The Drug Ontology Project for Elsevier

An RDF Architecture Enabling Thesaurus-Driven Data Integration

J. Broekstra, C. Fluit, A. Kampman
Aduna BV, The Netherlands

F. van Harmelen, H. Stuckenschmidt
Vrije Universiteit, The Netherlands

R.Bhogal, A.Scerri, A. de Waard[§]
Elsevier B.V., The Netherlands

E. van Mulligen
Collexis B.V./ Erasmus University Medical Center, The Netherlands

Abstract

The DOPE project (Drug Ontology Project for Elsevier) is driven by the need to access multiple information sources through a single interface. In this paper, we describe how DOPE allows thesaurus-driven access to heterogeneous and distributed data, based on the RDF data model. The architecture allows for the easy addition of thesauri and data sources, and can facilitate explorations in ontology mapping and data integration.

1 Introduction

With the unremitting growth of scientific information sources, the need for integrated access to these sources becomes ever more urgent. The aim of the DOPE project (Drug Ontology Project for Elsevier) is to allow access to multiple information sources in the area of life sciences through a single interface, using semantic web data standards. The DOPE prototype[*] allows thesaurus-driven access to heterogeneous and distributed data, based on the RDF data model. This architecture allows for the easy addition of thesauri/ontologies[**] and data sources, to facilitate the investigation of ontology mapping and data integration issues.

Thesauri provide controlled vocabularies for indexing information, and by grouping relevant terms, help overcome some of the problems of free-text search. Examples of medical thesauri include MeSH (http://www.nlm.nih.gov/mesh/meshhome.html) and Elsevier's life science thesaurus EMTREE(http://www.elsevier.com/homepage/sah/spd/site/). These thesauri are currently used to index specific information sources – such as the abstract databases PubMed (http://pubmed.org) and EMBASE (http://embase.com/).

To utilise the advantages of thesauri, we would like to use them to disclose other data sources, as well. Currently there is no open architecture available to do so. Furthermore, since the mental models and common terms for data access diverge between communities, different thesauri need to coexist. An ideal architecture would allow for the disclosure of distributed and heterogeneous data sources through different thesauri. The aim of DOPE is to work towards such an architecture.

Semantic web technologies facilitate access to distributed data through open standards that allow semantic qualifications, such as RDF [7]. To successfully disclose these data via a single query interface, however, we need to align the diverse conceptualizations used by different data sources. RDF represents the data conceptualization in a schema definition, and can therefore be used to map the data sources to each other. To execute this mapping step, an RDF query language is needed. Of the many RDF query languages available, only a few provide the required transformation functionality – these include Triple and SeRQL [1]. We used SeRQL because it is already implemented in Sesame, the RDF repository used for DOPE (see below).

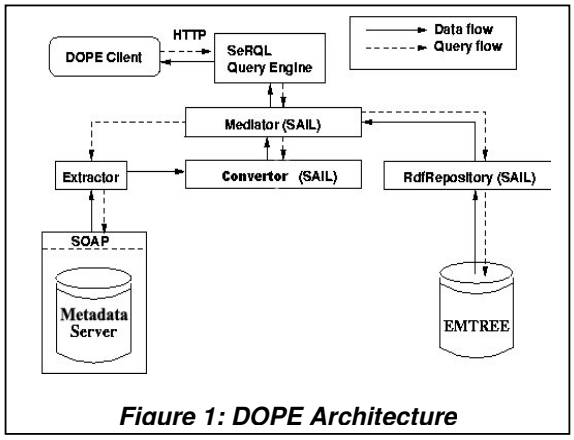

Figure 1: DOPE Architecture

The general architecture of DOPE is shown in Fig. 1:

1. DOPE Client: The DOPE Client makes use of a thesaurus-driven visualization technology called the Spectacle Cluster Map [2,3], developed by Aduna BV. The DOPE Client interacts with the other components via SeRQL queries sent by HTTP. The DOPE client is discussed in section 5.
2. EMTREE: Elsevier's main life science thesaurus, EMTREE 2003©, was converted to an RDF-Schema format. The thesaurus mapping to RDF is described in section 2.
3. Metadata Server: For the prototype, the full content of Science Direct (containing full text articles and several abstract databases) and the last 10 years of MEDLINE have been indexed against EMTREE 2003 thesaurus by the Collexis indexing technology. This indexing technology creates a weighted vector of a document to the thesaurus

