Achieving Efficient Access to Large Integrated Sets of Semantic Data in Web Applications

Pieter Bellekens[1], Kees van der Sluijs[1], William van Woensel[2],
Sven Casteleyn[2], Geert-Jan Houben[1,2]
[1]*Technische Universiteit Eindhoven, PO Box 513, 5600 MB Eindhoven, The Netherlands*
[2]*Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium*
{p.a.e.bellekens, k.a.m.sluijs, g.j.houben}@tue.nl,
{William.Van.Woensel, Sven.Casteleyn, Geert-Jan.Houben}@vub.ac.be

Abstract

Web-based systems can exploit Semantic Web-based approaches to link data and thus create applications that make the most out of the combination and integration of different sources and background knowledge. While a lot of attention is paid to the opportunities that this linking of data on the Web provides, the reality of implementing such solutions with currently available semantic technologies creates a serious engineering challenge. In developing such applications in a commercial setting, we have been confronted with requirements and conditions that show the limitations of current technologies for this type of Web applications. Using our experience from iFanzy, we illustrate in this paper the issues and steps in turning the concept of access to semantically integrated content into solutions that use available technology.

1. Introduction

Many Web applications today are characterized by the integrated use of data from several already available sources or applications. Their engineering therefore includes two important steps in the specification and subsequent efficient implementation of that integration. More and more of them use techniques from the Semantic Web initiative for this purpose. One of the strengths of the Semantic Web is the ability to combine and integrate data from different data sources, thereby increasing the total amount of knowledge that was contained in the separate sources. By obtaining more, derived knowledge out of the combination of data, applications can offer more and new functionality compared to the original individual sources or applications.

When we observe current Semantic Web-based applications, we see that most of the applications using Semantic Web data do this on a relatively small scale. However, one of the main attractions of such Semantic Web-based applications is that, like with the 'normal' World Wide Web, they can operate in a ubiquitous and large-scale setting. This certainly applies now that many large Semantic Web data sources are becoming available for integration, either in native RDF or as a transformation of an originally differently structured source. Example sources are DBpedia [15] and IMDb[1] which have over 218 million triples and 53 million triples respectively. Integrating and combining these data sources into a Web application poses challenging engineering problems.

A straightforward, first engineering approach to implement such Web applications is to put everything in currently available RDF repositories. This is an interesting approach as it allows us to obtain additional functionality in the application in terms of knowledge derived from the combination of the included sources. However, this approach leads to huge data sets that pose significant scalability problems (see e.g. [1] for an overview of problems with current RDF storage solutions). Therefore, this first step of specifying the integration and combination has to be followed by a second engineering step that considers the actual efficient realization and implementation of this integration.

We have gained experience with this realization step in several systems, e.g. CHIP [4], iFanzy [5]. There we experienced what was needed to achieve efficient access to the large integrated sets of semantic data that are used in our Web applications to deliver the desired functionality.

[1] http://imdb.com/

978-0-7695-3261-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWE.2008.18

In this paper we report on our experience with the realization of iFanzy (developed in collaboration with Stoneroos Interactive TV, Ltd.[2]). iFanzy[3] is a personalized TV guide application aimed at offering users television content in a personalized and context-sensitive way. It consists of a client-server system with multiple clients and devices such that the user can ubiquitously use TV set-top box, mobile phone and Web-based applications to select and receive personalized TV content. TV content and background data from various heterogeneous sources are integrated to provide a transparent knowledge structure which allows the user to navigate and browse the vast content sets nowadays available. Semantic Web techniques are applied to make the interconnections between the various data and content. The resulting RDF/OWL knowledge structure is the basis for iFanzy's main functionality like semantic searches of the broadcast content and execution of context-sensitive recommendations. iFanzy differs from other semantic TV recommender systems, like for instance presented in [7], [10] and [3], because we focus on very large datasets (and live datasources) and integration of that information. We also use the larger structure of our integrated set for recommendation. For the recommendation part of our application we can reuse many of the works on recommendation in the TV domain. Think for instance of MovieLens[4] that makes use of collaborative filtering. For an overview of different recommendation strategies e.g. refer to [14].

In [6] we have described the first step in the conception of iFanzy, which describes at the functional level how the application combines data from several sources to provide the desired functionality. As this system is moving towards a real commercial application, we have to ensure that the architecture of the system can meet the demands of a large-scale Web-based usage. We note here that in a lot of comparable work, this ambition of efficient access is often secondary to the ambition to achieve "interesting" derived knowledge. With iFanzy we put in considerable effort to realize significant improvements in the engineering of the efficient access. As iFanzy was nominated for the Semantic Web Challenge 2007 [5] and ended runner-up, it poses a good representative application and we used it in this research to experiment with this important and often underrated engineering step, and in this paper we share our experiences and results.

The paper describes how the general recipe to link the data at the semantic level in one's Web application leads to concrete challenges for realizing efficient access to the data on real-time timescales. We do so on the basis of our concrete experience with representative data from the iFanzy research, as this is not only a very characteristic example but also a challenging one for which the experimental documentation of the engineering steps can benefit a large class of similar applications.

If we look at related work we see that some approaches exist in trying to handle large ontological structures, for instance [13] and [16] which both deals with large ontologies, both using a data partitioning approach to increase performance. In contrast, we focus less on size of the ontology but more on the instances of those ontologies and on a much larger scale: where [13] and [16] deal with thousands of elements we look at data with tens of millions of elements. However, the techniques they use are still interesting and to a small extend applicable to our case. Also some semantic repositories focus on scalability using big datasets in sizes that are relevant for us (e.g. OWLIM [9], for a performance overview see [12]). These systems mainly focus on scalability in loading times and inference capabilities. Even though these systems are very interesting and we regularly test new systems on performance they currently still fall short with our stringent real-time requirements on query evaluation for the size of datasets and complexity of queries we are facing.

This paper is structured as follows. Section 2 explains the general recipe for semantically integrating data from different sources and applications. Section 3 shortly introduces the engineering steps required for efficient access to integrated data. Section 4 explains the first phase, data preparation. Section 5 elaborates on the decomposition of data sources for better performance and on the subsequent query splitting. Section 6 explains optimizations concerning reasoning, and section 7 discusses the use of existing techniques and tools to further improve performance. Finally, section 8 presents conclusions and future work.

