Integrating Ontology Storage and Ontology-based Applications Through Client-side Query and Transformations

Peter Mika

Vrije Universiteit, Amsterdam
pmika@cs.vu.nl

Abstract. This paper investigates the integration of ontology storage and ontology-based applications through the example of the EnerSearch case study conducted within the On-To-Knowledge research project. We look at the problem of integrating the Sesame storage and query facility with its client application and identify both functional and technical needs for a new software package for client-side query and transformations. We introduce the solution developed during the case study that also opens the way for creating Portable Inference Modules that capture transformation knowledge in a modular way. We discuss future extensions to this package based on the belief that the issues at hand will equally effect future Semantic Web applications.

Introduction

The World Wide Web has drastically changed both the form and availability of information in the past years. As the number of web pages and users on the public internet skyrocketed, companies around the world have just as eagerly adopted internet technologies as a basis for their own networked, electronic information stores. The resulting intranets, filled with weakly structured, weakly organized information, have created a knowledge management problem that could not be any more handled by existing document management solutions.

The On-To-Knowledge research project [1] is a joint, EU-funded research effort that aims to improve on the state of the art of web-based knowledge management solutions for SMEs and distributed organizations by leveraging ontologies. Instead of building a complete knowledge portal, the approach of On-To-Knowledge is to provide for the interoperability of the components developed within the project, based on open standards and agreements among the partners, such as a common data model for representing domain knowledge. The aim of the case studies within On-To-Knowledge is to validate this approach by showing that it is indeed possible to integrate these components into customized solutions that fit the specific knowledge management problems of the case study partners.

EnerSearch, a pan-European research organization investigating new IT based business strategies and customer services in deregulated energy markets, carried out the case study that we will use as an example in this paper. The company joined On-To-Knowledge for what ontologies promise with respect to greater user satisfaction through more effective querying of its corporate memory. The status of EnerSearch as a virtual knowledge organization warranted that the value attributed to finding the right information is high enough for an increased interest in state-of-the-art semantic solutions.

The ontology developed in the case study is a combination of a lightweight domain ontology obtained by natural language processing with OntoExtract [2] and a rich ontology reverse engineered from the publication database of EnerSearch[1]. The ontology, which currently contains over 140,000 statements about approximately 20,000 resources, is represented in RDF(S) format and stored in a repository at the central Sesame storage server [3]. It provided semantic data for the searching and browsing interfaces that were generated using the QuizRDF [4] and Spectacle [5] tools, respectively.

The case study provided ample evidence that ontology-based tools should not be evaluated as standalone applications, but should be qualified as part of integrated frameworks or applications. In the following we discuss our experience in application integration to demonstrate the relevance of such an approach.

In the course of the case study we identified a bottleneck in integrating the ontology storage facility and the ontology-based applications: the division of query and transformation work between client and server inhibited creating applications that would have scaled up to industrial standards. Moreover, support was lacking for custom inference using semantics that is available only at the client side.

This paper presents a detailed description of the issues at hand and proposes a solution to address the need for client-side query and transformations. In the following section we take a closer look at the workings of ontology storage facilities through the example of the Sesame storage server. This will lead us to a better understanding of the observations that follow in Section 0 with regard to the present approach to query and transformations. These observations also form requirements in that they point to the necessity of client-side query and transformations.

The solution that was developed during the case study is presented in Section 0. We also show that this solution finds an even wider application in creating Portable Inference Modules (PIMs). These modules not only serve to improve efficiency, but enable the sharing of transformation knowledge by capturing it in a portable way. We outline future work in Section 0 and offer some conclusions in the closing section of the paper.

[1] The database contains metainformation about the documents such as the author, title, publication date etc. It has been used on the current website to render a table of the publications.

The Sesame RDF(S) storage and query facility

In many early Semantic Web initiatives semantic data was embedded within HTML or XML pages or it was simply stored within separate files and served to clients by web servers (cf. SHOE [6] or OntoBroker [7]). However, as ontologies became all the more valuable and ontology-based applications started to appear, the need has arisen for specialized ontology servers to make data manipulation efficient and to provide advanced services such as querying, versioning, access control and security.