[§] Corresponding author: a.dewaard@elsevier.com

[*] The DOPE prototype can be viewed on http://www.aduna.biz/dope

[**] We consider thesauri to be "lightweight ontologies" and, throughout this paper, use the two words interchangeably

terms by statistical methods, called a fingerprint (See (www.collexis.com) and [4] for details). The metadata is mapped to an RDF source model via an *Extractor* component, and accessed via a SOAP protocol. Details of the indexer are discussed in section 4, the mapping to RDF is described elsewhere [5].

4. The Mediator: The RDF repository Sesame [6] plays a central role in the DOPE prototype. A Sesame RDF database communicates with the Collexis server and the RDF version of EMTREE. The core of the connected system is the *Storage And Inference Layer* (SAIL) API that forwards incoming method calls from the SeRQL query engine to the relevant information sources. The mediation architecture is discussed in section 3.

2 Thesaurus Representation in RDF

Design

To access different information sources in a uniform way, we have to deal with heterogeneity at different levels: information sources may use a different syntax, different data models, and/or different conceptualizations. Syntactic heterogeneity is largely solved by the widespread use of XML as a basis for encoding information, but the problem remains that information sources often use different, application specific data models, encoded in DTDs or schema definitions. Aligning these data models from XML schemas is often difficult due to the conceptual ambiguity of XML schema definitions that do not, for example, distinguish objects from relations.

In the Semantic Web initiative, a number of proposals exist for metadata language standards. One of the most stable ones is the Resource Description Framework (RDF) [7]. RDF is not only assumed to be the lingua franca for encoding meta-data about any kind of information on the Web, it also provides the syntax for defining other semantic web languages like RDF Schema (RDFS) [8] or the 'Web Ontology Language' (OWL) [9]. To convert our sources into a uniform model, we have to convert data sources into RDF. Our first step in building a distributed architecture for multi-thesaurus querying, therefore, was to convert the EMTREE thesaurus to RDF format.

EMTREE is a thesaurus maintained by Elsevier as a terminological resource for researchers in life science. EMTREE is used to index EMBASE, an online database that is indexed by human indexers. EMTREE 2003 contains about 45,000 preferred terms and 190,000 synonyms, organized in a multi-level hierarchy. Making EMTREE available in RDF is not a trivial task, because the result of the process has to satisfy a number of requirements:

- Thesaurus maintenance should not be affected
- The RDF model has to behave in the same way as the original thesaurus.

As our aim is to support effective access to information sources, we consider the following tasks, connected to the formulation and execution of a thesaurus-based query:

A) Suggest Query Term. Based on query terms provided by the user, the system can suggest additional terms to refine the query.

B) Normalize Query Term. As the indexes of the information sources are normalized to preferred terms, each query stated to the system will also have to use these preferred terms. Therefore the same mechanism is needed on the query side.

C) Expand Query based on Hierarchy. The main task of EMTREE in querying documents is to expand the query specified by the user by broader, narrower and related terms.

D) Dynamically Generate User Interface. As EMTREE provides the basic knowledge structure for interacting with information resources such as EMBASE, it also provides the basis for generating a user interface (discussed in section 5).

A number of RDF-based thesaurus interchange formats have already been proposed [10],[11]. We adopt the following principles mentioned in the different proposals when defining an RDF representation of EMTREE to support query processing.

- A thesaurus is organized as a set of hierarchies of concepts each with a unique root
- Concepts are represented independently from the terms used to describe them
- The semantic relations are defined between concepts rather than terms

Based on existing proposals and the specific needs of the project, we chose the following ways of encoding the EMTREE thesaurus in RDF-based languages:

a) Representation of the EMTREE meta-model

Checking the consistency of a set of query terms (part of task A and D above) requires reasoning about types of EMTREE terms. In particular, we have to check whether a combination of EMTREE terms is consistent with the conceptual model behind EMTREE. This kind of reasoning is only supported by a logic-based language like OWL (though much of the expressive power of OWL will not be needed in this case). In order to support this kind of reasoning, we have to define the ontology that underlies EMTREE at the meta level. The actual terms and their relations are instantiations of this ontology. If we encode this ontology in OWL, we can check a given set of index terms against it. The following statements could for example be part of an EMTREE ontology:

