, reasonable or little)
Bibliographic production	This category provides user's information about his/her bibliographic publications in proceedings, journals, book chapters, etc. Some metadata are: - *cv:article-title* - *cv:article-keywords* - *cv:article-language* - *cv:article-year*

Table 2 shows a *Lattes* CV metadata elements subset. It presents the categories used in this work to support the recommendation process and their associated descriptions. To better comprehension, the prefix "cv:" is used in this work to reference the metadata elements.

According to [11], there are three different methodologies used in Recommender Systems to perform recommendation: (i) *content-based*, which recommends items classified accordingly to the user profile and early choices; (ii) *collaborative filtering*, which deals with similarities among users' interests; and (iii) *hybrid approach*, which combines the two to take advantage of their benefits. In our work, the *content-based* approach is used, once the information about the user is taken from the *Lattes* CV and is matched with the DC metadata that best describes the articles of a DL.

The recommendation process can be perceived as an information retrieval process, in which user's relevant documents should be retrieved and recommended. Thus, to perform recommendations, we can use the classical information retrieval models such as the Boolean Model, the Vector Space Model (VSM) or the Probabilistic Model [20, 1, 9]. In this work, the VSM was selected since it provides satisfactory results with a convenient computational effort. In this model, documents and queries are represented by terms vectors. The terms are words or expressions extracted from the documents and from queries that can be used for content identification and representation. Each term has a weight associated to it to provide distinctions among them according to their importance. According to [19] the weights can vary continuously between 0 and 1. Values near to 1 are more important while values near to 0 are irrelevant.

The VSM uses an *n*-dimensional space to represent the terms, where *n* corresponds to the number of distinct terms. For each document or query represented, the weights represent the vector's coordinates in the corresponding dimension. The VSM principle is based on the inverse correlation between the distance (angle) among term vectors in the space and the similarity between the documents that they represent. To calculate the similarity score, the cosine (Equation 1) can be used. The resultant value indicates the relevance degree between a query (Q) and a document (D), where w represents the weights of the terms contained in Q and D, and t represents the number of terms (size of the vector). This equation provides ranked retrieval output based on decreasing order of the ranked retrieval similarity values [19].

$$Similarity(Q, D) = \frac{\sum_{k=1}^{t} w_{qk} \cdot w_{dk}}{\sqrt{\sum_{k=1}^{t} (w_{qk})^2 \cdot \sum_{k=1}^{t} (w_{dk})^2}} \quad \textbf{(1)}$$

The same equation is widely used to compare the similarity among documents, and similarly, in our case, Q represents the user profile and D the documents descriptors that are harvested in the DL (see Section 3.2 for details). The term weighting scheme is very important to guarantee an effective retrieval process.

The results depend crucially of the term weighting system chosen. In addition, the query terms selection is fundamental to obtain a recommendation according to the user necessities. Our research is focused in the query terms selection and weighting.

Any person that experienced a bibliographical retrieval may evaluate the process complexity and the difficulty to find the adequate articles. The central idea is to develop an automated retrieval and recommendation system where the price for the user is limited to the submission of an already existing *Lattes* XML CV at subscription time. For a researcher from a country without a similar CV system it will be necessary to substitute the XML CV upload for a Web extracting module that will try to recover the needed metadata from Web pages and, perhaps, from the Scholar Google or other equivalent systems.

3 The Recommender System

Our system focuses on the recommendation of scientific articles to the Computer Science community. The information source to perform recommendations is the Brazilian Computer Science Digital Library (BDBComp) [2], while the user profile is obtained from a Lattes CV subset. However, any DL repository providing DC metadata and supporting the OAI-PMH protocol can be used as a source. An alternative to the user profile generation is under development. This alternative approach is composed by an information retrieval system to gather data from personal homepages and other data sources in order to replace the Lattes CV where the Lattes personal data is not be available.

A DL repository stores digital documents or its localization (web or physical), and the respective metadata. A DL *data provider* allows an agent to harvest documents metadata through the OAI-PMH protocol. Our system handles the documents metadata described with XML in DC standard. The *Lattes* CV and the DC metadata are described as an XML standard document according to the W3C XML Schema, which can be found in [15] for the Lattes CV and in [7] for the DC standard.

3.1 The Recommender System Architecture

In this section we present the architecture elements of our system and its functionalities (Fig. 1). To start the process, the users must supply their *Lattes* CV in the XML version to the system. Whenever a user makes its registration in the system and sends his *Lattes* CV (1), the ***XML Lattes to Local DB*** module is activated and the information about the user's interests is stored in the local database named *User Profile* (2)**.** Then the ***Metadata Harvesting*** module is activated to update the local database *Articles Metadata*. This module makes a request to a DL *data provider* to harvest specific document metadata. It receives an XML document as response (3) and the ***XML DC to local DB*** module is activated (4). This module extracts the relevant metadata to perform the recommendations from the XML document and stores it in the local database named *Articles Metadata* (5). Once the user profile and the articles metadata are available in the local database, the ***Recommendation*** module can be activated (6). The focus is to retrieve articles of a DL that best matches the user profile described through the *Lattes* CV (7).

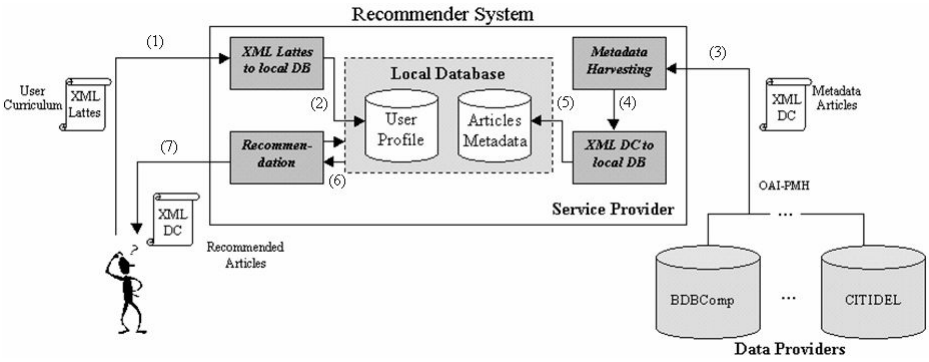

Fig. 1. The recommender system architecture.

3.2 The Recommendation Model

As stated before, the recommendation is based on the VSM model. The *query vector* is built with the terms parsed from: (i) the *cv:monograph-title* and *cv:monograph-keywords* of the user university degrees; and (ii) *cv:article-title* and *cv:article-keywords* of the bibliographic productions in *Lattes* CV (table 2). The parser ignores stop-words [5] (a list of common or general terms that are not used in the information retrieval process, e.g., prepositions, conjunctions and articles). The parser considers each term of the *cv:monograph-title* and *cv:article-title* as a single word. On the other hand, in both *cv: monograph-keywords* and *cv:article-keywords*, the terms are taken integrally, as single expressions.

The *query vector* terms weights are build up according to the Equation 2. This equation considers the type of the term (keyword or title), the language and the year of the publication (monograph or article). Keyword terms are considered more important than the titles and have higher weights assigned. Publications written in a language in which the user has more reading proficiency are more valorized (higher weight), and the terms obtained from the most recent university degree and productions are assigned a more important weight than the less recent ones.

$$w_t = w_{KeywordOrTitle} . w_{Language} . w_{Year} \quad \textbf{(2)}$$

The weights $W_{KeywordOrTitle}$, $W_{Language}$ and W_{Year} are calculated with Equation 3.

$$w_i = 1 - (i - 1) \left(\frac{1 - w_{\min}}{n - 1} \right) \quad \textbf{(3)}$$

In this equation, W_i varies according to the type of weight we want to compute. To illustrate, in the experimental evaluation (Section 4), for $W_{KeywordOrTitle}$, W_{min} was 0.95, and *i* is 1 for keywords and 2 for title terms. For $W_{Language}$, W_{min} was 0.60 and *i* is 1 if the language-skill-level is "good", 2 for "reasonable" and 3 for "few". For W_{Year}, W_{min} was 0.55 and *i* vary from 1 to *n*, where *n* is the interval of years considered, being 1 the highest and *n* the lowest. In the experimental evaluation it was considered

the interval between 2006 and 2003. However, if the interval is omitted, it will be considered as between the present year and the less recent year (the smallest between *cv:graduation-year* and *cv:article-year*).

If w_{min} is not informed, the default value will be used (presented in Equation 4). In this situation, Equation 3 is reduced to Equation 5.

$$w_{\min\, default} = \frac{1}{n} \quad \textbf{(4)}$$

$$w_i = \frac{n - i + 1}{n} \quad \textbf{(5)}$$

Once the *query vector* is build, the *documents vector* terms and the respective weights must be defined. The adopted approach was (*tf x idf*), i.e., the product of the term frequency and the inverse document frequency [19]. This approach allows automatic term weights assignment for the documents retrieval. The *term frequency* (*tf*) corresponds to the number of occurrences of a term in the document. The *inverse document frequency* (*idf*) is a factor that varies inversely with the number of the documents *n* to which a term is assigned in a collection of *N* documents (typically computed as log (N/n)).

The best terms for content identification are those able to distinguish individuals ones from the remainder of the collection [19]. Thus, the best terms correspond to the ones with high term frequencies (*tf*) and low overall collection frequencies (high *idf*). To compute *tf x idf*, the system uses the DC metadata *dc:title* and *dc:description* to represent the documents content. Moreover, as our system deals with different languages, the total number of documents will vary accordingly. After building the *query* and *documents* vectors, the system is able to compute the similarities values among the documents and the query according to Equation 1.