2. Linking Data and Integration

Before we can turn to the second step of engineering for achieving efficient access to the linked data, we need to set the stage by shortly explaining the general recipe for the first step of semantically integrating data from different sources and applications. As we mentioned before, we illustrate and document this here in this paper for data obtained from iFanzy as one possible representative, but this general

[2] http://www.stoneroos.nl/

[3] http://ifanzy.nl/

[4] http://www.movielens.org/

recipe we have also used in the development of other systems, like CHIP [4].

The main data sources used in iFanzy are the BBC Backstage[5] data set, the XMLTV[6] data set (which was obtained from crawling several online TV guides), the IMDb schema and data set, the TV Anytime[7] genre classification, the OWL Time ontology[8], a Geo Ontology (which we constructed on basis of IMDb location information), and RDF WordNet[9].

In abstract, the recipe for semantic integration consists of four steps:

1. *Making TV metadata available in RDF/OWL*: First, we make the relevant metadata from various data sources available in RDF/OWL. For example, we use three live data sources, online TV guides in XMLTV format (e.g. 1.2M RDF triples for the daily updated programs), online movie databases such as IMDb in custom text format (e.g. 53M triples including trailers from Videodetective.com), and broadcast metadata available from BBC-backstage in TV-Anytime format (e.g. 92K triples, daily updated). All these sets of metadata give us a quite detailed description of available TV programming and related material.
2. *Making relevant vocabularies available in RDF/OWL*: Having the metadata available, it is also necessary to make relevant vocabularies available in RDF/OWL. We created a genre ontology (5K triples) based on the TV-Anytime genre hierarchy that is used to describe all the genres that are used in our metadata sets (i.e. IMDb, XMLTV and BBC Backstage). We used the SKOS vocabulary[10] to express the relationships between genres. All these genres play a role in the classification of the TV content and the user's likings (in order to support the recommendation). For time and location-based reasoning we use the W3C Time ontology[11] (1.5K triples) and the location hierarchy as used in IMDb (60K triples) respectively. We also used WordNet 2.04 as published by W3C (2M triples) to exploit language relations like synonyms and hyponyms.
3. *Aligning and enriching vocabularies/metadata*: Here we did (a) alignment of genre vocabularies, (b) semantic enrichment of the genre vocabulary in TV-Anytime, and (c) semantic enrichment of TV metadata with IMDb movie metadata.
   a. First, aligning our genre ontology to the genre vocabularies used in the metadata sources was a small semi-automated exercise in which several translations were specified towards the TV-Anytime vocabulary, such as the associations between `xmltv:documentaire` and `tva:documentary`, between `IMDb:thriller` and `tva:thriller`, and between `IMDb:sci-fi` and `tva:science_fiction`. Simple matches like `IMDb:action` to `tva:action` were executed automatically by a string matching algorithm, while less straightforward matches were executed by a domain expert.
   b. Second, for the semantic enrichment of the genre vocabulary,
      i. `skos:narrower` relations are introduced based on the original TV-Anytime XML genre hierarchy, for example between `tva:sports` and `tva:soccer`.
      ii. `skos:related` relations are defined based on partial label matching, for example between `tva:sport_news` and `tva:sport`, or by using background knowledge of a domain expert, e.g. between sibling genres like `tva:rugby` and `tva:American_football`.
   c. Third, in terms of semantic enrichment of the TV metadata (that can come from different TV guides in different languages) we use an automated algorithm to couple the programs that happen to be movies to the corresponding movie information in the IMDb database. As titles might not also be matched one-on-one, consider for instance the semantic equivalent titles "Buono, il brutto, il cattivo, Il (1966)" and "The Good, the Bad and the Ugly", we consider not only the title fields, but also use AKA-title information and information on the director to be certain that a program matches a movie.
4. *Using the resulting RDF/OWL graph for recommendations*: To recommend TV programs or movies, the resulting RDF/OWL graph is extended with the user model knowledge such that the eventual RDF/OWL knowledge structure can be directly used for the recommendation. What happens is that when a specific user rates a program *P*, implicitly program *P* is rated together with all programs (and actors, directors and persons) that are related in the knowledge structure. Moreover, all programs with a genre that is related to a genre

5 http://backstage.bbc.co.uk/

6 http://xmltv.org/wiki/

7 http://www.tv-anytime.org/

8 http://www.w3.org/TR/owl-time/

9 http://www.w3.org/TR/wordnet-rdf/

10 http://www.w3.org/TR/skos-reference/

11 http://www.w3.org/TR/owl-time/

of *P* are rated, as well as the genres themselves via `skos:related` and `skos:narrower` relations. In this way, ratings are added within the user's context. When querying the graph, query expansion is used, exploiting ontology relationships like synonym relations from WordNet and the `skos:narrower` and `skos:related` relationships from the vocabularies.

Now that we have described the first step, namely the semantic linking of the data to arrive at the derived knowledge for the recommendations in the system, we turn to the concrete reality of realizing acceptable efficiency for this conceptual solution.

3. Engineering Efficient Access to the Integrated Data

The iFanzy scenario has given us a realistic challenge of a large, connected set of data in a Web application for which it is necessary to make the querying and retrieval of this data as efficiently as possible, in order to meet the demands of the application and its usage. Therefore, for the purpose of this presentation of experience results, we concentrate on the efficient access to the set of RDF-based data that constitutes the essence of this Web application, and disregard application-specific constraints about freshness of data which are not relevant for the general consideration of how to realize the efficient access.

In our consideration of steps to improve access efficiency, we look at the following issues:

1. Preparing the data in the proper format
2. Decomposing the combined data (graph) model into connected parts and rewriting queries in accordance with the data decomposition
3. Implementing inference
4. Using currently available tools and technology

In the next sections, we will consider each of these issues in isolation and thus document our (representative) experience for this concrete and illustrative data set. It shows how we start at the large RDF data set that results from the semantic integration and that is fit for the desired recommendations, and how we then turn this conceptual data set into a concrete system that uses currently available technology.