(hasDrugLink domain DrugTerm)
(hasDrugLink range DrugLink)
(aspirin type DrugTerm)

The model states that the 'hasDrugLink' –relation can only hold between 'DrugTerms' and 'DrugLinks'. Further we state that 'aspirin' is a drug term. From this small model we can automatically deduce that aspirin should only be related to a DrugLink by this relation. Note that in this model terms act as objects that are described. The model uses semantic relations taken from the OWL language.

b) Model of (Parts of) the Concept Hierarchy

Tasks like query expansion (task B) and term suggestion (task A) mainly require the hierarchy of Terms that constitutes the backbone of EMTREE. Reasoning about this hierarchy is mostly restricted to the retrieval of broader and narrower terms. This kind of reasoning is supported by RDF schema. As fairly scalable tools exist for storing and querying RDF schema models, RDF schema representations of (parts of) the EMTREE hierarchy should be considered as default views on the Thesaurus. The following example could be part of a concept hierarchy of EMTREE's drug facet:

(c1 subClassOf c2)
(c1 label 'acetylsalicylic acid')
(c2 label 'salicylic acid derivative')

The example states that the concept denoted by c1 is a subconcept of another concept c2, Further, we have information about the terms that are used to refer to these concepts: 'acetylsalicyclic acid' and 'salicylic acid derivative'. In fact, this small example resembles the 'broaderTerm' relationship between the two preferred terms mentioned and can be used to support an explosion query over the more general term. Note that the model only uses semantic relations that are part of the RDF schema specification (i.e. subClassOf).

c) Representation of Synonym Relations

Some tasks like the normalization of index or query terms (task B and part of task A) do not even require taxonomic reasoning, as we only have to look up the preferred term that is in a functional relation to the index or query term. For this purpose a plain RDF file with a single relation or even a database table is a sufficient and efficient representation. The following example could be part of a synonym table:

(c1 preferredTerm 'acetylsalicylic acid')
(c1 synonymTerms [aspirin aspirine asperina])

This example defines the concept c1 using associated terms from the thesaurus distinguishing between preferred and synonym terms. It states that the preferred term of concept c1 is 'acetylsalicylic acid' and that 'aspirin' is just a synonym. This model can therefore be used to normalize an index or a query. Note that the model does not use any semantic relationship from a schema language.

Discussion:

- Instead of a complete migration of EMTREE to RDF, we dynamically generate views on parts of the complete model in RDF when required. We found RDF and RDF schema sufficiently expressive to represent the information required for the tasks we performed on the thesaurus.
- However, we are not fully utilising the capabilities of semantic web languages such as OWL. To perform more interesting semantic queries (such as "which drug produces which side effects?") a richer model of the information space, and the thesaurus, should be made. The architecture used, based on Sesame and SeRQL (discussed below) does allow for this.

3 Mediation Architecture: SAIL API

Design

Sesame [6] allows persistent storage of RDF data and schema information, and subsequent querying of that information. To do this, Sesame needs a scalable repository. A logical choice for such a repository is a DBMS: a large number of DBMS's have been developed, each having their own strengths and weaknesses, targeted platforms, and API's. For each of these DBMS's, the RDF data can be stored in numerous ways. Depending on the application domain, other forms of (persistent) storage may be more suitable, for example storage in main-memory, in files, or in some external source. As we would like to keep Sesame's architecture independent from the actual choice of repository, all storage-specific code is concentrated in a single architectural layer of Sesame: the *Storage And Inference Layer* (SAIL) [12].

This SAIL is a Java application programming interface (API) that offers RDF(S)-specific methods to its clients and translates these methods to calls to its specific repository. An important advantage is that it lets us build Sesame on top of a wide variety of repositories without changing any of Sesame's other components. Sesame's functional modules are clients of the SAIL API. Currently, there are six such modules delivered as part of the Sesame standard package: three *query engines* (RQL, SeRQL, RDQL) , an *RDF admin* module and the *extract* and *remove* modules. Naturally, since these modules operate on the public SAIL API, extending the set of modules for a domain- or application-specific purpose is possible.