To better understand the problems related to ontology storage and query, it's helpful to take an in-depth look at how these facilities operate. For this purpose, we've chosen the Sesame storage and query server for its role in the EnerSearch case study and for its open architecture. Sesame, originally developed and further supported by AIdministrator B.V. in the Netherlands, has become open source since March 2002.

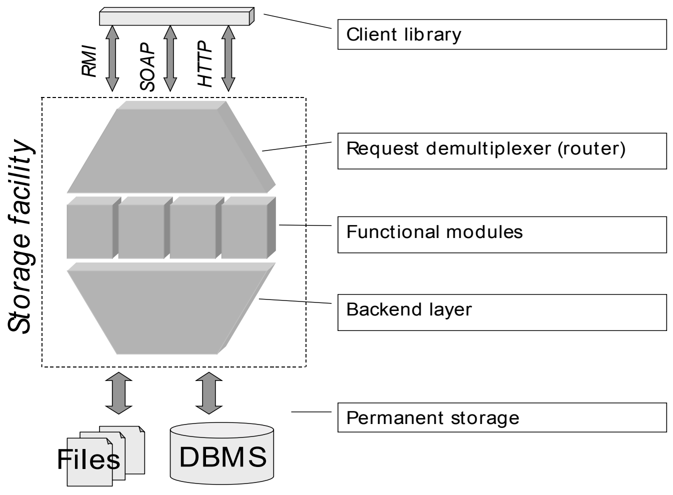

Figure 1. Overall design of web-based storage facilities

Although actual implementations differ, ontology storage facilities share the common design of web-based storage facilities shown in Figure 1. In the following, we will describe how Sesame implements the components of this design.

As all Semantic Web tools, Sesame primarily communicates with clients through HTTP, though in the future some services will be offered through SOAP and RMI as well. Sesame also has a client API, which is a Java programming interface that abstracts remote communication. In other words, programmers can use the API to access the services of Sesame in their ontology-based applications without having to deal with the actual communication between client and server.

The next layer, the request router forwards requests to the functional modules of Sesame. The server currently implements the following operations on repositories:

1) *Data manipulation*

   a) Upload of RDF(S) documents. Currently only RDF-XML format is supported.
   b) Extraction of ontology and data in RDF(S). It's possible to separately extract the ontology (schema) and the instance data.
   c) Removal of statements.
   d) Clearing the entire repository.

2) *Query*

a) Formal queries. Support for two RDF query languages have been implemented, namely RQL [8] and RDQL [9], with RQL being strictly more expressive than RDQL.[2]
b) Browsing of the repository. The user can navigate through the RDF graph model by clicking through the nodes that are of interest.

From the applications point of view, the significance of the query engines is that they provide access to the content of the repository on the level of the data model, i.e. the RDF graph model complemented with the semantics of the reserved vocabulary of RDF(S).[3] This means that queries may not only match triples, but can also refer to paths in the graph model or elements of the schema. For the software engineer this allows to formulate complex queries in a relatively simple language.

A support function to querying is the built-in inference module of Sesame that applies the RDF(S) closure rules [10] to data, thereby calculating the complete RDF model of the repository. At present inferencing is applied at upload time, where the alternative would be to run the inference module at query time. The difference is that the present solution suits query intensive applications better as opposed to applications that rely on frequent manipulation of the data.

The functional modules access the repositories through another abstract layer called SAIL. The purpose of the SAIL API is to hide the implementation of permanent storage. Storage SAILs exist for relational databases (PostgreSQL, mySQL) with several others under way. Development plans call for in-memory and peer-to-peer network storage.

Revisiting the design shown in Figure 1, the reader may notice that ontology-based storage facilities such as Sesame only differ from other web-based storage systems in their functional modules. More specifically, ontology servers are partially aware of the semantics behind the data which enables them to offer services such as query and reasoning. However, as we will see in the next section, server-based query and transformations do not always provide an ideal solution for building scalable ontology-based applications.

Ontology storage, query and transformations in the EnerSearch case study

As mentioned above, in the framework of the EnerSearch case study Sesame was used to store the lightweight domain ontology extracted by OntoExtract and the ontology obtained by converting the publication database of EnerSearch. This repository was then queried for the semantic data that was needed by the two ontology-based applications, the QuizRDF search engine and the Spectacle presentation. Through the course of building these applications, we have made six observations that led to the development of a new package for client side query and transformations. Note that these observations are generic on purpose. In particular, nothing is claimed about the absolute performance or scalability of Sesame, for such statements could only be made within a framework and in comparison to some other product, theoretical limit or baseline.