4. Data Preparation

A first issue to consider is the way in which the data is obtained in the application. In a realistic setting, data sources typically reside in different (physical) locations and are described in different (non-compatible) formats and schemas. In general, even when compatible with Semantic Web languages and standards, these raw data sources first need to be harmonized in order to be able to evaluate complex source-transgressing queries. A first step to harvest the decentralized knowledge is thus a data preparation phase, in which the data from the different semantic data sources are prepared for unified processing. The following data preparation steps were performed for the iFanzy sources:

- Data availability: The data is made available from the original source. In the ideal case, the original source is in Semantic Web format (i.e. RDF(S) / OWL), and offers direct possibility for remote querying (e.g. using a remote query service such as the Sesame[12] HTTP server or Virtuoso SPARQL Query Service[13]). In general, as was the case for iFanzy sources, remote sources do not offer this functionality. Therefore, data import is the most viable solution. In some cases the data is readily available (e.g. WordNet in RDF/OWL can be downloaded as a zip file[14]), in other cases, screen scrapers were needed to retrieve the data (e.g. we originally developed a scraper for the IMDb website). Note that the preferred or needed solution differs depending on whether one wants to have the entire data set available or wants to be able to query the data set with individual queries.
- Data conversion: Once the data is retrieved, we need to convert it into the correct RDF/OWL format. This conversion naturally depends on the initial format. In the case of WordNet the format was already RDF/OWL, in other cases a transformation was required. The TV-Anytime XML format retrieved from BBC Backstage or provided by the crawlers was transformed into our RDF/OWL format using an XSLT transformation. For IMDb, transforming the available plain text files to our RDF/OWL graph proved to be more challenging. In total there are 49 text files, each containing all the data on a certain domain (e.g. actors, producers, movies, countries, genres, ratings, quotes…). The files are of varying sizes, some quite large (e.g. the actors file is up to 360 MB). Every 'object' (a movie, an actor…) in those files has a unique identifier which is used throughout the other files. Parsers were written to parse each different file, exporting the required RDF/OWL format; a labor-intensive job.

[12] http://www.openrdf.org/

[13] http://docs.openlinksw.com/virtuoso/

[14] http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html

- Data integration: Once the data is transformed in RDF/OWL format, the different sources need to be integrated. This may require transforming the names and structure for certain concepts or properties, matching of resources from different sources and eliminating duplicate classes representing the same real-world concepts. For example, for the enrichment of the TV metadata (that can come from different online TV guides in different languages) we linked movie programs to the IMDb metadata, taking into account title data as well as AKA-titles and director information. For the XML grabber programs all time fields (startTime, endTime, duration…) are exposed as XML datetime/duration types, which we convert to a Time Ontology instance.

When we compare this to current examples of applications that use semantic content, we observe that most of them have such a data preparation phase. Considering for example some other representative applications of the last Semantic Web Challenge15, we also recognize the steps described above. GroupMe! [2] is a Web 2.0 style application that enables users to search for items from various sources (Google, Flickr, etc) and allows creating and tagging of groups of such resources, in addition to tagging the individual resources. GroupMe! starts by transforming any existing (structured) descriptions from the sources to RDF data, using ontologies that are consistent with the type of the resource. For example, Flickr-specific descriptions are copied into a well-defined RDF description using the Dublin Core16 vocabulary. Revyu [8] allows users to review and rate arbitrary items, providing a description, tags and URIs where related information can be found. Additional information is subsequently derived from various data sources, such as DBPedia or from the supplied URIs. In the latter case, the linked HTML pages are scraped (data import), the relevant data is converted to RDF format (data conversion) and subsequently integrated in the local semantic data store (data integration). For example, when an item is tagged as a book, the HTML pages provided with the description are scraped. When an ISBN number is found, an rdf:type property is added to the data store stating that the item is indeed a book.

5. Decomposition in Sources and Querying

As we motivated in the previous section, a first step in harvesting the strength of the Semantic Web lies in making different data sources compatible and combining them. While feasible for smaller-scaled projects, we will show in section 5.1 that this approach fails when working with real-life (huge) data sets. Therefore, a logical and necessary step to increase performance was to decompose the main data set into several smaller sets, thus making them practically more manageable by existing RDF storage and querying frameworks (e.g. Sesame and Jena[17]). The main method of operation is to consider which smaller parts of the data are queried regularly together: splitting them off in a smaller store with an increased performance for these queries, while maintaining the possibility to link and combine query results at the global level, can lead to an increase of performance for the overall system.

Drawing from work in relational databases (see e.g. [11]), such decompositions can be performed in two ways. Vertical, *property-based* decomposition in databases is based on the schema; instances related to certain classes and properties are split off from the data set. We will discuss this in section 5.2. Horizontal, *instance-based* decomposition is based on the resources: instances that are related in some way (e.g. via geographical similarity or specific knowledge about the typical queries), are split off from the data set. We will discuss that in section 5.3. All the experiments described in this section are performed using Sesame as an RDF storage and querying framework and SeRQL[18] as its associated and representative query language.

5.1 A single data set

As we saw earlier, real-life data sources can be huge. WordNet consists of almost 2 million triples; the IMDb data set we worked with contained over 53 million triples. Consequently, combining several of these sources, thereby linking data to be able to infer additional knowledge, induces serious performance considerations. During our experiments we noticed that in this setting, using realistic queries and data sources, performance and scalability indeed became critical issues. We also saw, for example when discussing with tool and technology providers, that these scalability problems in Semantic Web-based applications did not yet get the necessary attention to solve them in general. This might be in part because many such applications are created in a research setting, where examples are used that avoid the typical scalability and performance

15 http://challenge.semanticweb.org/

16 http://dublincore.org/

17 http://jena.sourceforge.net

18 Its current implementation in Sesame is more mature than that of the SPARQL W3C recommended query language.

problems that arise when employing large RDF data sets.

The straightforward and most common approach, which is successfully applied in such smaller-scaled projects, is to combine all data, including instance data and schema data, in one single data source. Applying this approach to the case of the iFanzy data lead to the following huge data set (table 1).