Depending on the environment in which it is deployed, different ways to communicate with the Sesame modules may be desirable. For example, communication over HTTP may be preferable in a Web context, but in other contexts protocols such as Remote Method Invocation (RMI) or the Simple Object Access Protocol (SOAP)[13] may be more suited. In other settings, it may be more desirable to use Sesame as a

Java library instead of a separate server, and use direct Java method calls on either the modules or even directly on the SAIL. For maximal flexibility, the handling of these protocols has been placed outside the scope of the functional modules. Instead, protocol handlers are provided as intermediaries between the modules and their clients, each handling a specific protocol.

The introduction of the SAIL and the protocol handlers makes Sesame into a generic architecture for RDFS storage and querying, rather than just a particular implementation of such a system. Adding additional modules or protocol handlers is also possible.

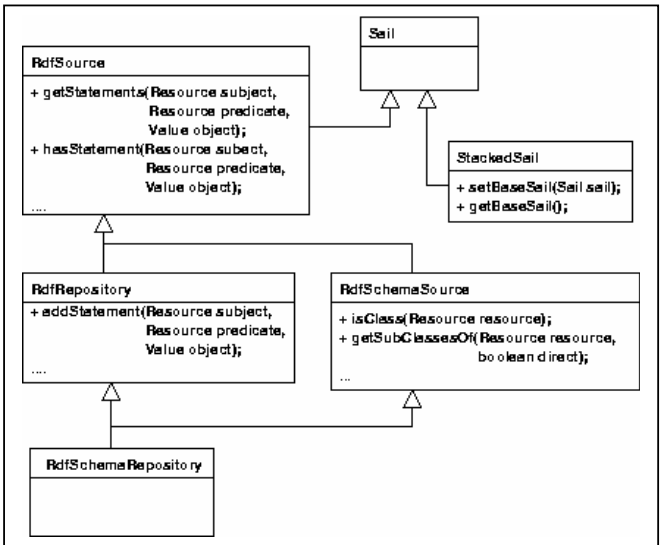

Figure 2: The SAIL Architecture

The core of the SAIL API is a set of four interfaces that extend across two dimensions: RDF vs. RDF Schema, and retrieve vs. manipulate (see fig. 2) :

- RdfSource offers retrieval methods for RDF, e.g. getStatements(subject, predicate, object).
- RdfRepository extends RdfSource and adds manipulation methods for RDF, e.g. addStatement(subject, predicate, object).
- RdfSchemaSource also extends RdfSource, adding retrieval methods for RDF Schema, e.g. getInstancesOf(class).
- RdfSchemaRepository extends both RdfRepository and RdfSchemaSource, combining RDF manipulation and RDF Schema retrieval methods. This interface does not add any new methods.

Any combination of these interfaces is possible, where each implementation can make its own choices on how to support storage and inference. This allows the higher functionality of Sesame (such as querying) to be ported to any environment in a flexible manner. The SAIL interface allows the implementation of SAILs that do not connect directly to a storage device, but instead connect to another SAIL. This mechanism allows for the 'stacking' of SAILs on top of each other, which is useful for storage-agnostic functionality like caching or synchronization.

Discussion

Overall, the SAIL architecture allowed a flexible and easy development of the prototype system. Coupling different parts of the distributed architecture was greatly facilitated by conforming to the SAIL API for the different components. Since this allowed reuse of existing components (such as the SeRQL query engine, and several different storage and inferencing components available through the SAIL), it reduced development cost considerably.

However, a number of problems arose during the implementation:

- The SAIL implementation for the DOPE prototype does not handle concurrency issues specific to a distributed environment. The converter stores a retrieved result to re-use for answering the next incoming query, but if two users query the system at the same time, this will have unexpected results. This is mainly an engineering issue that can be solved with a small additional effort.
- The DOPE implementation for the mediator is very domain-specific and would have to be re-implemented for a new domain. Efforts are underway to develop a more generic SAIL-based mediator implementation.
- The performance of the system has a number of bottlenecks. A number of these bottlenecks have been identified to be easily solvable with some engineering effort, but some are inherent to the distributed nature of the system.