Observation 1: Small, frequent queries are inefficient.

The first, naïve implementation of the Spectacle application called for querying Sesame on an on-demand basis, i.e. whenever the value of a property or the subclass(es) of a class became relevant. While testing the application with ontologies containing more than a hundred classes, it was found that posing small, frequent queries to Sesame presents a serious bottleneck in generating the presentation. Even though

[2] Note that parallel to the development of ontology representation languages, query languages go through an intensive phase of development. Moreover, there is no widely accepted RDF(S) query language yet that would also have the sanction of a standardization body.

[3] Similarly to how relational query languages operate on tables and XML query languages work on the XML tree, regardless of external representation.

the users would not encounter the problem[4], runtimes on the scale of days were deemed unacceptable even for off-line website generation.

Here, the bottleneck is caused by the communication costs of accessing the server through HTTP.

Observation 2: Some queries that are useful from an application perspective cannot be expressed or efficiently executed.

As it turned out, some of the more advanced queries also suffer from slow evaluation times. An example of such a query is the following RQL select statement that returns the concepts from the ontology along with page(s) where they appear.

```
select c, p from {p} oe:isAbout . rdf:type {c} . rdf:type { concept }

where concept = oe:Concept

using namespace

   oe = http://ontoserver.cognit.no/otk_rdf# ,

   rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

The path expression in the query matches a three edge long path in the graph and returns two points along the path. Despite its relative simplicity, evaluation times for this query ranged over a few dozen minutes even for an ontology with few hundred concepts.

An ever more challenging problem turned out to be querying for all direct subclass relations within an RDF(S) model. While such information is readily available in the level of the SAIL API, there is no direct support for it in the RQL language. More specifically, while there is a query with the intended meaning, evaluating it calls for matching all possible pairs of classes. Again, the net effect is an execution time that is unacceptable for sizeable ontologies.

Finally, there are transformations that require data manipulations that cannot be expressed in a declarative language, but require the full power of a programming language. We will see such an example in the following section when transforming a literal value of comma-separated words into several separate relations.

At the moment steps are being taken to optimize both the query language and the evaluation of queries. Naturally, there will always be inefficient queries in every sufficiently expressive language; the goal here is to optimize the queries that are useful from an application perspective. Unfortunately, there are very few real applications to provide feedback to that work.

It also seems that the technical barrier to optimization is the cost of communication with the underlying permanent storage, although an in-memory storage implementation could remedy the situation at the expense of size scalability.

Observation 3: The smaller the repository, the faster a transformation executes.

While this seems a trivial observation, its consequences are far reaching. Consider the case of applying the RDF(S) closure rules to data that is being uploaded to Sesame.