Table 1. Combined iFanzy data set

Data Source	#Triples	# Items
IMDb +BBC Backstage +XMLTV +Ontologies	54 525 724	# Persons: 1 653 543 # Movies: 976 174 # Locations: 19 946 # Time: 1 547 # Genres: 685 # Programs: 25 466

To test the feasibility of this first scenario for iFanzy, we executed the following four typical iFanzy queries with increasing time-complexity[19]:

- Query1: All programs with the genre 'drama' (or one of its subgenres).
- Query2: All programs with the genre 'drama' and the keyword 'wood' in the program metadata (title, synopsis and keywords)
- Query3: All programs with the keyword 'wood' in the program metadata (title, synopsis and keywords)
- Query4: All programs with the genre 'drama' and the keyword 'wood' in the program metadata or the person metadata (person name)

As for all experiments reported in this paper, each query was executed several times and the average execution times are reported. We also made sure that there was no caching effect that could influence our results. All queries were executed on Fedora Core 4 Linux system with 1 GB of main memory and a Intel(R) Pentium(R) 4 CPU running at 3.00GHz[20]. The average execution times for this experiment are given in table 2.

Although combining the different data sets was a necessary and beneficial step from a point of view of increasing the available knowledge, as we can see from the results below, applications employing huge data sets such as iFanzy suffer considerable performance problems. Even for the simplest queries, execution times quickly exceed the acceptable thresholds for real-time environments.

Table 2. Average execution times of typical iFanzy queries

Query	Average execution time (ms)
Query1	31 526
Query2	138 354
Query3	224 663
Query4	19 280 121

5.2 Vertical Decompositions

As already explained in the introduction, one way of decomposing data sets is by applying vertical decomposition: based on the "schema", certain classes and properties, together with the instances that go with them, are split off from the data set. Do note that in semantic databases like Sesame, there is no way to define 'views' like commonly used in relational databases. So querying a part of a triple store can be done either by adding extra restrictions in the where-clause or to physically divide one store into two new stores. In case of large data sets the second option can be preferred because semantic databases tend to get very slow when large. We applied this idea to split the data source in a set of smaller data sources, grouping data that conceptually belongs together. This grouping was based on the expected queries and how they access the data. For example, iFanzy uses the time ontology for time based reasoning. Therefore, it was decided to keep the time ontology separate, which means that only one (smaller) data set has to be queried when this time information is needed. Obviously, splitting off data sets has consequences for the queries: they need to be (transparently) rewritten and split up, fired to the partial data sets, and their results combined. Because vertical decomposition possibly distributes several properties of a class over different data sets, the original query has to be split up roughly along the same distribution. This entails identifying which properties reside in which data set, and subsequently isolating these properties (and the conditions acting upon these properties) in a single (partial) query. However, we deal with a tradeoff here. More decomposition might contribute to smaller repositories and therefore higher performance in those repositories, but the application's complexity will rise as well as will the overhead time to join results from the different sources.

To combine the results from such partial queries, the relation between these queries first has to be considered. More specifically, this means examining the path expressions used in the FROM clause. Figure

[19] Note that we do not display the actual SeRQL queries here because of their length.

[20] Note for the database systems we used multi core CPU systems hardly bring performance gain

1a and 1b illustrate the four general possibilities and their corresponding split. As can be seen from the figures, path expressions addressing a common subject or object (i.e. specified by a shared variable in the partial queries) give rise to partial queries which are dependent on each other: the results of one partial query influence the result of the other partial query.

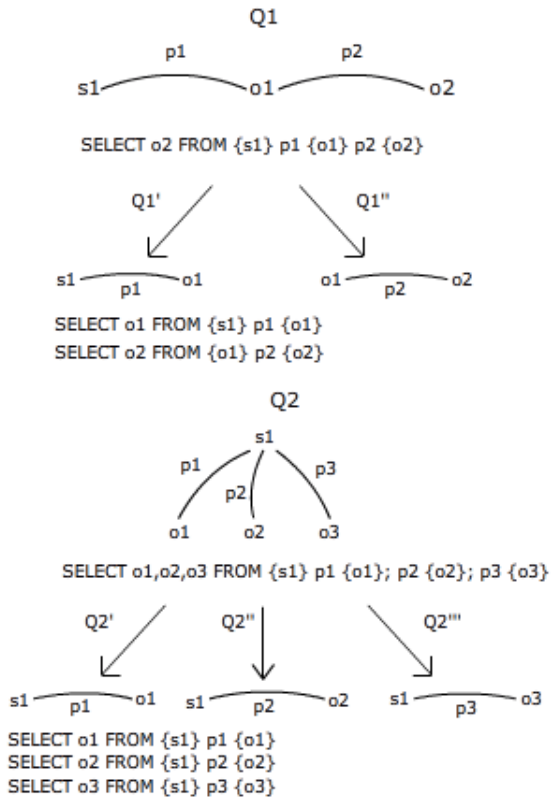

Figure 1a. Possibilities for (SeRQL) query decomposition

For example, in figure 1a Q1 is split up into Q1' and Q1'', using o1 as a shared variable; Q2', Q2'' and Q2''' share variable s1; in figure 1b, Q3', Q3'', Q3''' share variable o1. Different strategies to execute these partial queries exist. The most straightforward way is by performing a join on the result sets[21], using the shared variable(s) as the join attribute(s) and the constraints regarding the shared variables as the join condition. For example, in figure 1a, for Q1' and Q1'' o1 is used as a join attribute. These join attributes are included in the SELECT clause of the partial queries so that they can be compared afterwards using the join condition. Any (other) constraints on the shared variables are subsequently eliminated from the queries, as they were already fulfilled in the partial queries.