4 The Metadata Server

Design:

The metadata server consists of the following backend services that form the heart of the Collexis indexing technology:

- The *abstraction component* is used to recognize in a piece of text a number of phrases (concepts) as defined in a thesaurus and assign a relevance score to each concept. The set of concepts and their relevance score are named by Collexis a “concept fingerprint” and are stored together with the document metadata in a remote database.
- The *matching engine* allows for identifying those fingerprints in a collexion that most closely resemble a given query fingerprint. A document fingerprint can also be used as a search fingerprint. The matching engine has been based on the vector space model of Salton [14] and knows a number of variations.
- The *selection engine* takes care of the combination of matching and conditional searches on metadata. Any indexed metadata field can be used in a Boolean query together with a match on a search fingerprint.

- The *relation engine* maintains relations between concept fingerprints. It can be used, for instance, to relate fingerprints for a given author -- an aggregation of document fingerprints -- with the original document fingerprints, to see to what extent an author's subject coverage coincides with that of the set of documents.
- The *SOAP interface* provides functions to query the fingerprint and metadata repository. The statements retrieved can be viewed as expressions stating that a particular concept is present in a document with a certain weight.
- To query the metadata server through RDF, an *Extractor* component is deployed which, through use of the SOAP interface, converts the Collexis information in an RDF format that is a 1:1 mapping to the original information: the *physical model* (described in detail in [5]).

Discussion:

- The prototype currently uses a number of thresholds on relevancies and the numbers of results to get reasonably relevant documents and terms as well as a timely response. There are currently some performance bottlenecks caused by inefficient querying and network overhead.
- There are problems with performance, which mostly have to do with the query procedures between the Sesame system and the Collexis-SOAP interface. Improvements in the Sesame implementation for DOPE and the query mechanism of the DOPE Browser would likely make the thresholds on maximum number of documents and terms unnecessary, or at least orders of magnitude larger.
- Some of the performance issues we encountered are a result of the conversion of the Collexis metadata to an RDF format. In our experience, the writing of an "RDF-wrapper" for the metadata server is likely to be the biggest task to solve, both intellectually and in terms of development effort.

5 The DOPE Client UI

Design:

The DOPE Client user interface[†] is based on Aduna visualization technology, also called the Spectacle Cluster Map [2], [3] and implemented as a Java client side application. Java version 1.4 is the minimum requirement, which at the moment is available for the Windows, Mac OS X, Linux and Solaris platforms. Communication with the DOPE Sesame server takes place using the Sesame Java client library, which communicates with the server using SeRQL queries over HTTP.

EMTREE 2003 contains about 46,000 preferred terms and 190,000 synonyms, organized in a multi-level hierarchy. Clicking through a hypertext version of the thesaurus in order to create complex queries is too much effort to be acceptable. In the DOPE client, the user can quickly focus on a topically related subset of both the document collection and the thesaurus. First, the user selects a single thesaurus term. The system then fetches all documents indexed with that focus term, as well as all other terms those documents are indexed with. These co-occurring terms are used to provide an interface in which the user can explore the set of documents indexed with the focus term.

For example, a user can enter the string "aspirin" in the text field at the upper left of the figure. The system then consults Sesame for all concepts than can be related to this string. It responds with a dialog showing four possible EMTREE terms, asking the user to pick one. (This dialog is omitted when there is only one exact match with an EMTREE term.) We assume the user chooses the term "acetylsalicylic acid", this is now the focus term.

The system consults Sesame again and retrieves (at maximum) the 500 most relevant documents about "acetylsalicylic acid", their metadata fields (such as titles and authors) and the other most important terms with which these documents are indexed. The co-occurring terms are presented in the tree at the left hand side, grouped by their facet term (the most generic broader term, i.e. the root of the tree they belong to). The user browses the tree and checks one or more checkboxes that appear behind the term names, to see their contents visualized at the right hand side.

Figure 3 shows the state of the interface after the user has checked the terms "mortality", "practice guideline", "blood clot lysis" and "warfarin". The visualization graph shows if and how their document sets overlap. Each sphere in the graph represents an individual document, with its color reflecting the document type, e.g. full article, review article or abstract. The colored edges between terms and clusters of spheres reveal that those documents are indexed with that term. This visualization shows that within the set of documents about aspirin there is significant overlap between the terms "blood clot lysis" and "mortality", and that 4 of the practice guidelines documents relate to these two topics as well.