Figure 2 shows the processing speed for a series of 17 consecutive uploads to an originally empty repository for two implementations of the RDF inference engine. For the 'old' series we used the first, naïve implementation, while values for the 'new' series were obtained using an optimized version of the inference module. The optimizations, for example, streamline communication with the underlying database and employ heuristics such as dependencies between the inference rules.

[4] In the current version, both the navigation and content of the Spectacle website is rendered in advance, and server only adds the design at request time. For a truly dynamic site slow queries could hinder the user experience as well.)

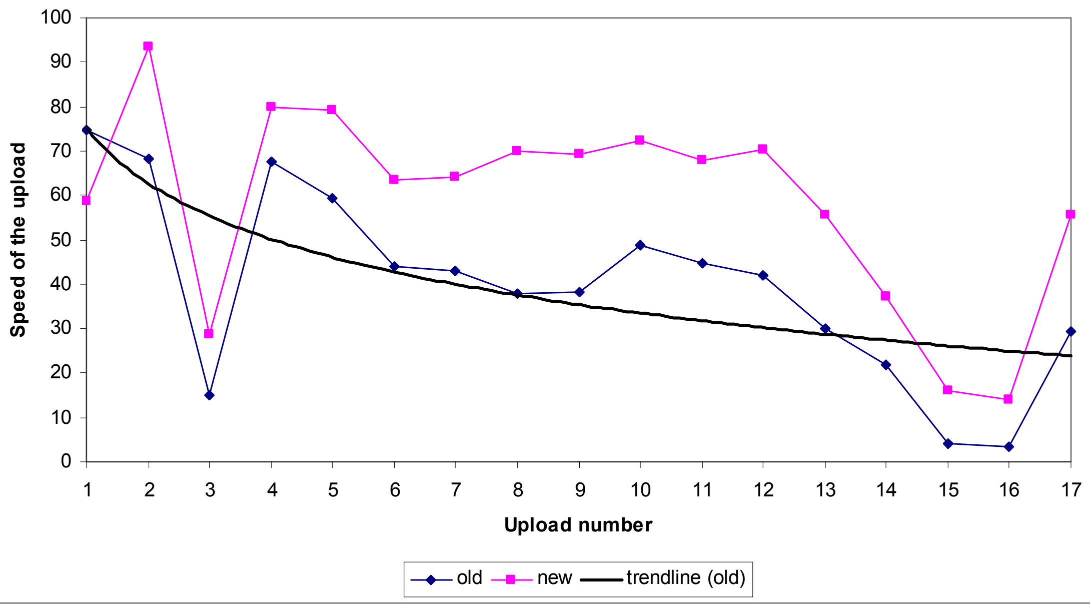

Figure 2. The total number of statements per second over a series of 17 uploads for the old and new implementation of the inference engine, with a logarithmic trend line for the former.

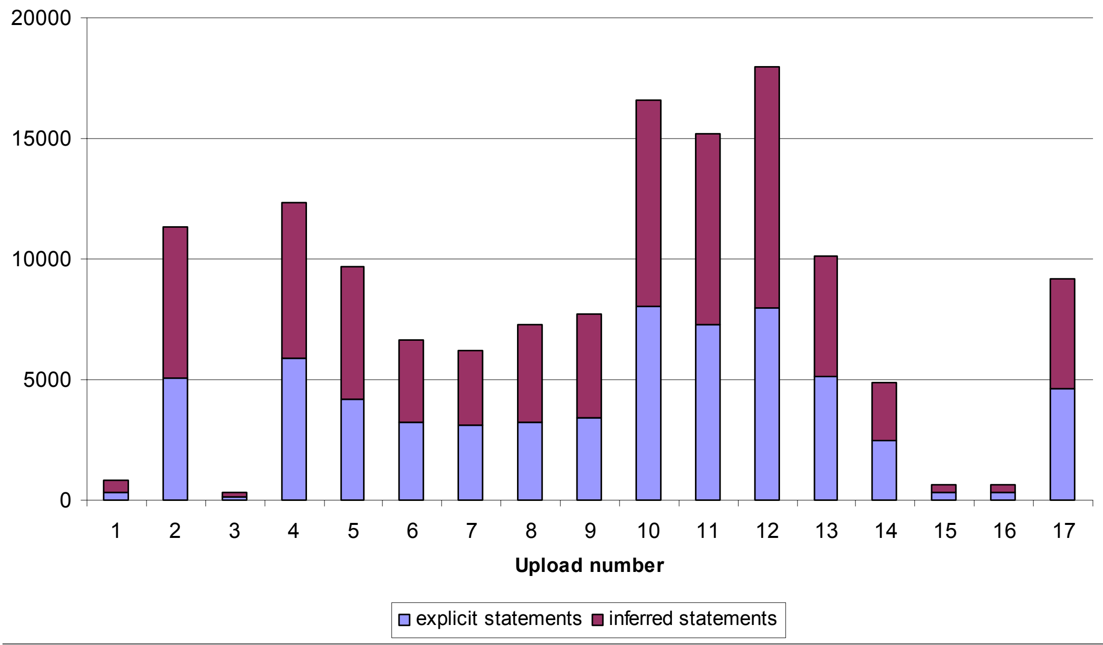

Figure 3. Size of the uploads as the total number of statements.

The logarithmic trend line fitted on the graph reveals that the time needed for the inference is proportional to the number of statements already in the repository. As the figure also demonstrates, improvements over earlier implementation of the inference module considerably alleviated the situation and most of the time during upload is now actually spent on parsing the input and adding the parsed statements to the repository. For relational data stores, it would be even possible to move some of the computation to the database server in the form of built-in procedures at the expense of portability. Nevertheless, the basic observation would remain the same, due to the very nature of inference: every added statement has to be matched against the rules and the statements in the repository to see if it results in new, inferred statements.

Observation 4: Server-side transformation is inefficient for small datasets.

Looking at Figure 2, the reader might have wondered what is the reason behind the variation in the upload speed, i.e. the noise that seems to be superposed over the general declining trend. Figure 3 provides the clue: the size of the respective uploads in Figure 2.

The sharp dips on the earlier figure correspond to comparatively small uploads on the order of a few hundred statements compared to several thousand statements for the other uploads. However, the difference in the time needed for the upload to complete was not nearly as big as the difference in sizes. In other words, the speed of processing smaller uploads is significantly lower then for processing larger ones.

Similarly to Observation 1, the explanation concerns the fixed administration costs of carrying out an upload. The consequence for the applications that rely on frequent manipulation of smaller parts of the dataset is again a performance problem.

Observation 5: Transformations weigh heavily on the storage facility.

Although this cannot be seen from the previous figure, reasoning also consumes considerable processing power and makes the storage facility much less responsive to other requests during the inference process. Inference over more expressive languages such as DAML+OIL [11] will be even more costly and might seriously set back performance with respect to base functionality.

Observation 6: There is a distinct need for client side transformations.

Even if server side query and transformations could be made efficient at will, clients are in all cases aware of much more of the semantics behind the data – semantics that in many cases cannot be described in today’s ontology languages, yet do not warrant reasoning with a full scale predicate logic.