4 Experimental Evaluation

In order to evaluate the recommender system, we have asked for the *Lattes* CV from a group of individuals of our Institution entailed to different research teams of different Computer Science research areas, such as Information Systems and Theory of Computation. As response, a group of 14 people send us their *Lattes* CV, whose information were loaded in the User Profile local database. The Articles Metadata local database was loaded with metadata of all digital documents stored in BDBComp Digital Library up to June of 2006, totalizing 3,978 articles from 113 conferences editions.

After, 20 recommendations were generated by the system for each participant, considering individual's university degrees and bibliographic production information present in the Lattes CV. This information corresponded just to the last three years (i.e., 2003 to 2006). Each recommendation had the following attributes extracted: title (*dc:title*), authors (*dc:creator*), URL (*dc:identifier*), idiom (*dc:language*), publication year (*dc:date*), conference (*dc:source*) and abstract (*dc: description*).

Two evaluations were performed. The first was based on the hypothesis that the best articles to describe the profile of a researcher should be those produced by the researcher himself. Since we had information about the articles written by each author (from the curriculum), we can match the items recommended to those that were actually written by them. This evaluation was accomplished by the *recall* and *precision* metrics that is a standard evaluation strategy for information retrieval systems [20, 1]. The *recall* is used to measure the percentage of relevant documents retrieved in relation to the amount that should have been retrieved. In the case of document categorization, the *recall* metric is used to measure the percentage of documents that are correctly classified in relation to the number of documents that should be classified. *Precision* is used to measure the percentage of documents correctly recovered, i.e., the number of documents correctly retrieved divided by the number of documents retrieved.

As the profiles can be seen as classes and the articles as items to be classified in these profiles, we can verify the amount of items from the author that are correctly identified (i.e., classified) by the user profile. As we have many users (i.e., many classes), it is necessary to combine the results. The *macroaverage* presented in Equation 6 was designed by *D. Lewis* [14] to perform this specific combination ("*the unweighted mean of effectiveness across all categories*"), and was applied by him in the evaluation of classification algorithms and techniques.

$$macroaverage = \frac{\sum_{i=1}^{n} X_i}{n} \quad \textbf{(6)}$$

In this formula, X_i is the *recall* or the *precision*, depending on the metric we want to evaluate, of each individual class (user in our case) and *n* is the number of classes (users). Thus, the *macroaverage recall* is the arithmetic average of the recalls obtained for each individual, and the *macroaverage precision* is the arithmetic average of the *precisions* obtained for each individual.

Given that the users are not interested in its own articles as recommendations, we performed another evaluation that takes in to account only the items from others authors. Then, 15 recommendations were presented to each individual ranked on the relative grade of relevance generated by the system In this rank, the article with the highest grade of similarity with the user profile was set as 100% relevant and the others were adjusted to a value relative to it. In this case, each author was requested to evaluate the recommendations generated to them assigning one of the following concepts (following the bipolar five-point Likert scale): "Inadequate", "Bad", "Average", "Good", and "Excellent", and were also asked to comment the results. The following section presents the results obtained.

5 Analysis of the experiments

The first experiment was designed to evaluate the capability of the system to correctly identify the user profile (i.e., to represent its research interests), since we believe that the best articles to describe the user profile are those written by themselves, as stated before. To perform such evaluation, we identified the number of articles that each

author had at BDBComp. After that, we employed the *recall* metric to evaluate the number of articles recovered for each author and combined them with the *macroaverage* equation explained before.

We have found a macroaverage recall of 43.25%. It is important to state that each author received 20 recommendations. This is an acceptable value as the query construction was made automatically without human intervention. It happened to be lower than it should be if we have used more than the last three years of information stored in the *Lattes* CV. Thus, articles related to the previous research interest areas were not recommended as the objective of the system resumed on the recommendation of articles associated to recent research interest areas of the users. Other important consideration is that the recommendation ranking was generated with a depreciation degree that was dependent on the publication year and on the user language proficiency, as explained in the previous section. As the time-slice considered corresponds to a small part of the full conference period stored in the BDBComp, not all articles are good recommendations since the research profile changes along the time.

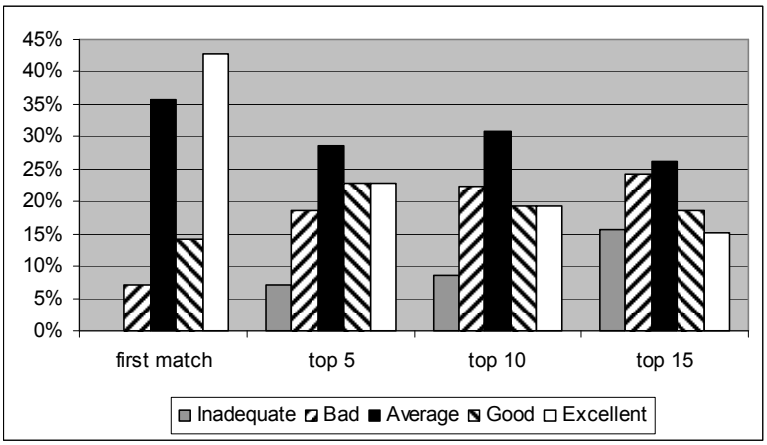

Fig. 2. Users' evaluations of the recommendations.

Figure 2 presents the results of the second experiment, which was based on the users' qualitative evaluation of the recommended articles. On this experiment each user received 15 recommendations and evaluated them according to one of the following concepts: "inadequate", "bad", "average", "good" and "excellent". The results were grouped into the categories "first match", "top 5", "top 10", and "top 15", and are presented in Figure 2.

Analyzing these results, it is possible to observe that, if we only consider the first article recommended (the "first match"), the number of items qualified as "excellent" is greater than the others (i.e., 42.86%) and none of them were classified as "inadequate". This strengthens the capability of the system on performing recommendations adjusted to the present user's research interests. We have also grouped the concepts "good" and "excellent" into a category named "positive

recommendation" and the concepts "bad" and "inadequate" into a "negative recommendation" group, so we could obtain a better visualization and comprehension of the results (Fig. 3).

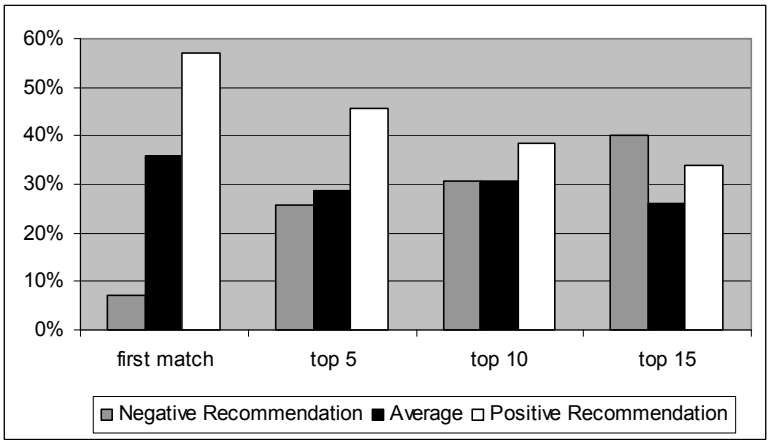

Fig. 3. Grouped users' evaluations.

We could perceive that the positive recommendations, considering only the "first match", are superior (57.14%) in relation to the negative ones (7.14%). The same behavior can be perceived in the "top 5" and "top 10" categories, the recommendations had a negative evaluation only in the "top 15" category, and that probably happened because as the number of recommendations grows, the number of correct recommendations falls. It is clear that the automated procedure here adopted is adequate for an alert recommender system. Our proposal is to add to the BDBComp an automated alert system that periodically sends to the user a list of the most relevant papers recently published in some of the nearly 35 Brazilian computer symposiums and 64 co-organized local events.

It is important to observe that today BDBComp has a limited coverage of the Computer Science area and it may have negatively influenced the quality of the recommendations. This was perceived in the commentaries made by some users, such as *"[...] I suppose that the generation of such results is a very complex task, as I worked with two distinct areas and mixed with even more themes. Besides, the two fields in which I have more publications have a very limited group of people working in this subjects here in Brazil. To conclude, considering such circumstances, the list of recommendations is good*.", and "*I can conclude that: (a) there are not many articles in my research area in BDBComp; or (b) I have not correctly described my articles metadata in Lattes CV*". In a near future all the SBC (Brazilian Computer Society) sponsored conferences will be automatically loaded [6].

Further, in our tests the authors that have changed their research area in the last three years have negatively qualified the recommendations. In the next experiments a variable time threshold and different depreciation values will be employed and the temporal component will be exhaustively analyzed.

6 Conclusion

This article presented a Recommender System to researchers and academics of the Computer Science area. In current days, in which the recovery of relevant digital information on the web is a complex task, such systems are of great value to minimize the problems associated to the information overload phenomena, minimizing the time spent to access the right information.

The main contribution of this research consists on the heavy utilization of automated CV data provider and in the use of a Digital Library (DL) metadata to create the recommendations. The system was evaluated with BDBComp, but it is designed to work with the open digital library protocol OAI-PMH, then it may be easily extended to work with any DL that supports this mechanism. The same occurs with the Curriculum Vitae, the system will be able to receive any XML-base CV data. Presently, the system uses the *Lattes* CV format, but it can be extended to support other formats or to analyze information about the user stored on tools like Scholar Google and DBLP. Alternatively the operational prototype offers the possibility to the user to load the CV data via an electronic form.