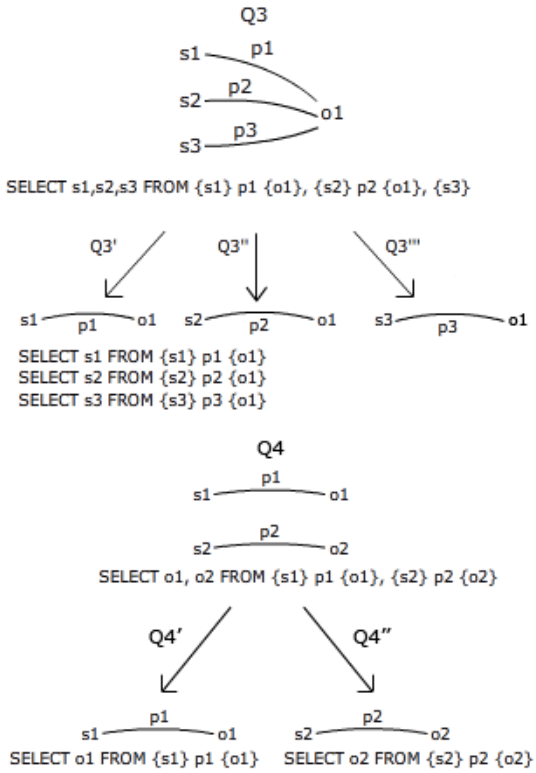

Figure 1b. Possibilities for (SeRQL) query decomposition

As we noticed in iFanzy, in some cases smarter and more efficient strategies can be employed to combine the result sets. For example, if it is known beforehand that one of the result sets from a partial query will be significantly smaller, say a handful, it is more efficient to include the results of this set directly into the query to the other data set, thus additionally constraining it and significantly reducing the execution time of this query. This approach gives rise to a query pipeline (and thus a sequential process), where the results from the previous step are used as input for the next step.

We exploited this approach in our use of WordNet together with the rest of the main program dataset. Instead of executing one query that searches for a program with terms that are connected to WordNet concepts that have certain restrictions, we first query WordNet to retrieve synonyms of input terms and then we use the results of that query as input for our query to the program data sources.

There are two ways in which this last part can be implemented: either by including all of the results of WordNet in the program query at once, or to execute the program query for every synonym. We

[21] Note that this join is thus not performed by Sesame, but by custom Java code.

experimented on our program data set to see what the differences for those two approaches are. The results can be found in table 3.

Table 3. Average execution times for two alternative query approaches

Query	Execution time (ms)
1 query 5 synonyms	3 597
5 queries 1 synonym	14 412

As can be seen from the results, the first approach can be favored over the second. This is because for each program resource the entire data source has to be traversed to make the necessary checks: when doing all these checks at once for each program resource (as is the case in the first approach) this traversal only has to be done once for each, while in the second approach all the program resources are checked (and therefore the graph traversed) every time a query is executed. However, when only requiring a limited amount of results, the execution of only one such query can be sufficient. For example, in the above experiment the query for the first synonym already returned 73 results. Such cases could greatly influence the average execution time of the second approach. In this particular case, the average execution time could be reduced by a factor of 5 (since only one of the five queries needs to be executed). In iFanzy, the first approach was used to combine the result sets from the program data sources and WordNet.

The case of combining separate result sets (see Q4 in figure 1b) consists of partial queries that are independent of each other, and therefore their result sets are independent as well. In this case, the partial queries can be parallelized and the results combined with a union. Sometimes, the assumption was that it was smarter to keep some data sets together. The BBC Backstage and XMLTV data sets, both representing broadcast information and transformed into the same RDF concepts, were kept in one single data set because of their conceptual relation and because, as a consequence, it was reasonable to assume that this data would mostly be needed together.

Another example of vertical decomposition is the separation of the genre classification hierarchy. In iFanzy, users can select a genre and thus limit the broadcast shows (or movies) to the ones conforming to the specified genre. Based on the fact that we frequently required this information separately and retrieving the genre information from the full data set took a very long time, we decided to extract the genre hierarchy from this data set and put it in a separate store. After this first series of decompositions, we obtained the collection of data sets as shown in table 4.

Table 4. iFanzy data set after first decompositions

Data Source	# Triples	# Items
IMDb data set	53 268 369	# Persons: 1 653 543 # Movies: 976 174 # Locations: 19 946
WordNet	1 942 887	# Words: 78 761 # Synsets: 104 433
BBC Backstage + XMLTV	1 250 949	# Programs: 25 466 on 137 channels
TV-Anytime Genres	4 859	# Genres: 685

Analogous to the separation of the genre hierarchy, the location hierarchy was split off from the IMDb data set. This was done for the same reasons: it was frequently needed in separation, and retrieving location data from the IMDb data set took a very long time.

5.3 Horizontal Decompositions

When we considered the querying of the data set containing broadcast data (i.e. BBC Backstage and XMLTV) we observed that this presented a performance bottleneck (which was the opposite of the previous assumption). We opted for a horizontal decomposition, which split the broadcast data into two separate data sets. As was already mentioned before, horizontal decomposition decomposes a data set based on the relations between the resources; based on e.g. geographical similarity or popularity, resources are split off and put in separate data sets.

For this particular decomposition, we used our knowledge of the original data sources and the queries posed on them, and split up the instances accordingly. This is advantageous when only requiring a limited amount of results (which is often the case in iFanzy), since in that case it suffices to query only one (smaller) data set, providing it contains the desired amount of results. Note that this is not the case for vertical decomposition, because resource properties can be spread over several data sets, thus requiring to query all the data sets containing (part of) the information.

As was the case for vertical decomposition, a horizontal decomposition has consequences for the query execution process. In case of horizontally decomposed data sets, where data belonging to the same "schema" can be distributed across different data sets, the same (partial) query has to be sent to the relevant data sets. Therefore, dependency issues between partial queries do not have to be considered.

However, another problem arises: because instances belonging to (a certain part of) the schema cannot be localized to one particular data set, it cannot be determined (without a priori knowledge) where specific information can be found. The strategies tackling this problem differ in the manner they deal with this uncertainty. The simplest strategy queries every (decomposed) data set until enough results are found. For example, in iFanzy this strategy is applied when retrieving broadcast data from BBC Backstage and XMLTV. If possible, knowledge about (the results returned by) the queries can be used to give priority to some data sets that are most probable to contain answers. In iFanzy, this strategy is used to retrieve information from IMDb when priority is given to a data set containing popular movies (as will be explained later). A third strategy uses an indexing technique to compute this probability at runtime; in other words, it enables the identification of data sets that are more likely to contain results given a specific query. In iFanzy, the latter strategy has not yet been applied; this is future work.