A typical example from the case study is the situation when a new relation is composed from other properties. For example, if we would like introduce an occursIn relation between concepts and pages we would need to compose three relations using the following rule:[5]

$$\exists \text{instance}\,(\text{page}, \text{oe}:\text{isAbout}, \text{instance}) \wedge (\text{instance}, \text{rdf}:\text{type}, \text{concept}) \wedge$$
$$\wedge\, (\text{concept}, \text{rdf}:\text{type}, \text{oe}:\text{Concept}) \rightarrow (\text{concept}, \text{oe}:\text{occursIn}, \text{page})$$

Such composition cannot even be described in DAML+OIL. Certainly, languages could always be extended to allow for encoding all kinds of rules and axioms (up to the expressiveness of full logic), but we would most likely still prefer a less expressive language considering the lightweight nature of the ontology.

Moreover, if the transformation includes resources or properties from separate repositories, inferencing cannot be done at any single server. How to execute queries on disparate servers is an oft-neglected research area, even though it is paramount to realizing the Semantic Web.

The above observations compelled us to develop the core of a new API for client-side query and transformation. This package –code named on2k.graph- is presented next.

The on2k.graph package

The on2k.graph package is a generic Java API that operates on a powerful graph paradigm to support programmatic query and transformations of RDF(S) models. The solution itself is based on a freeware Operations Research package from DRA Systems.

[5] The reader might conclude that the above RQL query returns those concept and page pairs that satisfy the precedence of this rule. In fact, many reasoning tasks and constraint checks can be rewritten as simple actions/checks on the result of queries.

The API is geared toward importing, manipulating and exporting mono-property graphs, i.e. graphs where every edge is labeled using the same property. This design decision is motivated by

- Performance issues. Ontologies, especially lightweight ontologies, typically contain only a select number of properties. Mono-property graphs are therefore ideal cross-sections from a performance point of view: importing them is efficient and once the transformation has been carried out then the result can be exported just as efficiently in the form of such graphs.
- Functional issues. Queries and transformations typically involve only a select number of properties and the resources related by them. Some examples from the EnerSearch case are presented later in this section.

Therefore, the solution addresses both the performance and functional concerns that have been listed as observations in the previous section.

The package has generic interfaces to import and export graphs using any source or target that is capable of producing or consuming statements with a given predicate. Actual implementations of these interfaces are given for Sesame using the client library.

Support is also provided for the mapping of ontological classes to Java classes. (At present the programmer provides these classes, although it would be possible to compile them from their descriptions as done in the work of Cranefield [12].) These dynamically instantiated classes are used to hold literal properties and to provide operations on literal properties. For example, in the EnerSearch domain the class Publication provides a getDate() operation that returns the date of the publication.