The developed system will have many applications. One of them is the recommendation of articles to support the learning process, especially on eLearning systems. Thus, the student could log into a specific distance or electronic learning environment supported by this system and receive recommendations of articles containing actualized relevant material to complement its current study topic.

References

1. Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, Wokingham, UK (1999)
2. BDBComp: Biblioteca Digital Brasileira de Computação, http://www.lbd.dcc.ufmg.br/bdbcomp/, Nov. (2006)
3. Callan, Jamie et al.: Personalisation and Recommender Systems in Digital Libraries. Joint NSF-EU DELOS Working Group Report. May (2003)
4. CITIDEL: Computing and Information Technology Interactive Digital Educational Library, http://www.citidel.org/, Nov. (2005)
5. CLEF and Multilingual information retrieval, http://www.unine.ch/info/clef/, Institut interfacultaire d'informatique, University of Neuchatel (2005)
6. Contessa, Diego Fraga; Oliveira, José Palazzo Moreira de: An OAI Data Provider for JEMS. Proceedings of the ACM DocEng 2006 Conference, Amsterdam. Oct. (2006) 218-220
7. DC-OAI: A XML schema for validating Unqualified Dublin Core metadata associated with the reserved oai_dc metadataPrefix, http://www.openarchives.org/OAI/2.0/oai_dc.xsd, Mar. (2005)
8. Dublin Core Metadata Initiative, http://dublincore.org, Sept. (2005)
9. Grossman, David A.: Information retrieval: algorithms and heuristics. 2nd ed. Dordrecht: Springer, 332p. (2004)
10. Gutteridge, C.: GNU EPrints 2 overview, Jan. 01 (2002)
11. Huang, Z. et. al.: A Graph-based Recommender System for Digital Library. In: JCDL'02. Portland, Oregon (2002)

12. Laender, A. H. F.; Gonçalves, M. A.; Roberto, P. A.: BDBComp: Building a Digital Library for the Brazilian Computer Science Community. In: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries, Tucson, AZ, USA (2004) 23-24
13. Lattes-CNPq: Plataforma Lattes - Conselho Nacional de Desenvolvimento Científico e Tecnológico, http://lattes.cnpq.br/, Mar. (2005)
14. Lewis, D. D. : Evaluating text categorization. In Proceedings of Speech and Natural Language Workshop. Defense Advanced Research Projects Agency, Morgan Kaufmann. (1991) 312-318.
15. LPML-CNPq. Padronização XML: Curriculum Vitae, http://lmpl.cnpq.br/lmpl/?go=cv.jsp, Mar. (2005)
16. Maly, K.; Nelson, M.; Zubair, M.; Amrou, A. ; Kothamasa, S.; Wang, L.; Luce, R.: Light-weight communal digital libraries. In Proceedings of JCDL'04, Tucson, AZ (2004) 237-238
17. OAI: Open Archives Initiative, http://openarchives.org, Oct. (2005)
18. OAI-PMH: The Open Archives Initiative Protocol for Metadata Harvesting, http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm, Nov. (2005)
19. Salton, Gerard; Buckley, Christopher.: Term-Weighting Approaches in Automatic Text Retrieval, Information Processing and Management: an International Journal, v.24, Issue 5, 513-523. (1988)
20. Salton, Gerard; Macgill, Michael J.: Introduction to Modern Information Retrieval. New York: McGRAW-Hill. 448p. (1983)
21. Sompel, H. V. de; Lagoze, C.: The Santa Fe Convention of the Open Archives Initiative. D-Lib Magazine, [S.l.], v.6, n.2, Feb. (2000)
22. Tansley, R.; Bass, M.; Stuve, D.; Branschofsky, M.; Chudnov, D.; McClellan, G.; Smith, M.: DSpace: An institutional digital repository system. In Proceedings of JCDL'03, Houston, TX. (2003) 87-97

An OWL-Based Approach Towards Representing Time in Web Information Systems

Viorel Milea[1], Flavius Frasincar[1], Uzay Kaymak[1], and Tommaso di Noia[2]

[1] Erasmus University Rotterdam, Netherlands
Burgemeester Oudlaan 50, 3062 PA Rotterdam
{milea, frasincar, kaymak}@few.eur.nl
[2] Politecnico Di Bari, Italy
via E. Orabona, 4 - 70125 Bari
t.dinoia@poliba.it

Abstract. In this paper we present an approach towards representing dynamic domains by means of concrete domains and perdurants. This approach is based on Description Logic and enables the representation of time and time-related aspects such as change in ontologies. The approach taken in this paper focuses on two main goals that need to be achieved when talking about time in ontologies: representing *time* itself and representing *temporal aspects*. We employ an explicit representation of time and rely on the internal method for the purpose of reflecting the changing aspects of individuals over time. We also present a proof of concept for the developed approach. This consists of a system that extracts the relevant information regarding company shares from analyst recommendations and uses this aggregate information to generate buy/hold/sell signals based on predefined rules.

1 Introduction

One of the challenges posed by the Semantic Web is dealing with temporal aspects in a variety of domains, such as knowledge reasoning. While the Web Ontology Language (OWL) is the preferred alternative for representing domain knowledge, currently the language offers only little support for representing temporal information in ontologies. In this paper we present an approach for representing dynamic domains by means of concrete domains and perdurants. This approach is based on Description Logics (DL) and enables the representation of time and time-related aspects such as change in ontologies.

The approach we take in this paper focuses on two main goals that need to be achieved when talking about time in ontologies: representing *time* itself (in the form of dates, times, etc.) and representing *temporal aspects* (changing individuals, temporal knowledge, etc.). The representation of time is rather straight forward, and relates to making the latter available in the ontology in the form of dates, hours, minutes, etc. This type of representation allows for more (semantically) useful time representations such as instants and intervals. We talk here about an *explicit* representation of time, as this representation allows the usage

of temporal operators and combining the latter for obtaining new expressions [1]. Representing time in such a manner allows the use of the 13 Allen relations [2] in combination with temporal intervals. The symbiosis between the time intervals and Allen's relations represents the concrete domain employed for the current purpose. Representing time is an essential feature of a language that seeks to represent dynamic domains. However, this representation must be supported by means of consistently expressing temporal aspects, such as change, in ontologies. Two approaches are possible for this purpose, namely the internal and the external method [1]. In this paper we employ the *internal method* for the purpose of reflecting the changing aspects of individuals over time. This method relates to representing entities (perdurants) in a unique manner at different points in time. The actual individual is then nothing more than the sum of its (temporal) parts. Following the approach taken in [3], we implement this representation by making use of time slices (the temporal parts of an individual) and fluents (properties that hold between timeslices).

The remainder of this paper is organized as follows. In section 2 we present a detailed description of the TOWL language. An extended example of the possible use(s) of the language is presented in section 3. Section 4 provides some concluding remarks and suggestions for further research.

2 The TOWL Language

The focus of this section is on introducing the concepts necessary for describing time and change in ontologies. The resulting ontology language, TOWL, is a symbiosis of the $\mathcal{SHOIN}(\mathcal{D})$ description logic and its extension. This extension consists of a concrete domain that represents time and the perdurantist approach towards modeling the changing aspects of entities through time. The first part of this section consists of a more general overview of the TOWL language, while in the second part the afore mentioned language is formally introduced by means of describing its syntax and semantics.

2.1 Introducing TOWL

The TOWL ontology language is intended to be an extension of the current Web Ontology Language (OWL), and thus an extension of the $\mathcal{SHOIN}(\mathcal{D})$ description logic. The thus obtained language allows the representation of knowledge beyond the constraints of a static world, enabling the representation of dynamic entities that change (some) traits through time by expressing them as perdurants. A number of powerful time relations/operators, as described by Allen [4], are available for the purpose of reasoning with time.

The Concrete Domain: Intervals and Allen's Relations
A concrete representation of time is available in TOWL through *time intervals*, modeled as a concrete domain. A time interval is defined as a pair of *time points* and is available in TOWL through the *towl:TimeInterval* class. The ends

of an interval, represented as time points, are available in TOWL through the *towl:TimePoint* class. Following the reasoning in [5], time points are modeled as reals, and a time interval I_t can thus formally be defined as a pair (t_1, t_2) with $t_1, t_2 \in \mathbb{R}$. A proper interval is then defined as a pair (t_1, t_2) where $t_1 < t_2$. Intervals are related by Allen's thirteen relations: equals, before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by. These relations are *exhaustive* and *mutually exclusive* [5], i.e. for each pair of intervals there exists at least one relation holding true between them and, respectively, for each pair of intervals there exists at most one relation that holds true amongst them. It should be noted that all of Allen's 13 relations can be expressed in terms of the endpoints of intervals and the set of predicates $\{<, =\}$, the only two predicates of the concrete domain, that apply to all reals.