We set up some experiments to test this particular horizontal decomposition of broadcast data. A set of typical large queries was first sent to the combined data sets, and subsequently to the split data sets. This was repeated several times, and the average execution times are given in the table below:

Table 5. Execution times of separate and combined XMLTV and BBC Backstage data set

XMLTV (ms)	BBC Backstage (ms)	BBC Backstage + XMLTV (ms)
2142	551	2714

From the figures, it can be concluded that the query to the decomposed data sets executes faster than the query on the combined data set. As already mentioned, since only a limited amount of results is needed, querying only one data set is sufficient when it returns the desired amount of results. However, even when the desired amount of results was not obtained from one data set, still a performance gain can be made, since the queries to both data sets can be performed in parallel. In other words, due to parallelization of the queries, the total execution time is equal to the execution time of the slowest data set[22]. However, because of the use of Ajax technology we are able to show results the moment they become available, thus for us latency is more important then total computation time. In this case the global latency equals the latency of the fastest set.

The aforementioned decompositions gave rise to the data sets as seen in table 6. Compared to the previous data sets (see table 4), the BBC set and the XMLTV set are now separated, and the genre and location hierarchy have been split off from the main data set.

Table 6. iFanzy data set after further decompositions

Data Source	# Triples	# Items
IMDb data set	53 208 444	# Persons: 1 653 543 # Movies: 976 174
WordNet	1 942 887	# Words: 78 761 # Synsets: 104 343
BBC Backstage	83 871	# Programs: 1 565 on 8 channels
XMLTV	1 167 078	# Programs: 23 901 on 129 channels
Geo	59 925	# Locations: 19 946
TV-Anytime Genres	4 859	# Genres: 685

We also observed that queries to the IMDb data set were too slow for real-time query answering. Based on studying the representative queries to be executed, and the queries actually performed by customers, we concluded that mainly a small popular subset of the movies is frequently requested. Therefore, we chose to perform a horizontal decomposition based on the popularity of the movies. As a criterion, in accordance with the desired application functionality, we used the votes that were issued by the IMDb users. Based on a stepwise restriction of the amount of movies via their popularity (see first two columns in table 7), we compared the amount of user queries still answerable by this subset with the average query response time for typical queries to the IMDb data set (third column), and decided to split off movies that have 500 or more votes from the main IMDb data set and put them in a separate data set.

Since this IMDb subset only contains a fraction of the total amount of movies, this data set is not always sufficient to answer each user query. In that case, the main IMDb data set, which we still keep available, is consulted. Evidently, every time we need to query this (full) IMDb data set, we will surrender any performance gain made by employing the split off data set. However, because this IMDb subset contains the movies that are most requested (i.e. the popularity of the movies), we are guaranteed that in most cases this data set will suffice.

[22] Since the combination of the result sets is a union, this computation overhead is negligible.

Note that these optimization steps lead to some space overhead because of some overlap in the split repositories (i.e. the connections). However this space overhead is no significant drawback given the speed increase. The initial database where all content resides in one big semantic repository comprised 7,9 GB, whereas the sum of the final decomposed set of repositories takes up 8,4 GB.

Table 7. IMDb size and query execution times

Minimum # of Votes	# Movies	Query execution time (ms)
0	976 174	54 198
1	261 749	14 832
10	141 438	8 137
25	82 996	4 322
100	33 386	1 805
500	11 500	678
1000	7 173	487

6. Reasoning Optimization

Next to the concept of linked data, another important strength of the Semantic Web is its possibility to reason over facts. As the reasoning capabilities of RDF played a significant role in the choice for using semantic technology, implementing efficient reasoning into iFanzy was considered an important step in its development, and a necessary one to provide for a realistic performance. For example, the following (custom) entailment rules represent the transitivity of the partOf relationship:

<X, loc:partOf, Y>
<Y, loc:partOf, Z>
} ⇒ <X, loc:partOf, Z>

<X, imdb:filmingLocation, Y>
<Y, loc:partOf, Z>
} ⇒ <X, imdb:filmingLocation, Z>

Two strategies can be considered when applying custom inferencing rules to an RDF data set. The first one consists of storing the closure of these rules in the data set, thus minimizing the run-time cost of inferencing (not taking into account the performance loss related to recalculating the closure after the addition of new data or when the data set has been updated). The second strategy consists of computing such inference rules at run-time, by translating them into the query and/or by using custom code.

To decide for which set we calculate the closure in advance and which can we do on the fly, we need to inspect the data closely. Let us first consider the location hierarchy by means of the location "shepperton studios" in the UK:

<"shepperton studios", part of, "shepperton">
<"shepperton", part of, "england">
<"england", part of, "uk">

If we calculate the closure in advance, every program *P* which is annotated with "shepperton studios" will also be annotated with "shepperton", "England" and "UK". If the user now requests all programs annotated with "UK", we also retrieve all programs *P*. However, if the closure for the locations hierarchy is not pre-calculated, every *P* will only be annotated with "shepperton studios", and thus, to obtain the same result set, the inference-logic needs to be included in the query, which in practice leads to a huge WHERE clause explicitly enumerating all the locations. In table 8 we see the difference in average execution times for a location in "USA". In the first column a pre-calculated closure is used; in the second column, all 8877 relevant locations are included in the WHERE clause.

Table 8. Average execution times for queries with pre-calculated closure vs. manual reasoning (location)

Pre-calculated closure (ms)	Query with manual reasoning (ms)
2 375	140 818

Because the performance clearly suffers from the extremely large WHERE clause, it is straightforward to assume that in this case pre-calculating the closure (for the location hierarchy) helps considerably.

In case of the genre hierarchy however, we see a different story. When we compare the execution times for the pre-calculated closure and the reasoning included in the query (10 genres), we obtained the following results (see table 9).

Table 9. Average execution times for queries with pre-calculated closure vs. manual reasoning (genre)

Pre-calculated closure (ms)	Query with manual reasoning (ms)
3 352	3 375

As can be seen from table 9, the difference in execution time between the two approaches is negligible, as opposed to the execution times in table 8. This can be explained by the difference in amount of results returned from the reasoning process (i.e. 8877 locations in the first example vs. 10 genres in the second example). We can thus conclude that explicitly

storing the pre-calculated closure in the data set is worthwhile when the closure is relatively large.