Various ways exist to further explore this graph. The user can click on a term or a cluster of articles to highlight their spheres and list the document metadata in the panel at the lower right. Moving the mouse over the spheres reveals the same metadata as a tool tip. The visualizations can also be exported to a clickable image map that can be opened in a web browser.

The user can start with a new query by typing in a new search string. This will empty the rest of the interface and load a new set of documents and co-occurring terms. The Thesaurus Browser provides an alternate starting point for a next query. When a focus term has been selected, the user can click the "Navigate

[†] The DOPE prototype can be viewed on http://www.aduna.biz/dope

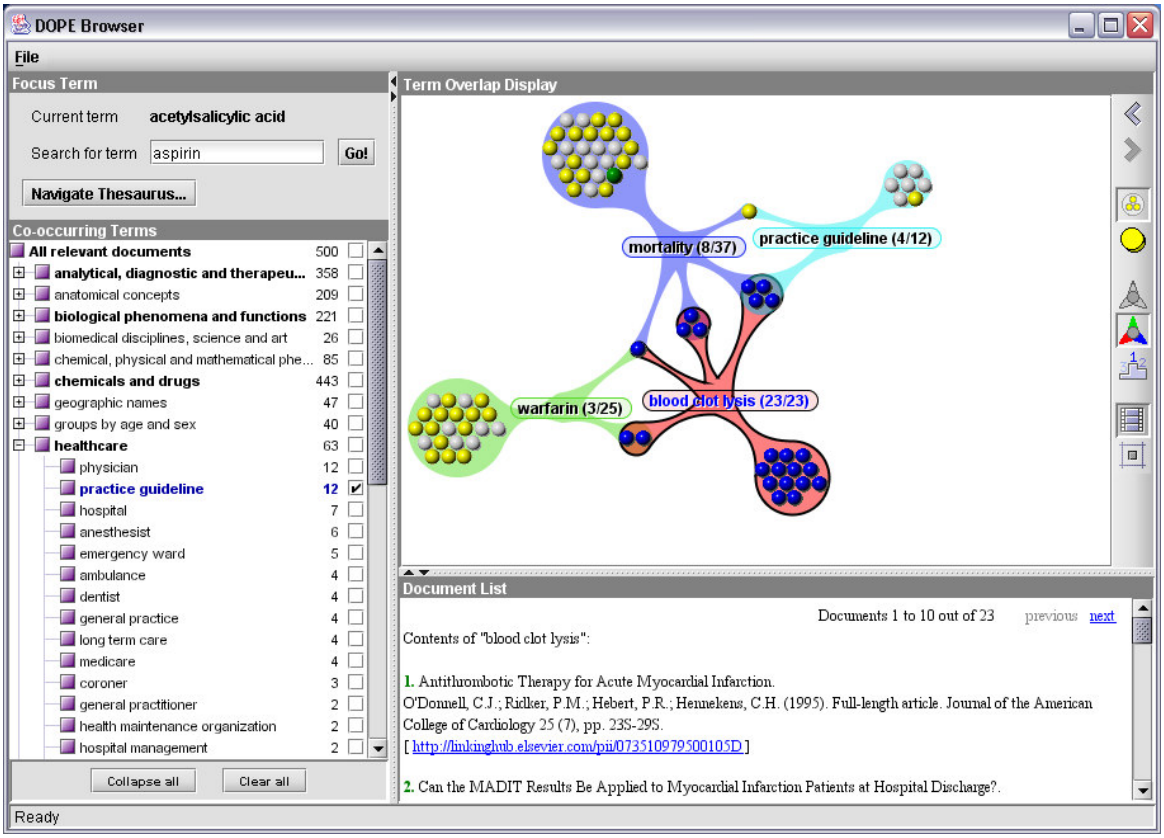

Figure 3: the DOPE Client UI

Thesaurus..." button at the upper left. She is then confronted with a dialog that lets her select a new focus term, by browsing through the thesaurus, starting from the focus term. The user can iteratively select a broader, narrower or alternative term until she encounters a term that she wants to make the new focus term.