Perdurants in TOWL Ontologies

The concrete domain approach for representing time provides a good foundation towards representing change in ontologies. For this purpose the following TOWL concepts are introduced following the reasoning in [3]: *towl:TimeSlice, towl:tsTimeSliceOf, towl:fluentObjectProperty, towl:fluentDatatypeProperty* and *towl:tsTime*. The temporal parts of a perdurant are described as timeslices, and each of these timeslices is an individual of type *towl:TimeSlice*. They can be regarded as snapshots (slides) of an individual at a particular moment (interval) in time. The period of time for which each individual timeslice holds true is described as a pair (t_1, t_2) from the concrete domain and is associated to the timeslice through the *towl:tsTime* property. In case the temporal information does not regard an interval, but a single time point, then this time point can also be associated to the timeslice through the *towl:tsTime* property. The individual that is described by each particular timeslice over a time interval I_t, i.e. the perdurant, is indicated by means of the *towl:tsTimeSliceOf* property. Finally, timeslices are connected through (subproperties of) the *towl:fluentObjectProperty* relation while the association between timeslices and literals is indicated by the *towl:fluentDatatypeProperty*.

Advantages of this Approach

The advantages of the approach presented here are twofold. First, unlike previous approaches [3, 6–8], the current approach provides the means to represent both time in its quantitative nature, as well as temporal entities. Each of the previous approaches mentioned here focusses on only one of these aspects, such as [7] where the main focus is on representing time in its quantitative meaning by employing concrete domains. Other approaches that try to tackle the same problem, such as [8] where time is made available through an ontology of time, offer little to no support for automated temporal reasoning, thus bringing the discussion to the second advantage of the representation we chose for the purpose of representing time. Since all concepts are modeled at language level, this provides the basis for designing appropriate algorithms that will enable temporal reasoning with regard to both meanings of time as underlined here: quantita-

tive time (order, duration, etc.) as well as temporal entities (change, temporally bounded existence, evolution, etc.).

2.2 TOWL: Syntax and Semantics

In this subsection we formally introduce the syntax and semantics of the TOWL language. This presentation only includes the additional syntax and semantics of TOWL. Figure 1 gives an overview hereof. A further specification of the TOWL concepts is given in Figure 2, where the extensional semantics of the newly defined language is presented. Finally, the OWL schema of TOWL is presented in Figure 3, in OWL abstract syntax.

$C, D \longrightarrow$	$TS \mid$	(towl:TimeSlice)
	$TE \mid$	(towl:TimeEntity)
	$I_t \mid$	(towl:TimeInterval)
	$P_t \mid$	(towl:TimePoint)
	$\ell \mid$	(rdfs:Literal)
	$TS \sqsubseteq \forall FOP.TS \mid$	(towl:fluentObjectProperty)
	$TS \sqsubseteq \forall FDP.\ell \mid$	(towl:fluentDatatypeProperty)
	$TS \sqsubseteq \forall TSO_{ts}.C \mid$	(towl:tsTimeSliceOf)
	$TS \sqsubseteq \forall T_{ts}.(I_t \sqcup P_t) \mid$	(towl:tsTime)
	$I_t \sqsubseteq \forall S_{ti}.P_t \mid$	(towl:tiStart)
	$I_t \sqsubseteq \forall E_{ti}.P_t \mid$	(towl:tiEnd)
	$P_t \sqsubseteq \forall D_{tp}.\ell \mid$	(towl:tpDate)

Fig. 1. TOWL syntax rules

functional properties :	$TSO_{ts}, T_{ts}, S_{ti}, E_{ti}, D_{tp}$.
$(TS)^{\mathcal{I}}$	$= \{a \in \Delta^{\mathcal{I}} \mid TSO_{ts}(a) \in \Delta^{\mathcal{I}}\}$
$(I_t)^{\mathcal{I}}$	$= \{a \in \Delta^{\mathcal{I}} \mid S_{ti}(a) = x_1,\ E_{ti}(a) = x_2 \text{ and } D_{tp}(x_1) < D_{tp}(x_2)\}$
$(P_t)^{\mathcal{I}}$	$= \{a \in \Delta^{\mathcal{I}} \mid D_{tp}(a) \in \ell\}$
$(\forall T_{ts}.I_t)^{\mathcal{I}}$	$= \{a \in (TS)^{\mathcal{I}} \mid \forall b \in \Delta^{\mathcal{I}} : (a,b) \in (T_{ts})^{\mathcal{I}} \rightarrow b \in (I_t)^{\mathcal{I}}\}$
$(\forall T_{ts}.P_t)^{\mathcal{I}}$	$= \{a \in (TS)^{\mathcal{I}} \mid \forall b \in \Delta^{\mathcal{I}} : (a,b) \in (T_{ts})^{\mathcal{I}} \rightarrow b \in (P_t)^{\mathcal{I}}\}$
$(\forall FOP.TS)^{\mathcal{I}}$	$= \{a \in (TS)^{\mathcal{I}} \mid \forall b \in ((TS)^{\mathcal{I}} \setminus \{a\}), \exists t_1, t_2 \in (I_t)^{\mathcal{I}} :$ $(a,b) \in (FP)^{\mathcal{I}}, a \in TI.t_1, b \in TI.t_2 \rightarrow t_1 = t_2\}$
$(\forall FDP.\ell)^{\mathcal{I}}$	$= \{a \in (TS)^{\mathcal{I}} \mid \forall (a,b) \in (FDP)^{\mathcal{I}} \rightarrow b \in \ell\}$
$(\forall TSO_{ts}.C)^{\mathcal{I}}$	$= \{a \in (TS)^{\mathcal{I}} \mid \forall b \in \Delta^{\mathcal{I}} : (a,b) \in (TSO_{ts})^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$

Fig. 2. TOWL semantics

When compared with the original fluents approach [3] that stands at the basis of the approach we present here, a number of additional features have been

incorporated in the language, thus adding to its flexibility and expressiveness. One of these features is allowing the association of time slices not only with intervals, but also with time points, a representation essential for systems as the one described in this paper. Additionally, we make a distinction between two different types of fluent properties: datatype fluent properties, that point to objects of type rdfs:Literal, and object fluent properties that point to objects, i.e. actual timeslices of individuals present in the ontology. The case of the datatype fluent property is special in that it does not require a timeslice of a specific type of literal, but it may point to the actual value itself.

3 An Extended Example

In this section we present an example of how the TOWL language can provide for added value. For this purpose we sketch a system that uses last minute news for the generation of buy/hold/sell signals based on market consensus. The system consists of five parts, reflecting the essential components of the system: 1) the financial TOWL ontology - the ontology used to store all knowledge relevant to the system, 2) information extraction, the component that extracts the relevant knowledge from news messages, 3) ontology update - the actual updating of the ontology with the new information, 4) query evaluation - an important part of the process of answering queries regarding the current state of the world as described in the financial TOWL ontology and, finally, 5) the actual application that generates buy/hold/sell signals on an on-demand basis derived from the domain knowledge modeled in the ontology.

3.1 The Financial TOWL Ontology

For the purpose of this example a simple TOWL financial ontology has been developed. The schema of this ontology consists of a *Company* class, the class of all companies that are of relevance in the ontology, a *CompanyAdvice* class that denotes the advices issued by experts regarding companies, and a class *AdviceType* defined by means of its only three instances *buy*, *hold* and *sell* - the actual recommendation(s) of the expert for some company. Additionally, a number of properties have been defined that further specify the meaning of classes. The property *hasName* indicates the actual name of the individuals of type *Company*, the property *adviceType* relates all individuals of type *CompanyAdvice* to individuals of type *AdviceType*, while the property *priceTarget12* indicates, for all individuals of type *CompanyAdvice*, the expected price over 12 months, as formulated in a particular advice. Finally, two fluent properties 'connect' timeslices of individuals of type *Company* to corresponding timeslices of individuals of type *CompanyAdvice*. The *adviceIssuedBy* property indicates the company that has issued a particular advice and the *adviceIssuedFor* property indicates the company for which the particular advice has been issued. A formal representation of this ontology is given in Figure 4, in OWL abstract syntax.

```
Ontology(TOWL
  Class(TimeSlice)
  Class(TemporalEntity)
  Class(TimeInterval partial TemporalEntity)
  Class(TimePoint partial TemporalEntity
   restriction(complementOf(TimeInterval)))
  DisjointClasses(TimeSlice TemporalEntity)

  ObjectProperty(fluentObjectProperty Symmetric)
   domain(TimeSlice)
   range(TimeSlice))

  DatatypeProperty(fluentDatatypeProperty Symmetric)
   domain(TimeSlice)
   range(rdfs:Datatype))

  ObjectProperty(tsTimeSliceOf Functional
   domain(TimeSlice)
   range(complementOf(unionOf(TimeSlice TemporalEntity rdfs:Literal))))

  ObjectProperty(tsTime Functional
   domain(TimeSlice)
   range(TemporalEntity))

  DatatypeProperty(tiStart Functional
   domain(TimeInterval)
   range(TimePoint))

  DatatypeProperty(tiEnd Functional
   domain(TimeInterval)
   range(TimePoint))

  DatatypeProperty(tpDate Functional
   domain(TimePoint)
   range(xsd:dateTime)))
```

Fig. 3. OWL Schema of TOWL

3.2 Information Extraction

The information extraction phase is responsible for providing the system with the necessary input in the form of processed knowledge from news messages. For this example we focus on a particular type of news - analyst recommendations - in the form of buy/hold/sell signals, sometimes accompanied by a price target. In this example we use the following three news messages.

```
Ontology(finTOWL
  Class(Company)
  Class(CompanyAdvice)
  EnumeratedClass(AdviceType buy hold sell)
  DisjointClasses(Company CompanyAdvice AdviceType)

  DatatypeProperty(hasName
   domain(Company)
   range(xsd:String))

 ObjectProperty(adviceType Functional
   domain(CompanyAdvice)
   range(AdviceType))

 ObjectProperty(adviceIssuedBy super(fluentObjectProperty) Functional
   domain(restriction(tsTimeSliceOf(allValuesFrom Advice)))
   range(restriction(tsTimeSliceOf(allValuesFrom Company))))

  ObjectProperty(adviceIssuedFor super(fluentObjectProperty) Functional
   domain(restriction(tsTimeSliceOf(allValuesFrom Advice)))
   range(restriction(tsTimeSliceOf(allValuesFrom Company))))

  DatatypeProperty(priceTarget12 Functional
   domain(CompanyAdvice)
   range(xsd:double)))
```

Fig. 4. The Financial TOWL Ontology

News1

(MarketAdvices.com) New York(7-17-2006) - Mark Hebeka of Standard & Poors reiterates his buy recommendation for the American bank and insurance company Citigroup. The 12-months target price for Citigroup is reiterated at 55 USD.