7. Applying Available Tools and Technologies

To further improve query execution times, additional optimizations were applied. In particular, existing tools and technologies were evaluated for their usefulness: the use of relational databases, keyword indexing, use of limited queries and improvement to Sesame, the RDF storage and querying framework used in iFanzy.

7.1. Use of Relational Databases to Store RDF Data

Where the use of semantic data models offers great possibilities for linking data, current software for storing and manipulating semantic RDF data has its problems when it comes to performance for data sets, such as the one from real-world Web applications like iFanzy. One way to recover some of this performance loss is to store well-structured (strongly structured) parts of such a large data set in a relational database. It should be noted that many RDF data architectures like Sesame already allow the use of a relational database to store RDF data. However, such relational backends are typically very generic, and cannot be configured to store a given part of the RDF data in a specific way. Also, they do not allow for specific optimization techniques to improve query evaluation (e.g. indexing, de-normalization).

Exploiting our knowledge of the strongly structured IMDb data set, we therefore devised an optimized relational database matching this specific structure. The links to other RDF data (kept in an RDF data set) were maintained by using resource URIs in the relational database, and referring to these URIs in the RDF stores (and vice versa).

Table 10. Comparison of execution times between RDF data with and without relational database optimization

Query	IMDb response time (ms)	IMDb with relational database optimization (ms)
Query 1	2 157	117
Query 2	1 135	417
Query 3	11 265	3 252
Query 4	648	12
Query 5	14 344	3 495

As can be seen from table 10, the performance gain that was achieved from this migration was significant.

7.2. Use of Keyword Indices

Indices in relational databases are data structures that are constructed to decrease access time to certain parts of the database. Analysis of the running iFanzy application showed that the free text search (available to the user in the form of a string search searching through all properties of programs like title, synopsis, person names, etc) is problematic when working with large data sets, since Sesame's pattern matching facility has a performance linear to the size of the data set. As Sesame does not provide indexing support for terms and literals, we decided to build our own index specifically to speed up full text search queries. We did so using a relational database. For every program (both broadcast and movie) resource, we extracted the literal objects of the properties title, keywords, synopsis and associated people like actors, directors, presenters, etcetera. Subsequently, we parsed these literals (except for the associated people) using white space as a delimiter and filtered them using a stopword filter, retaining only the useful keywords. Every resource and keyword pair was then put in a relational database table containing an index on the keywords. Every search query issued by the user is first filtered using a stopword filter, and then (together with synonyms obtained from WordNet) sent to the MySQL database. Consequently, the result set of this query is a list of program URIs. The other constraints specified by the user are sent to the relevant Sesame data sets, also resulting in a list of resource URIs. The intersection of these two lists is afterwards returned as the complete result set.

To test the performance gain, we executed a series of free text search queries over the IMDb data set. This lead to a spectacular performance increase for free text search queries, as can be seen in table 11.

Table 11. Average execution times for free text search queries with and without index

Without index (ms)	With index (ms)
57 726 731	3 193

7.3. Use of Limit for Optimization

Studying the representative queries in the application, we saw that a major optimization could be obtained by the use of limit and offset operators in queries (to the relational databases and Sesame repositories). Indeed, in most cases users do not inspect the whole result set, but only the first few result pages

(in the case of iFanzy, a result page contains 20 results). By using a limit clause, the repository/database will be searched for matches until the number of results as specified in the limit clause is found. This can have a significant influence on performance, as the first *X* results may be found early on in the query execution process, while the whole data set needs to be inspected in order to obtain a complete result set. If the user requests the next page of results an offset is used: the first *Y* of matching results will be discarded by the database and the next *X* number of matching results will be returned. This results in re-evaluating the query with a limit of *X*. Therefore, subsequent result pages will be more expensive to calculate. However, as in our experience users typically do not navigate beyond the first couple of results pages this in general does not reduce the overall performance of our system. The test results below illustrate the performance gain by including a limit clause (with increasing limit operands) in a typical iFanzy query executed on the IMDb data set:

Table 12. Average execution times for queries with increasing limit operand

Limit	Average execution time (ms)
10	595
100	628
1000	834
No limit	1 665

7.4. Sesame Versions

During the development of iFanzy, in cooperation with the creators of Sesame, we started using Sesame 2 when it was still in its alpha stages. One of the reasons the development of Sesame 2 was started were the limitations of the internal representations of the RDF model, which lead to limited query optimization possibilities and therefore limited scalability. Sesame 2 was designed to eliminate these limitations.

However, using alpha software also had its disadvantages. Besides frequent API changes that forced us to recode database calls and connections, the scalability of Sesame 2 proved problematic. While Sesame 2 does not have the same limitations as Sesame 1, Sesame 1 had been greatly optimized during the many years since its introduction, while the query optimization of Sesame 2 is still (onto this date) in its early stages. Because of this, Sesame 2 generally performed worse in practice than Sesame 1. The table below shows the average execution times of several typical iFanzy queries for the broadcast data set:

Table 13. Sesame 1 vs. Sesame 2

Query	Sesame 1 (ms)	Sesame 2 (ms)
Query 1	621	1 776
Query 2	1 599	2 256
Query 3	1 909	3 798

As can be seen from the results, performance dropped significantly when using Sesame 2. Therefore, we decided to revert most of the data sets (namely the BBC Backstage and XMLTV data sets) back to Sesame 1. On the other hand, Sesame 2 did have some useful features (e.g. the context mechanism) that are not available in Sesame 1; the data sets that were populated depending on these features, most notably context, were left in Sesame 2 (e.g. WordNet and the different ontologies such as genres and location). We do foresee a migration back to Sesame 2, when the performance of Sesame 2 will exceed that of Sesame 1.