Simple graph manipulations such as taking the union or intersection of graphs and the support for symmetric properties are provided by the underlying graph environment. Additional expressiveness, however, is kept to the minimum, because extra expressive power could become an overhead for applications using lightweight semantics. Instead a helper class built using the functionality of the package provides basic operations such as determining subclasses, direct subclasses, superclasses and instances of classes. (Types of instances are readily available without this class.) A more extensive library of queries, inferences, constraint checks might be added later and invoked on an on-demand basis.

Using on2k.graph in the EnerSearch case

Originally, the package has been developed within the case study to optimize communication with Sesame when generating the Spectacle presentation, but it has been later employed to facilitate simple transformations as well.

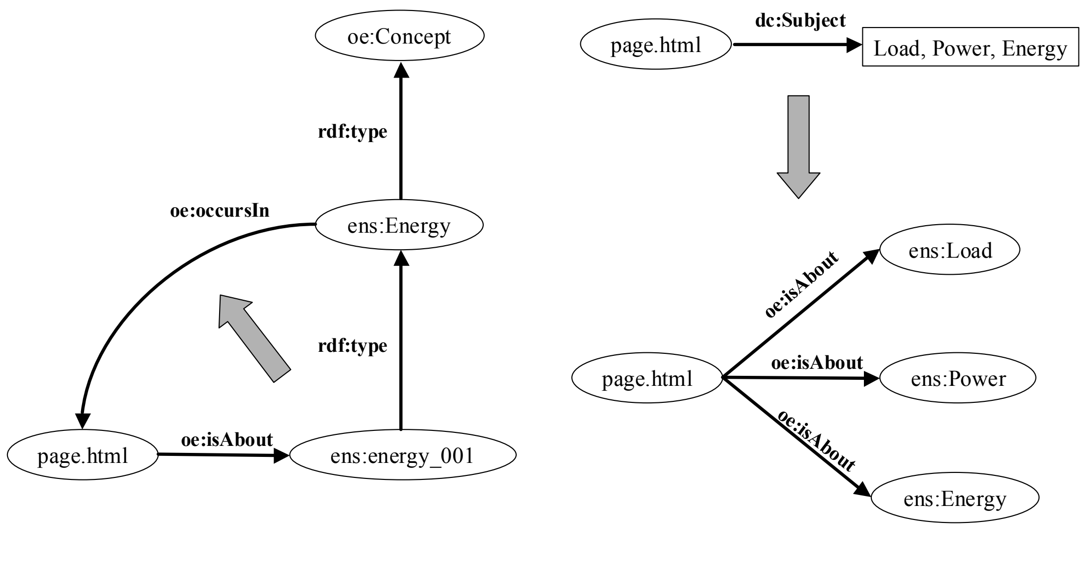

Figure 4. An example of a simple inference and a transformation on the domain ontology.

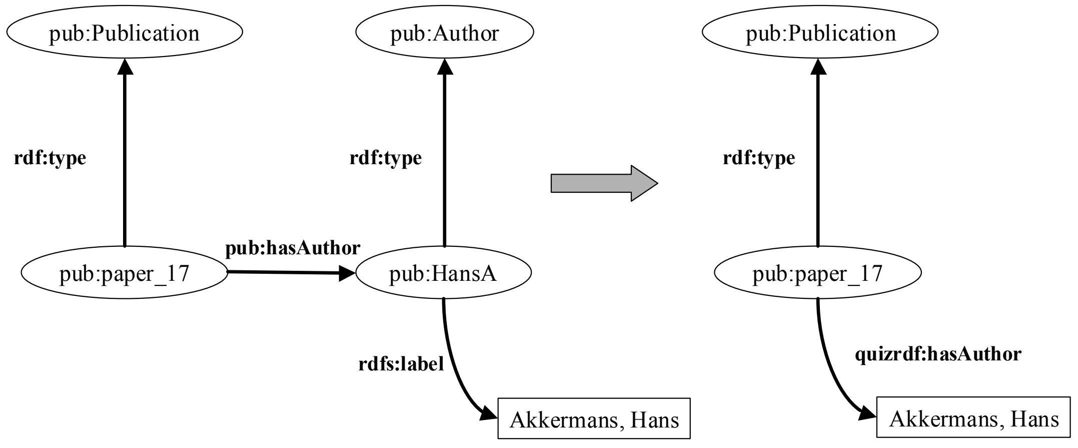

Figure 5. An example of a transformation on the publication ontology.

Some examples of the transformations carried out are shown on Figure 4 and Figure 5. On the domain ontology, we infer the occursIn relation as mentioned before and convert the comma separated list of values in the Dublin Core [13] Subject field into relations between pages and concepts. (See Observation 2 in Section 0.) Furthermore, to transform the publication ontology into a form preferred by QuizRDF, we transform the relation between a publication and an author into a literal attribute of the author resource. Other examples not shown on the figures include inverting relations and filtering out non-key concepts.

All transformations work in three steps, (1) querying the necessary graphs from Sesame, (2) carrying out the transformations and (3) uploading the resulting graph(s) if necessary. For better performance, the output of a transformation may be kept in memory if it's used by another transformation, as is the case with the ones shown in Figure 4. Currently, there is no automated discovery of such a dependency (or trigger), although this information could be inferred based on the input/output relations of our transformations.

The on2k.graph package not only enabled client side transformation (see Observation 6), but also allowed the expressiveness mentioned in Observation 2. It greatly improved performance and made it possible for our application to scale to the level of the EnerSearch ontology (addressing the issues in Observations 1, 3 and 4.) The overburdened central Sesame server was relieved from the effects of transformations (see Observation 5), except for RDF inference. On the client side memory requirements have never exceeded 100 MB, not even with all resources kept in memory as Java classes.

Portable Inference Modules

The queries and transformations implemented using the on2k.graph package do not need to be limited to the scope of a single application. In fact, once there is an agreement on the programmatic representation of the data model of the ontology language, the procedures that operate on this model become what we call Portable Inference Modules. PIMs are a sharable, reusable form of transformation knowledge captured in a procedural form.

PIMs need not to be constrained to represent custom semantics either. Semantics of RDF-S, for example, can be fully described as a set of inference rules that operate on the RDF model [10]. PIMs can be used to capture the built-in inference rules and constraints of ontology languages, thereby making it possible to use only the subset of a language that is necessary for an application or to mix-and-match functionality from various ontology languages.