News2

(MarketAdvices.com) New York (1-19-2007) - The analysts of Goldman Sachs reiterate their hold recommendation for the American bank and insurance company Citigroup (ISIN: US1729671016 / Symbol: C). The 12-months target price for Citigroup is 59 USD.

News3

(MarketAdvices.com) New York (1-29-2007) - Analyst Frank Braden of Standard & Poors reiterates his buy recommendation for the American bank and insurance company Citigroup (ISIN:US1729671016 / Symbol: C). A price target was not provided.

For all of the three news messages, a feature selection has been performed, and the selected features have been highlighted in the examples above. An analysis of the *News1* example provides the following: *17/7/2006*, the date when the advice was issued and thus the date starting at which the advice holds true, *Standard and Poor's*, the company that has issued the advice, *Citigroup*, the company for which the advice was issued, *buy*, the advice type, and *55 USD*, the value of the *12-months target price* for one Citigroup share according to the expectation of the analysts at Standard and Poor's. This process is repeated for each of the news messages.

3.3 Knowledge Base Update

Having performed the extraction phase, the resulting knowledge relevant to the domain is modeled explicitly in the knowledge base (KB). The way in which this can be achieved is presented below, in OWL abstract syntax, for each of the news messages previously processed. One assumption is that static knowledge regarding the three companies involved is already present in the ontology, and modeled as presented in Figure 5.

```
Ontology(finTOWL
  Individual(iCitigroup
   type(Company)
   value(name "Citigroup"^^xsd:String))

  Individual(iStandardPoors
   type(Company)
   value(name "Standard and Poor's"^^xsd:String))

  Individual(iGoldmanSachs
   type(Company)
   value(name "Goldman Sachs"^^xsd:String))
```

Fig. 5. Static individuals of finTOWL

News1

For the purpose of representing the information contained in the first news message, a number of timeslices have to be created of the individuals StandardPoors, Citigroup and CompanyAdvice. This results in three timeslices, one for each of the aforementioned individuals: *iSandP_TS1*, *iCitigroup_TS1* and finally *iCitiAdvice1_TS1* for the CitiAdvice1 individual. The beginning of the period in which the advice holds true is modeled as an individual of type TimePoint that contains the relevant xsd:dateTime object: *iTP1*. Finally, the timeslice of the

advising company, iStandardPoors_TS1 is associated with the timeslice of the issued advice, iCitiAdvice_TS1, through the adviceIssuedBy property. Similarly, the timeslice of the company that received the advice, iCiti_TS1 is associated with the timeslice of the received advice - iCitiAdvice_TS1, through the adviceIssuedFor property. It should be noted that at the moment this knowledge became available, no additional information is available on the duration of this advice, hence only the starting moment of this advice has been modeled as an individual of type TimePoint. This representation is summarized in Figure 6, where a model of the *News1* news message is given in OWL abstract syntax. As soon as a new advice is issued by Standard and Poor's for the company Citigroup, the duration of the new advice will be known and will equal the time between the already known starting point (iTP1) and the time point at which the new advice has been issued. Thus, the tsTime property will not have an argument of type TimePoint, but of type TimeInterval as soon as this information becomes available. This is the case after the issuing of a new advice by Standard and Poor's for Citigroup, as in the *News3* news message. The concrete changes in the KB, that ideally are automatically performed, are illustrated in Figure 7.

3.4 Query Evaluation

The most basic operations that the TOWL language enables become evident through the queries that may be posed upon the system. These queries form an essential part of the rules used to determine the final output (the buy/hold/sell signals). In this section we present some examples of queries and how the results of these queries can be inferred by the system.

A number of four query examples are presented below, where the most relevant part referring to time or some aspect of time is in bold:

IEX_1 Was any advice for Citigroup issued **after** Goldman Sachs issued an advice for Citigroup on January 19th, 2007?

IEX_2 Was any advice for Citigroup issued **while** the Goldman Sachs advice for Citigroup issued on January 19th, 2007 was holding?

IEX_3 When were **the last two** buy advices issued for Citigroup?

IEX_4 Was there any positive (buy) advice for Citigroup **in** January 2007?

The first query example, IEX_1, relates to comparing individuals of type TimePoint, whether they individuals are present individually or as part of a TimeInterval object, i.e. as argument of the tiStart or tiEnd of an individual of type TimeInterval. First, the specific advice issued by Goldman Sachs on January 19th 2007 should be identified as the individual iCitiAdvice2_TS1 of type TimeSlice. Next, the moment in time when this advice was issued must be retrieved; this moment in time is the argument of the tpDate property of

iCitiAdvice2_TS1 in case this property is defined, or the argument of the tiStart property in case this property is defined. Next, all advices that have been issued after the date of January 19th, 2007 must be retrieved. These advices are individuals of type TimeSlice for which the property tsTimeSliceOf has an argument of type CompanyAdvice. The property adviceIssuedFor of the previously selected individuals must point to a timeslice of Citigroup. Additionally, one of the properties tpDate (in the case of a TimePoint) or tiStart (in the case of a TimeInterval) must be defined, and the argument of this property should be strictly larger than January 19th, 2007. If the set of individuals satisfying all these constraints is not empty, then the answer to this query is positive.

A similar procedure can be applied for answering the remaining query examples (2 through 4) by trying to find a set of individuals that satisfies the constraints specified in the query. A special case is the query example *IEX_3*, where the answer to this query is not of Boolean type, but consists of a set of date objects. In order to answer this query, the set of buy advices for Citigroup must be selected and ordered according to the date these advices were issued. Then, the arguments of the tpDate properties or of the tiStart properties should be returned.

3.5 Application

At application level, the basic query examples described in the previous section can be combined for the purpose of generating buy/hold/sell signals based on predefined rules. A simple example of such a rule is given below.

APR_1 "If a buy advice is issued for a company while another buy advice holds, then buy".

A possible way of firing this rule reduces to checking, each time a new buy advice is issued for some company X, whether another buy advice holds true for X. If this is the case, then generate a buy signal. Of course, this rule does not say anything about the situation in which there are already two buy advices still holding true for company X when a third buy advice is issued, but in this case we just assume that the already generated (buy) signal is not changed, or perhaps it is reiterated.

The trading signal generation system based on TOWL can also be used to generate buy/hold/sell signals based on more complex rules, such as in rule *APR_2*. Here we assume that advices are of the form -1, 0, 1 for sell, hold and buy, respectively, and the final result returned by $SYST_{ADV}$ can be rounded to the nearest whole number. The m, n and p variables represent prespecified weights specific to each company that issued an advice.

APR_2 "Companies X, Y, Z issued advices A, B, C for company W. For each point in time the advice issued by the system, is a weighted average of the three

```
Individual(iStandardPoors_TS1
 type(TimeSlice)
 value(tsTimeSliceOf StandardPoors)
 value(tsTime iTP1))

Individual(iCitigroup_TS1
 type(TimeSlice)
 value(tsTimeSliceOf iCitigroup)
 value(tsTime iTP1))

Individual(iTP1
 type(TimePoint)
value(tpDate "17/7/2006"^^xsd:date))

Individual(iCitiAdvice1
 type(CompanyAdvice)
 value(adviceType buy)
 value(priceTarget12 "55"^^xsd:double))

Individual(iCitiAdvice1_TS1
 type(TimeSlice)
 value(tsTimeSliceOf iCitiAdvice1)
 value(tsTime iTP1)
 value(adviceIssuedBy iStandardPoors_TS1)
 value(adviceIssuedFor iCitigroup_TS1))
```

Fig. 6. A TOWL representation of *News1*

```
Individual(iTI1
 type(TimeInterval)
 value(tiStart iTP1)
 value(tiEnd iTP3))

Individual(iStandardPoors_TS1
 value(tsTime iTI1))

Individual(iCitigroup_TS1
 value(tsTime iTI1))

Inivididual(iCitiAdvice1_TS1
 value(tsTime iTI1))
```

Fig. 7. Ontology update after *News3* has been issued

individual advices: $SYST_{ADV} = (mA + nB + pC)/(m + n + p)$ if all three companies have an advice for W at time T".

4 Conclusions and Further Research

This paper presents a new ontology language that allows the expression of time and change in ontologies: the TOWL language. Two aspects of time are deemed essential: the actual/concrete time and the concept of change. The TOWL language offers the possibility of representing both these aspects in ontologies and offers a consistent way of expressing the changing aspect of the entities in some world by means of perdurants. Although the concept of concrete domains or fluents is not new, the symbiosis between the two is unique in representing time and change in KR languages. Moreover, the original approach described in [3] has been extended towards added expressivity and increased flexibility, while the perdurants syntax has a basic underlying semantics. There are however a number of issues requiring attention in further research, such as cardinality restrictions on fluents with regard to overlapping timeslices.