8. Conclusions and Future Work

One of the greatest opportunities in using techniques from the Semantic Web in the engineering of Web applications is the possibility to link data, thereby increasing the total amount of knowledge that was available in the separate sources. Building real-life Web applications grounded on such huge sets of linked data however is far from trivial with the currently available technology and tools. In this article, we discussed the engineering of such large-scale and real-life Web applications, with the focus on practical implementation strategies that make the applications work with the available technology. In summary, with all the data semantically linked and thus ready to be accessed, for any RDF/OWL-based application it remains a challenge to turn the (huge) single complete conceptual knowledge structure into parts that can be handled separately by the tools for data management and access. For this aim, we studied both vertical and horizontal decompositions of semantic data sets, discussed reasoning, and considered optimizations based on existing tools and technologies. The results from our research that we illustrated here were obtained in the context of representative data, and we have shown and substantiated with practical experiments that, using specific engineering steps and practical measures to increase performance, such real-life applications are feasible.

Where we have presented here a general recipe for engineering large-scale Web applications that exploit semantic linking while dealing with the inherent implementation and engineering challenges, we obviously will further develop and test the lessons learned from this experience. First of all by extending

them where possible (e.g. we plan a performance optimization step with parallel query evaluation and automatic load-balancing strategies). Second, by performing further experiments to expand the basis for conclusions, which includes applying and evaluating them in even more real-life scenarios. New applications with different constraints may of course give rise to new techniques that can be applied, and thus lead to an extension of the spectrum of solutions we can offer. As an example, we mention the use of OWLIM[23] [9] in iFanzy, as it promises to improve query evaluation performance considerably. It will be interesting to see how this backend performs in the iFanzy setting, and what is needed in terms of engineering to turn the general possibilities of OWLIM into a concrete and specific advantage for this application.

9. References

[1] D.J. Abadi, A. Marcus, S. Madden, K.J. Hollenbach, "Scalable Semantic Web Data Management Using Vertical Partitioning", *Proceedings of the 33rd International Conference on Very Large Data Bases (DBLP)*, ACM, Vienna, Austria, (2007), pp. 411-422.

[2] F. Abel, M. Frank, N. Henze, D. Krause, D. Plappert, P. Siehndel, "GroupMe! – Where Semantic Web meets Web 2.0", *Proceedings of the 6th International Semantic Web Conference*, LNCS 4825, Springer, Busan, Korea, (2007), pp. 871-878.

[3] L. Ardissono, C. Gena, P. Torasso, F. Bellifemine, A. Chiarotto, A. Difino, B. Negro, "Architecture of a system for the generation of personalized Electronic Program Guides", *In Proceedings of the UM2001 Workshop on Personalization in Future TV*, Sonthofen, Germany, (2001).

[4] L. Aroyo, N. Stash, Y. Wang, P. Gorgels, L. Rutledge, "CHIP Demonstrator: Semantics-Driven Recommendations and Museum Tour Generation", *Proceedings of the 6th International Semantic Web Conference*, LNCS 4825, Springer, Busan, Korea, (2007), pp. 879-886.

[5] P. Bellekens, L. Aroyo, G.J. Houben, A. Kaptein, K. van der Sluijs, "Semantics-Based Framework for Personalized Access to TV Content: The iFanzy Use Case", *Proceedings of the 6th International Semantic Web Conference*, LNCS 4825, Springer, Busan, Korea (2007), pp. 887-894.

[6] P. Bellekens, K. van der Sluijs, L. Aroyo, G.J. Houben, "Engineering Semantic-Based Interactive Multi-device Web Application", *Proceedings of the 7th International Conference on Web Engineering* (ICWE 2007), LCNS 4607, Springer, Como, Italy, (2007), pp. 328-342.

[7] Y. Blanco-Fernández, J.J. Pazos-Arias, A. Gil-Solla, M. Ramos-Cabrer, M. López-Nores, "Bringing together content-based methods, collaborative filtering and semantic inference to improve personalized TV", *Proceedings of the 4th European Conference on Interactive TV* (EuroITV), Athens, Greece, (2006), pp. 174-182.

[8] T. Heath, E. Motta, "Revyu.com: A Reviewing and Rating Site for the Web of Data", *Proceedings of the 6th International Semantic Web Conference*, LNCS 4825, Springer, Busan, Korea, (2007), pp. 895-902.

[9] A. Kiryakov, D. Ognyanov, D. Manov, "OWLIM – a Pragmatic Semantic Repository for OWL", *Proceedings of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005*, LNCS 3807, Springer-Verlag, New York City, USA (2005), pp. 182-192.

[10] D. O'Sullivan, B. Smyth, D.C. Wilson, K. Mcdonald, A. Smeaton, "Improving the Quality of the Personalized Electronic Program Guide", *User Modeling and User-Adapted Interaction* 14, 1, Kluwer Academic Publishers, (2004), pp. 5-36.

[11] G. Ramakrishnan, J. Gehrke, *Database Management Systems (International Edition)*, McGraw-Hill Inc., ISBN 0-07-246563-8 (1999).

[12] K. Rohloff, M. Dean, I. Emmons, D. Ryder, J. Sumner, "An Evaluation of Triple-Store Technologies for Large Data Stores", *In On the Move to Meaningful Internet Systems: OTM 2007 Workshops*, LNCS 4806, Springer, Vilamoura, Portugal, (2007), pp. 1105-1114.

[13] J. Seidenberg, A. Rector, "Web ontology segmentation: analysis, classification and use", *In Proceedings of the 15th international Conference on World Wide Web* (WWW 2006), ACM, Edinburgh, Scotland, (2006), pp. 13-22.

[14] M. van Setten, "Supporting People In Finding Information: Hybrid Recommender Systems and Goal-Based Structuring", *Telematica Instituut Fundamental Research Series, No.016 (TI/FRS/016),* Universal Press, (2005).

[15] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z.G. Ives, "DBpedia: A Nucleus for a Web of Open Data", *Proceedings of the 6th International Semantic Web Conference*, LNCS 4825, Springer, Busan, Korea, (2007), pp. 722-735.

[16] H. Stuckenschmidt, M. Klein, "Structure-Based Partitioning of Large Concept Hierarchies", *Proceedings of the Third International Semantic Web Conference* (ISWC 2004), LNCS 3298, Springer, Hiroshima, Japan, (2004), pp. 289-303.

[23] http://www.ontotext.com/owlim/big/index.html