Moreover, if PIMs are trusted and fine grained enough to be explanatory, a series of such modules can be used to support a chain of reasoning, since they provide evidence that can be checked if necessary by executing them over the data. Looked at from another perspective, PIMs can be taken as a compression mechanism: applications only need to transfer the explicit data and the PIMs, because the full model of the data (i.e. all statements that are valid) can be reconstructed from these components.

PIMs can also be considered for use on the server side as the semantic equivalent of stored procedures.[6]As we know it from the database world, such server-side transformations are indeed justified in many scenarios.

The concept of Portable Inference Modules differs from Semantic Patterns introduced by Staab, Erdmann and Maedche [15] in several respects. Semantic Patterns are defined on a higher level of abstraction: while PIMs capture inference knowledge, patterns concern themselves with design knowledge. Patterns consist of a natural language description and a set of formal constraints on their instantiations. (There are four types of constraint relating to what an instantiation must, must not, should and should not entail, based on the input.) PIMs, on the other hand, are given in a programmatic form (although they might be accompanied by formal descriptions in the future), which means that they work on the level of the representation model of a specific language (RDF). Due to the higher level of expressiveness PIMs are expected to be practical building blocks of Semantic Web applications, while semantic patterns are better suited for communicating, cataloguing, reverse-engineering and problem-solving on the design level.

Future work

The major value driver of the Semantic Web will be its applications ability to reason over data. Unlike software using conventional databases, semantic applications not only reuse previously stored data, but base their workings on facts inferred from data.

Despite its significance, there is very little attention paid to how inference knowledge will be represented on the Semantic Web. Even the most expressive of ontology languages such as DAML+OIL provide only limited ways to express axioms. Although they could be extended up to the expressiveness of first order predicate logic, the resulting language would be a large overkill for applications operating with lightweight ontologies, such as the system developed within the EnerSearch case study.

However, if at some point in the future an agreement is reached on how transformation knowledge should be represented in a declarative way, it may become possible to compile declarative descriptions into procedural form. Alternatively, such descriptions may accompany PIMs as an interpretation of their workings.

In a similar fashion parameterized rules could be translated into parameterized procedures. Note that many of the modeling constructs used in ontology languages can be interpreted as such: for example, the transitivity of a property can be considered a generic rule that is parameterized by a property. Once we have a 'transitivity PIM' available, and the transitivity of a property becomes evident, the PIM can be loaded, instantiated and executed to carry out the constraint check or inference.

In the late future rule types as the ones used in the CommonKADS methodology [14] for clustering similar rules may also be considered as templates for PIMs.

Conclusion

Due to their complexity and dynamic nature, Semantic Web applications of the future are likely to be loosely coupled, distributed or agent-based solutions woven from a variety of components and services. In such a setting efficient query and transformation of semantic data will largely influence the performance and scalability of applications.