Acknowledgement

The authors are supported by the EU funded IST STRP Project FP6 - 26896: Time-determined ontology-based information system for realtime stock market analysis. More information is available on the official website[3] of the project.

References

[1] Artale, A., Franconi, E.: A survey of temporal extensions of description logics. Annals of Mathematics and Artificial Intelligence **30** (2000) 171–210
[2] Allen, J.F., Ferguson, G.: Actions and events in interval temporal logic. Journal of Logic Computation **4** (1994) 531–579
[3] Welty, C., Fikes, R., Makarios, S.: A reusable ontology for fluents in owl. In: Proceedings of FOIS, IOS Press (2006) 226–236
[4] Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26** (1983) 832–843
[5] Lutz, C.: The Complexity of Description Logics with Concrete Domains. PhD thesis, RWTH Aachen (2002)
[6] Artale, A., Franconi, E.: Introducing temporal description logics. In: TIME '99: Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning, Washington, DC, USA, IEEE Computer Society (1999) 2
[7] Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz (1991)
[8] Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Transactions on Asian Language Information Processing (TALIP) **3** (2004) 66–85

[3] http://www.towl.org

Optimising XML-Based Web Information Systems

Colm Noonan and Mark Roantree

Interoperable Systems Group, Dublin City University, Ireland - {mark,cnoonan}@computing.dcu.ie

Abstract. Many Web Information Systems incorporate data and activities from multiple organisations, often at different geographical (and cultural) locations. Many of the solutions proposed for the necessary integration in Web Information Systems involve XML as it provides an interoperable layer between different information systems. The impact of this approach is the construction of large XML stores and often there is a need to query XML databases. The work presented in this paper supports the web computing environment by ensuring that this canonical representation (XML) of data can be efficiently processed regardless of potentially complex structures. Specifically, we provide a metamodel to support query optimisation for Web Systems that employ XPath.

Key words: XML Query Optimisation, XPath, Web Information Systems

1 Introduction

As today's Web Information Systems (WIS) often incorporate data and processes from multiple organisations, the concept of *data everywhere* is more evident. Furthermore, this distribution and diversity of information will increase as we move towards ambient, pervasive and ubiquitous WIS computing. There has been over 20 years of research into interoperability for information systems and it is likely that many of the solutions proposed for the necessary integration in the WIS environment will involve XML as it provides an interoperable layer between different information systems. The effect of this approach is the construction of large XML stores (referred to as XML databases from now on), together a need to efficiently query these XML databases. In fact, it is often suitable to retain this information in XML format due to its interoperable characteristics or because source organisations have chosen XML as their own storage method. Our motivation is to support the WIS computing environment by ensuring that the canonical representation (XML) of data has the sufficient query performance. Our approach is to provide an XML metamodel to manage the metadata necessary and support query optimisation for WIS applications.

In the WIS environment, the canonical model must be semantically rich enough to represent all forms of data structure and content. However, XML databases perform badly for many complex queries where the database size is

large or the structure complex and thus, some form of indexing is required to boost performance. The index-based approach provides an efficient evaluation of XPath queries against target XML data sets. Given an XPath query, a query processor must locate the result set that satisfies the content and structural conditions specified by the query. Suitable indexing structures and query processing strategies can significantly improve the performance of this matching operation. There are numerous index-based query processing strategies for XML databases [2, 3, 9, 15, 16], although many fail to support the thirteen XPath axes as they concentrate on the `child`, `descendant` and `descendant-or-self` axes [15, 16].

1.1 Contribution and Structure

In this paper, we present a metamodel for XML databases that models three levels of abstraction: the lowest layer contains the index data; the metamodel layer describes the database (tree) and the meta-metamodel layer describes the schema (tree). However, for this paper we do not describe the third layer. This provides for an advanced schema repository (or system catalog in the relational sense) that is extended to hold index information for query optimisation. We also demonstrate our query processing strategy by discussing methods for evaluation of XPath axes and provide details of experiments to assess the merits of this approach.

The paper is structured as follows: in §2, we provide the concepts and terminology used in our metamodel; in §3, we provide details of the metamodel, and the extended metamodel containing index data; the algorithms that use the metamodel to assist the query optimisation process are described in §4; in §5, we describe the results of our experiments while in §6 we discuss similar approaches; and finally in §7, we offer conclusions.

2 XML Tree Fundamentals

For this work, we examined XML trees and schemas and sought to formalise the relationships between them. An attempt at providing some form of structure and statistics for semi-structured sources was proposed in [6] where the authors defined a concept called DataGuides (described later in this paper). We have created a more extensive metadata description of both data and schema trees and differ in the set of allowable paths. In this section, we define our terminology and specify the relationships between metadata and data structures.

2.1 Terminology

We refer to the XML data tree as the *database*; the schema tree as the *schema*; and later in the paper, we describe a higher level of abstraction: the *meta-schema*. In figure 1(a), two schemas are illustrated (actual sub-trees from the

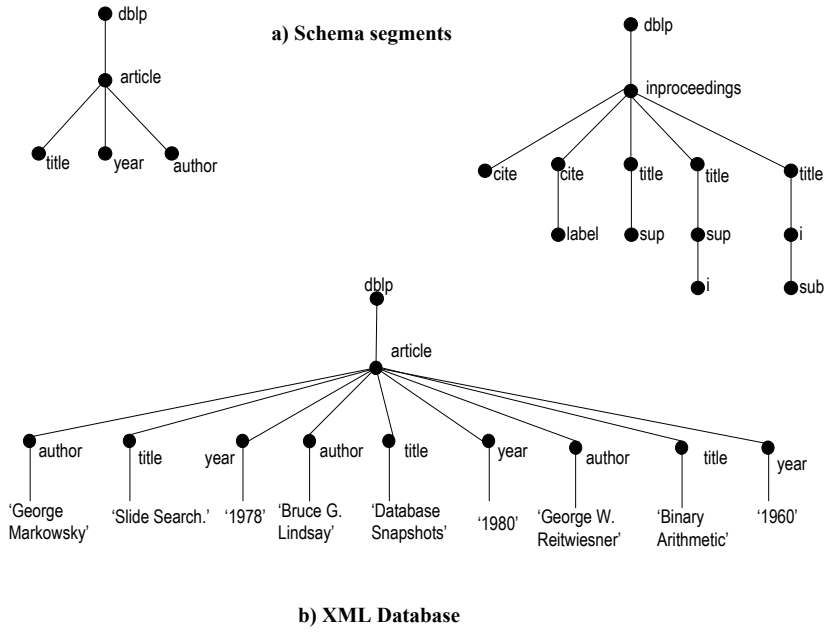

Fig. 1. Mapping Across Schema and Data Trees

dblp schema) and in figure 1(b), a database is illustrated for the first of these schemas. We begin with a description of the *schema*.

XML *schemas* are tree structures containing paths and nodes, with the *root node* being the entry point to the tree. A tree-path that begins at the root node, continuing to some *context* node is called a *FullPath*. Tree-paths need not be unique in XML trees[1] but in our metamodel, each FullPath instance is unique. In figure 1(a), the node `title` has the FullPath `//dblp/article/title`. Unlike [1, 6], we do not recognise sub-paths ie. those paths not connected to the document root.

Property 1 *A Schema S contains a set of FullPaths* $\{P\}$*, with one or more instances in the database.*

The schema is divided into levels with the root at the topmost level (level 0). Each node has 0 or more child nodes, with child nodes also having 0 or more children (at one level greater than the parent). Rather than use the term sub-tree, we chose the term *Family* to refer to all child and ancestor nodes for a given context node. Thus, the *Family* for the root is the entire tree, with all other nodes having smaller families.

[1] There are multiple path instances in the dblp schema.

Property 2 *Each Node N at level(x) is the Head of a Family F of connected nodes at levels (x+1),(x+2),...(x+n).*

In 1(a), the `article` Family is small (3 children) while the `inproceedings` Family is larger with child and descendant nodes.

Property 3 *A Family F is a set of FullPaths with a common ancestor node.*

The term *Twig* is used to refer to a Family with some members missing (a pruned Family). This is useful where a Family is very large and one wishes to reduce the size of the Family sub-tree (perhaps for querying performance reasons). The term Twig has been used in the past (eg. [4])to refer to data sub-trees returned as a result of queries and our usage of the term is not dissimilar here.

Property 4 *A Twig T is a subset of some Family F.*

XML databases are instances of XML schemas. Specifically, they have one or more instances of the FullPath construct, and by extension contain Families and Twigs. For example, the **article** Family in figure 1(a) has three instances in figure 1(b).

Similar to many other projects that focus on XPath and XQuery optimisation, we developed an indexing system to both prune the tree space and perform the subsequent query processing. Unlike these approaches, we adopt a more traditional database approach of specifying an Extended Schema Repository (ESR) for the XML database and provide a common interface to the repository. The metadata features of the schema repository (i.e. `FullPath` and `Level` structures) are the principle constructs for query optimisation. These structures describe the database schema, together with some statistics for the data in the tree. In the Extended section of the ESR, we store physical data taken from the XML tree and used to form indexes to boost query performance.

3 The XML Metamodel

In this section we provide an overview of the two main layers of the ESR: index data and metadata.