The visualization conveys several types of information. The user obviously sees document characteristics, such as index terms and article types. Visualizing a set of terms shows all Boolean combinations, without the need for the user to express them all separately. The graph also shows how terms relate within the selected set of documents, i.e. if they have some overlap and if so, which documents constitute that overlap.

Discussion:

We have performed a user test with 10 potential end users at a Drug Discovery conference in 2003. A full account of the results of the user studies is given elsewhere [5] but, in summary, the visualization tool was found to be a useful to aid in the information discovery process, and in particular it was found to provide richer contextual information about the documents presented and simpler scanning of the data sources

Users indicated that the main benefit of the UI is to aid the exploration of a large, mostly unknown information space rather than offer support for searching for concrete articles. Examples of beneficial applications mentioned by potential end users included:

- Filtering material for preparing lectures about a certain topic,
- Supporting graduate students in doing literature surveys (e.g. using a "shopping basket" to collect search results).

A more advanced potential application that was mentioned was to monitor changes in the focus of the research community. This, however, would require a mechanism for filtering documents by date of publication, as well as the visualization for changes that happen over time.

Three issues were identified that would improve the visualisation:

- Interpretation of the subset names was found problematic
- Complex term overlaps were difficult to interpret
- Manipulation of the graph could be improved

6 Conclusions and next steps

We conclude that:

- The system we have built is a working implementation of semantic web technologies, which can serve as an open architecture for thesaurus-based access to distributed data sources.
- RDF was used as a common language for integrating data and schema information from diverse partners in industry and academia. We created RDF models of the EMTREE thesaurus and wrapped an existing information source into an RDF model. We found RDF and RDF schema sufficiently expressive to represent the information.

- We used the SeRQL query and transformation language to access and convert data in the system. Using SeRQL as a transformation language was essential for mediating between different sources. The fact that many RDF query languages do not support the transformation of data into a different formal is major problem for their use in real applications. Recently, there is some work on providing explicit view definition languages for RDF information. We expect this work to further improve the engineering of RDF-based applications.
- The DOPE prototype currently provides access to about half a million full text articles and more than ten million abstracts. These are numbers that are realistic for real life applications. We did not encounter major bottlenecks with respect to storing and processing RDF data in the Sesame system. In fact, we found that other factors such as the bandwidth of the metadata server and the user interface were the limiting factors with respect to scalability.
- To provide the required functionality, we had to link our RDF infrastructure with XML technologies, both at development and at run time. We converted an XML representation of the thesaurus into RDF and provided an RDF wrapper for the metadatabase based on an existing SOAP interface. The use of XML as a common denominator made it easy to connect these technologies. While this integration could easily be done on a technical level, we still had to cope with the differences at the conceptual level. The conceptual differences between the XML and RDF data model, for example, forced us to develop the two-stage transformation process described in the paper. There are attempts to leverage this problem by providing a tighter integration of XML and RDF technologies (e.g. (for example [15,16]). We consider these attempts to be crucial RDF-based application engineering.
- The DOPE Client UI works well with the rest of the technology, and offers an interesting graphical environment for scientific users to browse this vast data store.

There are several issues that could be improved on:

- In this prototype we only used one thesaurus, so the extension with different thesauri, which was our original goal, has not been tested in practice.
- The limitations of the JAVA-based client interface forced us to limit the set of documents presented to the user to the 500 most relevant documents. With respect to the problem of high communication costs and low bandwidth, results from the area of distributed databases could be used to optimize data access using techniques such as pre-fetching or caching.
- We are not fully utilising the semantic capabilities of RDF, or OWL. The Sesame architecture and SeRQL query language do, in principle, allow for a much greater semantic richness of the queries.
- We have only used data indexed by a specific indexing technology (Collexis). When using this architecture to disclose different metadata repositories, there could be an improvement in performance and semantic search capabilities. However, in our experience the writing of an "RDF-wrapper" for the metadata server is likely to be the biggest task to solve, both intellectually and in terms of development effort.
- With the user tests, it is hard to say to what extent the GUI obscures or clarifies the data integration and thesaurus manipulation that occurs with DOPE.