As we have seen through the example of the EnerSearch case study conducted within the On-To-Knowledge project, the present division of query and transformation between the ontology storage and query facility and its clients presents a significant bottleneck. The solution is to move part of these tasks to the client, which is also aware of much more of the semantics behind the data. Besides greater efficiency, this also opens the way to the creation of portable transformation modules that can be shared between agents and applications.

Transformation knowledge is key to creating interoperable semantic applications. Therefore, when captured in the right form, it may become a valuable commodity on the Semantic Web.

[6] Note that “server side” is a relative notion: storage engines such as Sesame can be embedded into applications, if required.

Acknowledgements

The author would like to express his gratitude to Frank van Harmelen (VU), Jeen Broekstra (VU, AIdministrator) and Arjohn Kampman (AIdministrator) for their valuable comments to this paper, with special thanks to Arjohn for his work with measuring and charting the performance of Sesame's inference engine.

References:

[1] The European On-To-Knowledge project (IST-1999-10132). See http://www.ontoknowledge.org.

[2] R. Engels. CORPORUM-OntoExtract: Ontology Extraction Tool. On-To-Knowledge Deliverable D6. See http://ontoserver.cognit.no.

[3] J. Broekstra and A. Kampman. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. On-To-Knowledge Deliverable D10. See http://sesame.aidministrator.nl.

[4] U. Krohn and J. Davies. The Search Facility RDFferret. On-To-Knowledge Deliverable D11. See http://www.ontoknowledge.org.

[5] Spectacle: White Paper on Advanced Information Disclosure, 2001. See http://www.aidministrator.nl/publications/SpectacleWhitePaper.pdf

[6] J. Heflin and J. Hendler. Dynamic Ontologies on the Web. In *Proceedings of American Association for Artificial Intelligence Conference 2000*. See http://www.cs.umd.edu/projects/plus/SHOE/

[7] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman et al., editor, *Proceedings of DS-8: Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, 1999. See http://ontobroker.semanticweb.org/

[8] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis & M. Scholl.
RQL: A Declarative Query Language for RDF. To appear in *Proceedings of the 11th International Conference on the WWW*, Hawaii, 2002. See http://zeus.ics.forth.gr/forth/ics/isl/publications/paperlink/dql-rdf.pdf.

[9] A. Seaborne. RDQL: A Data Oriented Query Language for RDF Models, 2001. See http://www.hpl.hp.com/semweb/rdql.html.

[10] P. Hayes. RDF Model Theory. W3C Working Draft, April 2002. Available online at http://www.w3.org/TR/2002/WD-rdf-mt-20020429/.

[11] F. van Harmelen, P. Patel-Schneider and I. Horrocks. Reference description of the DAML+OIL ontology markup language, March 2001. See http://www.daml.org/2001/03/reference.html.

[12] S. Cranefield. UML and the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001.

[13] Dublin Core Metadata Element Set, Version 1.1: Reference Description. Available at http://dublincore.org/documents/1999/07/02/dces/

[14] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt W. Van de Velde and B. Wielinga. *Knowledge engineering and management: The CommonKADS Methodology*. MIT Press, Massachussets, 1999.

[15] S. Staab, M. Erdmann and A. Maedche. Engineering Ontologies using Semantic Patterns. In *Proceedings of the IJCAI'01 Workshop on E-business and the Intelligent Web*. Seattle, 2001. See http://www.csd.abdn.ac.uk/~apreece/ebiweb/programme.html.