3.1 XML Index Data

When the XML database is parsed, all nodes are given {preorder,level} pairs to uniquely identify them. Please refer to our earlier work on indexing XML databases [7] for a detailed discussion on the indexing scheme and to [11] for a description of the object-based interface to index data. This segment of the ESR contains the base and level indexes. To improve efficiently in evaluating XPath location steps along the thirteen XPath axes, a Level index is employed.

– **The BASE Index**
– The `PreOrder` and `Level` values, together with node `parent` are recorded.
– `Type` is used to distinguish between elements and attributes.
– The `name` and `value` of the node.
– `FullPath` is the entire path to the leaf node (direct relationship to FullPath structure.
– The DocID in the event that the database contains multiple documents.
– `Position` This refers to the position of this data item across the *level* and is determined by counting all nodes at this level with smaller preorder value.
– **The LEVEL Index**
– The `Level` value.
– An ordered sequence of `PreOrder` values occurring at a given level in the data tree.

3.2 XML Metadata

The purpose of the XML Metadata layer is to describe, through the FullPath and Level meta-objects, the structure and content of the XML database. As this is the main construct for query optimisation, its content was heavily influenced by algorithms for the 13 XPath axes [13]. In addition, every FullPath entry has a relationship with a Level object, providing information about the size of the level in the data tree.

– **The FullPath Structure**
– `FullPath`.
– `NodeCount` is the number of instances of this FullPath.
– `DocID` is necessary where the database contains multiple documents.
– `name` is the name of the leaf node (part of FullPath).
– `Type` attribute is one of Element, Attribute or Root.
– `Level` records the depth of this node.
– `PreorderStart` is the first preorder value for this node.
– `PreorderEnd` is the last preorder value for this node.
– `LeftPos` is smallest PreOrder value at this level for this FullPath (only instances of this FullPath).
– `RightPos` is the biggest PreOrder value at this level (only instances of this FullPath).
– `Children` is the number of child nodes (in the data tree).
– `Descendants` is the number of descendant nodes (in the data tree) excluding child nodes.
– **The Level Structure**
– `Level` is the level number (identical to Level in FullPath).
– `LeftPos` is smallest PreOrder value at this level.
– `RightPos` is the biggest PreOrder value at this level.
– `NodeCount` is the number of nodes at this level (in the data tree).

A further level of meta-metadata is described elsewhere [12] and is used processing view data.

4 XPath Query Processing

Algorithms for evaluating location steps against prominent XPath axes are presented here and in [8], we provide algorithms for the entire set of 13 XPath Axes. In brief, our optimisation strategy is to prune the search space using both the query structure and axes properties and then apply content filtering for both content and arbitrary nodes.

- Step 1: **Structure Prune**. A sub-tree is extracted using the content and arbitrary nodes and is specific to the particular axis.
- Step 2: **Context Filter**. Where the context node has a predicate, this is used to filter the remaining sub-tree. For some axes (eg. Following and Preceding) this function involves a single lookup, while other axes (eg. Descendant and Ancestor), it requires a reading of each node in the sub-tree and this step is merged with Step 3 for performance reasons.
- Step 3: **Axis Prune**. Those nodes that do not evaluate to the Axis properties are filtered.
- Step 4: **Arbitrary Filter**. Where the arbitrary node has a predicate, this is used to filter the final result set.

Algorithm 1 PrecedingAxis (Query Q)

```
Require: Parsed Q provides context node con and arbitrary node arb
 1: Vector resultset = null
 2: int startSS = Min(FullPath.getPreStart(Q,arb))
 3: int endSS = Max(FullPath.getPreEnd(Q,con))
 4: if hasPredicate(con) then
 5:    endSS = Max(BaseIndex.getPre(Q,con))
 6: end if
 7: arbPreOrd = startSS // and now we need the corresponding preorder for con
 8: conPreOrd = Min(FullPath.getPreStart(Q,con))
 9: while arbPreOrd < endSS do
10:    if IsAncestor(arbPreOrd, conPreOrd) != TRUE then
11:       if supportsPredicate(Q, arbPreOrd) then
12:          resultset.add(arbPreOrd) // if no predicate, always add to resultset
13:       end if
14:    end if
15:    arbPreOrd = BaseIndex.getnextPre(arbPreOrd)
16:    conPreOrd = BaseIndex.getnextPre(conPreOrd)
17: end while
18: return resultset // a set of preorder values
```

4.1 Preceding Axis

The `preceding` axis contains all nodes in the same document as the context node that are before the context node in document order, excluding `ancestor` nodes of

the context node. Consider the query in *example* 1 to retrieve all **title** elements preceding the last **mastersthesis** (context node) element for the named **author** (arbitrary node).

Example 1. //mastersthesis[child::author = 'Peter Van Roy']/preceding::title

In the `PrecedingAxis` Algorithm, Q is an object of type Query, variables `con` and `arb` are strings while `StartSS`, `endSS`, `arbPreOrd` and `conPreOrd` are preorder values. Lines 2 and 3 perform the **Structure Prune** step using two single lookup functions. For Preceding queries, **Context Filter** is separated from **Axis Prune** as Context Filter requires a single lookup function and reduces the search space further (in this case) by moving the end point backwards. In lines 7 and 8, we obtain the initial {context,arbitrary} preorder pair. Lines 9 to 17 read through all nodes in the search space, testing for `Preceding` and *predicate* evaluations each time. In prior work [9], we demonstrated that functions `IsPreceding`, `IsFollowing`, `IsDescendant` and `IsAncestor` each execute with optimal efficiency given our index structure (nodes as {preorder,level} pairings). `BaseIndex.getnextPre` is passed a preorder value and returns the next value by checking its type against the FullPath index.

Algorithm 2 FollowingAxis (Query Q)

Require: Parsed *Q* provides context node *con* and arbitrary node *arb*

```
 1: Vector resultset = null
 2: int startSS = Min(FullPath.getPreStart(Q,con))
 3: int endSS = Max(FullPath.getPreEnd(Q,arb))
 4: if hasPredicate(con) then
 5:    startSS = Min(BaseIndex.getPre(Q,con))
 6: end if
 7: arbPreOrd = startSS // no need for context preorder value
 8: while arbPreOrd < endSS do
 9:    if supportsPredicate(arbPreOrd) then
10:       resultset.add(arbPreOrd) // if no predicate, always add to resultset
11:    end if
12:    arbPreOrd = arbPreOrd + BaseIndex.getSizeOfSubTree(startSS)
13: end while
14: return resultset
```

4.2 Following Axis

The `following` axis contains all nodes in the same document as the context node that are after the context node in document order, excluding `descendant` nodes of the context node. An XPath expression to retrieve all `volume` (arbitrary) elements with a value of 2 after the first `incollection` (context) element in the DBLP dataset is displayed in example 2.

Example 2. //incollection/following::volume[self::volume = '2']

The `Following` axis is similar to that of `Preceding` in that the **Structure Prune** and **Context Filter** steps are alike. However, the **Axis Prune** step uses the `getSizeOfSubTree` function (demonstrated in [9] to execute efficiently) is used to perform a jump within the search space. This jump ignores all descendants of the arbitrary node.

4.3 Ancestor Axis

The `ancestor` axis contains all ancestors of the context node. Consider the query in example 3 to retrieve all **book** elements (arbitrary node) that are ancestors of the **year** (context node) element with the given value.

Note that the algorithm creates a more refined search space (actually multiple sub-trees) than for the `Preceding` axis and thus, the **Prune Structure** step is inside a `for-next` loop. The algorithm begins by creating a set of FullPaths that meet the structure specified in Q, and then for each FullPath the algorithm will generate a result. Lines 3 and 4 preform the **Structure Prune** step using two lookup functions. Steps 2-4 are merged as all require a traversal of the sub-tree(s).

Example 3. //year[self::node() = '1998']/ancestor::book

Algorithm 3 Ancestor(Query Q)

```
Require: Parsed Q provides context node con and arbitrary node arb
 1: Vector resultset = null
 2: for each Fullpath do
 3:    int startSS = FullPath.getPreStart(Q,con)
 4:    int endSS = FullPath.getPreEnd(Q,con)
 5:    conPreOrd = startSS
 6:    arbPreOrd = Min(FullPath.getPreStart(Q,arb))
 7:    while conPreOrd < endSS do
 8:       if IsAncestor(conPreOrd,arbPreOrd) == TRUE then
 9:          if supportsPredicate(Q, conPreOrd) then
10:             if supportsPredicate(Q, arbPreOrd) then
11:                resultset.add(arbPreOrd)
12:             end if
13:          end if
14:       end if
15:       arbPreOrd = BaseIndex.getnextPre(arbPreOrd)
16:       conPreOrd = BaseIndex.getnextPre(conPreOrd)
17:    end while
18: end for
19: return resultset
```

5 Details of Experiments

Experiments were run using a 3.2GHz Pentium IV machine with 1GB memory on a Windows XP platform. Algorithms were implemented using Java virtual machine (JVM) version 1.5. The Extended Schema Repository was deployed using an Oracle 10g database (running the Windows XP Professional operating system, with a 3GHz Pentium IV processor and 1GB of RAM). The eXist database (version 1.0b2-build-1107) operated on a machine with an identical specification to that of the Oracle server to ensure an equal set of experiments. The default JVM settings of eXist were increased from -Xmx128000k to -Xmx768000k to maximise efficiency. All experiments used the DBLP dataset [14], containing 3.5 million elements, 400k attributes with 6 levels and a size of 127mb.