Therefore, promising steps forward would be:

- **Inclusion of multiple ontologies** Work is currently underway at the Vrije Universiteit to map the Gene Ontology [17] to EMTREE, which would allow the disclosure of genetic and document data through a single user interface
- **Generalization of the architecture** This work can be used as a basic architecture for further explorations of distributed data sources and thesauri. To open it up for general use, the architecture needs to allow for the inclusion of distributed databases for RDF data (other than those of Collexis used in this project) Work is currently underway between Aduna and the Vrije Universiteit to investigate this.
- **Inclusion of full-text search** To do a full comparison between thesaurus-based search and full-text search, which is an issue of interest to us, we need to have a representative set of data to query via both methods via a bare-bones interface. Some steps have been taken in this direction, but further work needs to be done to disclose a representative amount of data which will allow realistic user tests.
- **Retrieval of semantic relations** The RDF query engine is in principle capable of answering such entity- relationship extractions to answer questions such as: "What diseases does this drug treat?" or "What drugs treat this disease?". A further collaboration between all parties participating in DOPE is being explored to investigate this promising route.

7 Acknowledgements

We thank Jan van Buel and Ian Crowlesmith for helping us understand the intricacies of EMTREE. This work was funded by the Elsevier Advanced Technology Group.

8 References

[1] Arjohn Kampman, Jeen Broekstra: SeRQL User Manual. Technical Report, Aduna, 2003. See http://www.openrdf.org/doc/SeRQLmanual.html

[2] Christiaan Fluit, Marta Sabou, Frank van Harmelen. Supporting User Tasks through Visualisation of Light-weight Ontologies. In: S. Staab and R. Studer (eds.), Handbook on Ontologies in Information Systems. Springer-Verlag, 2003

[3] Christiaan Fluit, Marta Sabou, Frank van Harmelen. Ontology-based Information Visualisation. In: V. Geroimenko, C. Chen (eds.), Visualizing the Semantic Web. Springer-Verlag, 2003.

[4] Van Mulligen EM, Van Der Eijk C, Kors JA, Schijvenaars BJ, Mons B., Research for research: tools for knowledge discovery and visualization. Proc AMIA Symp. 2002. 835-9.

[5] H.Stuckenschmidt, A. de Waard, R. Bhogal, Chr. Fluit, A. Kampman, J.van Buel, E. van Mulligen, J. Broekstra, I. Crowlesmith, F. van Harmelen and A. Scerri, Exploring Large Document Repositories with RDF Technology – the DOPE Project. IEEE Intelligent Systems, special Issue on the Semantic Web Challenge, accepted for publication 2004.

[6] Broekstra, J., Kampman, A., and van Harmelen, F. Sesame: An Architecture for Storing and Querying RDF data and Schema Information. Proceedings of the First International Semantic Web Conference ISWC 2001.

[7] O. Lassila and R. Swick. Resource description framework (RDF). Proposed recommendation, W3C, January 1999. http://www.w3c.org/TR/WD-rdf-syntax.

[8] Dan Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF schema. Working draft, W3C, April 2002. http://www.w3.org/TR/2002/WD-rdfschema-20020430/

[9] L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Web ontology language (OWL) reference version 1.0. Working draft, W3C, November 2002. http://www.w3.org/TR/owl-ref/.

[10] Phil Cross, Dan Brickley, and Traugott Koch. Rdf thesaurus specification. Institute for Learning and Research Technology, 2001. http://ilrt.org/discovery/2001/01/rdfthes/

[11] B.M. Matthews, K. Miller, and M.D. Wilson. A thesaurus interchange format in RDF. LIMBER project, 2002. http://www.limber.rl.ac.uk/External/external.htm.

[12] See http://sesame.aidministrator.nl/publications/api/server/

[13] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1. W3c note,World Wide Web Consortium. See http://www.w3.org/TR/SOAP/

[14] G. Salton, Wong, and C.S. Yang. A vector Space Model for automatic indexing. Communications of the ACM, 18:613 620, 1975.

[15] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Peer data management systems: Infrastructure for the semantic web. In Proceedings of the 12th International World Wide Web Conference, 2003.

[16] J. van Ossenbruggen and L. Hardman. Smart style on the semantic web. In Proceedings of the Semantic Web Workshop at the 11th International World Wide Web Conference, 2002.

[17] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. Nature Genetics, 25(1):25--29, May 2000.