Table 1. DBLP Queries

#	**XPath**	**Matches**
Q1	//title[. = 'A Skeleton Library.']/ancestor::inproceedings	1
Q2	//year[self::node() = '1998']/ancestor-or-self::book	32
Q3	//mastersthesis[child::author = 'Peter Van Roy']/preceding::title	77
Q4	//incollection/following::volume[self::volume = '2']	3,123
Q5	/dblp/descendant::phdthesis	72
Q6	/descendant-or-self::article[* = 'Adnan Darwiche']	6
Q7	//book/parent::title	0
Q8	/dblp/phdthesis[* = '1996']	4
Q9	/dblp/descendant::book[child:author = 'Bertrand Meyer']	13
Q10	//article/@rating	61

Table 1 presents our XPath query set and for each query, we provide the number of results returned by the DBLP dataset. Each query is executed eleven times with execution times recorded in milliseconds (ms), together with the number of matches. The times were averaged with the first run eliminated to ensure that results are warm cache numbers.

Table 2 displays the execution times for eXist, the Extended Schema Repository (ESR), and in the final column the factor at which the ESR out-performs eXist. A value of 1 indicates that both are equal and anything less than 1 indicates that the ESR is slower than eXist. The range from 0.84 to 69.0 indicates that that at best, we were 69 times faster than eXist.

For queries Q3 and Q4, the eXist query processor failed to return any results as it does not support the `following` and `preceding` axes. The features of our metamodel allow us to quickly identify queries with paths not supported in the target database (see query Q7). For query Q6, we achieved a speed up of 69, as the eXist query processor is inefficient at processing predicates containing wildcards on frequently occurring nodes scattered throughout the database (i.e. the `article` node occurs 111,609 times throughout DBLP). In contrast, our processing strategy allows us to quickly filter for any predicate. Query Q8 contains

Table 2. Query Results

#	eXist (ms)	ESR (ms)	Factor
Q1	989.2	86.0	11.5
Q2	6,419.8	535.7	12.0
Q3		143.9	
Q4		1,265.5	
Q5	140.7	167.2	0.84
Q6	28,533.2	413.4	69.0
Q7	149.6	57.4	2.6
Q8	123.9	119.3	1.04
Q9	414.4	147.3	2.8
Q10	233.1	38.9	6.1

a wildcard and while the ESR was slighter faster for this query, we encountered instances where eXist out-performs the ESR as it can efficiently process predicates with wildcards on nodes that are *clustered into a very small segment* of the XML database.

For queries Q1 and Q2, we obtain improvements ranging from 11.5 to 12. The eXist database cannot process predicates on these queries efficiently as the context nodes have a very high frequency scattered throughout the database (i.e. for Q2 the `year` node occurs 328,831 times in DBLP). Although `year` nodes are scattered throughout the database, the `getNextPre` function provides us with direct access to them. Furthermore, our pruning and filtering techniques use metadata information within the FullPath index to quickly identify the relevant search space along the `ancestor` axis, or over any wide search space.

Queries Q5, Q9 and Q10 do not contain any predicates and the improvements range from 0.84 to 6.1. These are less impressive as queries without predicates bypass our optimisation steps 2 and 4. The ESR performs best against queries with selective predicates as they allow us to quickly prune and filter the database using all four optimisation steps. Query Q5 performs marginally better with eXist as the few `phdthesis` nodes in the database are clustered closely together.

6 Similar Approaches

In this work, optimisation takes place at the axes level, an approach also taken in [5] whereby recording minimal schema information (preorder, postorder and level values), they provide a significant optimisation for XPath axes. However, they do not employ the different levels of abstraction presented here and thus, operations such as `GetPreStart` and `GetPreEnd` for the FullPath object (to limit the search space) are not easily computable.

One of the earliest effort at generating schema information for semi-structured sources was in [6] where the authors introduced the concept of a DataGuide. They are used by query processing strategies to limit or prune the search space of the target database. However, DataGuides do not provide any information

about the parent-child relationship between database nodes and unlike our approach, they cannot be used for axis navigation along an arbitrary node. In [10], they overcame this problem by augmenting their DataGuide with a set of instance functions that keep track of parent-child relationships between nodes although [10] was an early indexing schema that did not cover all categories of XML queries.

The FIX index [16] is based on spectral graph theory. During the initial parsing of the XML dataset, FIX calculates a vector of *features* based on the structure of each twig pattern found in the dataset. These features are used as keys for the twig patterns and stored in the index. During query processing, the FIX processor converts a twig query into a feature vector and subsequently searches the index for the required feature vector.

Experiments illustrate that FIX has strong path pruning capabilities for high selectivity queries (i.e. queries that query a small portion of an XML document), especially on highly unstructured XML documents (e.g. TreeBank). However, results also indicate that FIX is poor at processing low selectivity queries, especially when the target database is highly structured. Furthermore, FIX supports only the `child`, `descendant`, `descendant-or-self` and `self` axes and does not support the evaluation of queries containing the `text`, `node`, `comment` and `processing-instruction` functions. In contract, we provide algorithms for the full set of XPath axes [8].

Perhaps the most significant work in this area can be found in [2] where they compile a significant amount of metadata to support the optimisation process but also provide a full query processor that optimises at the level of *query* rather than the level of *location path* as we do. This facilitates an access-order selection process where more than one location path can be processed simultaneously. Furthermore, they can process queries in both top-down and bottom-up directions, thus, providing a further level of optimisation based on the query type and database statistics. However, we have discovered improvements can be achieved by fine-tuning the axes algorithms. In §4, we illustrated how optimisation steps differ for each axis and in some cases, it was necessary to merge these steps.

7 Conclusions

In this paper, we introduced our metamodel for XML databases and demonstrated how it could be used to optimise XPath queries by rewriting the axes algorithms. Our experiments were conducted against the eXist database as many web sources indicated that eXist outperformed its competitors. Two important characteristics in the engineering of WIS applications is that they are interoperable and can be viewed at different levels of abstraction. XML an provide the platform for interoperability as it acts as a canonical model for heterogeneous systems and by using a metamodel approach, it has been shown to assist the integration process. We discovered that using a metamodel with different functioning layers facilitated well-engineered algorithms but also greater performances due to the type of information stored at each level. While similar efforts

in this area have demonstrated long build times for their indexes, we reported relatively small build times in our earlier work on constructing XML indexes [7]. This is extremely useful in our current area of research: managing updates for XML databases.

References

1. Aboulnaga A., Alameldeen A. and Naughton J. Estimating the Selectivity of XML Path Expressions for Interney Scale Applications. *Proceedings of the 27th VLDB Conference*, Morgan Kaufmann, pp. 591-600, 2001.
2. Barta A., Consens M. and Mendelzon A. Benefits of Path Summaries in an XML Query Optimizer Supporting Multiple Access Methods. *Proceedings of the 31st VLDB Conference*, Morgan Kaufmann, pp 133-144, 2005.
3. Boulos J. and Karakashian S. A New Design for a Native XML Storage and Indexing Manager. *Proceedings of EDBT 2006*, LNCS vol. 3896, pp. 755-772, 2006.
4. Bruno N., Srivastava D., and Koudas N. Holistic Twig Joins: Optimal XML Pattern Matching. *Proceedings of SIGMOD 2002*, ACM Press, 2002.
5. Grust T. Accelerating XPath Location Steps. *Proceedings of ACM SIGMOD Conference*, pp.109-120, ACM Press, 2002.
6. Goldman R. and Widom J. DataGuides: Enabling Query Formulation and Optimisation in Semisztructured Databases. *Proceedings of the 23rd VLDB Conference*, Morgan Kaufmann, pp 436-445, 1997.
7. Noonan C., Durrigan C. and Roantree M. Using an Oracle Repository to Accelerate XPath Queries. *Proceedings of the 17th DEXA conference*, LNCS vol. 4080, pp. 73-82, Springer, 2006.
8. Noonan C. The Algorithm Set for XPath Axes. *Technical Report ISG-06-03*, Dublin City University, November 2006.
   at: URL http://www.computing.dcu.ie/∼isg/technicalReport.html.
9. O'Connor M., Bellahsene Z. and Roantree M. An Extended Preorder Index for Optimising XPath Expressions. *Proceedings of 3rd XSym*, LNCS Vol. 3671, Springer, pp 114-128, 2005.
10. Rizzolo F. and Mendelzon A. Indexing XML Data with ToXin. *Proceedings of the 4th WebDB Workshop*, pp. 49-54, 2001.
11. Roantree M. The FAST Prototype: a Flexible indexing Algorithm using Semantic Tags. *Technical Report ISG-06-02*, Dublin City University, January 2006.
    at: URL http://www.computing.dcu.ie/∼isg/technicalReport.html.
12. Roantree M. and Noonan C. A Metamodel Approach to XML Query Optimisation (submitted for publication). *Proceedings of the 11th ADBIS Conference*, 2007.
13. The XPath Language,
    at: URL http://www.w3.org/TR/xpath, 2006.
14. Suciu D. and Miklau G. University of Washingtons XML Repository.
    at: URL http://www.cs.washington.edu/research/xmldatasets/, 2002.
15. Weigel F. et al. Content and Structure in Indexing and Ranking XML. *Proceedings of the 7th WebDB Workshop*, pp. 67-72, 2004.
16. Zhang N. et al. FIX: Feature-based Indexing Technique for XML Documents. *Proceedings of the 32nd VLDB Conference*, pp.359-370, 2006.