Lesson Learnt from a Large-Scale Industrial Semantic Web Application

Sylvia C Wong, Richard M Crowder, Gary B Wills, and Nigel R Shadbolt
School of Electronics and Computer Science
University of Southampton, UK
sw2@ecs.soton.ac.uk, rmc@ecs.soton.ac.uk, gbw@ecs.soton.ac.uk,
nrs@ecs.soton.ac.uk

ABSTRACT

The design and maintenance of an aero-engine generates a significant amount of documentation. When designing new engines, engineers must obtain knowledge gained from maintenance of existing engines to identify possible areas of concern. We developed a Semantic Web based document repository for transferring front-line maintenance knowledge to design. The Semantic Web is an ideal candidate for this application because of the size and distributed nature of an aerospace manufacturer's operation. The Semantic Web allows us to dynamically cross reference documents with the use of an ontology. However, during the design and implementation of this project, we found deficiencies in the W3C[1] recommended Semantic Web query language SPARQL. It is difficult to answer questions our users sought from the document repository using SPARQL. The problem is that SPARQL is designed for handling textual queries. In industrial applications, many common textual and semantic questions also contain a numerical element, be it data summarization or arithmetic operations. In this paper, we generalize the problems we found with SPARQL, and extend it to cover web applications in non-aerospace domains. Based on this analysis, we recommend that SQL-styled grouping, aggregation and variable operations be added to SPARQL, as they are necessary for industrial applications of the Semantic Web. At the moment, to answer the non-textual questions we identified with an RDF store, custom written software is needed to process the results returned by SPARQL. We incorporated the suggested numerical functionalities from SQL for an example query, and achieved a 21.7% improvement to the speed of execution. More importantly, we eliminate the need of extra processing in software, and thus make it easier and quicker to develop Semantic Web applications.

[1]World Wide Web Consortium

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
HT'07, September 10–12, 2007, Manchester, United Kingdom.
Copyright 2007 ACM 978-1-59593-820-6/07/0009 ...$5.00.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query languages*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; J.2 [**Computer Applications**]: Physical Sciences and Engineering—*Engineering*

General Terms

Languages

Keywords

Semantic Web, Industrial Hypermedia, SPARQL, SQL, RDF

1. INTRODUCTION

The design and maintenance of large and complex engineering systems requires a significant amount of documentation, particularly if the system being considered is a turbofan engine used on the current generation of aircraft. These engines are amongst the most complex machine ever designed, incorporating a wide range technologies including high temperature materials, complex fluid dynamics and high speed rotating components.

A fundamental shift is currently occurring in the aerospace industry away from selling products to providing services. Companies such as Rolls-Royce aim to make an increasing number of its engine fleet subject to long-term maintenance service agreements [10]. Essential to the success of this market shift is to design new products with lower and more predictable maintenance costs. To minimize maintenance costs throughout the engine's life cycle, engineers must obtain knowledge gained from maintenance histories of similar products during the design phase of new products. This will help engineers identify parts most likely to be problematic throughout the engine's entire life cycle. It should be noted that engine design is typically undertaken by a number of teams who are responsible for individual engine modules, e.g compressor or turbine. Therefore it is impossible for any single member of a design team to access more than a fraction of the available documentation. As is widely recognized, information systems usually develop over time into a set of heterogeneous resources with ill-defined metadata. As a result, it becomes difficult for engineers to follow a trail through the resources [23]. The challenge for organizations is therefore to develop an information system that is both comprehensive and will satisfy the increasing demands from industry for up-to-date and easily accessible information.

In response to these challenges, we are developing a Semantic Web based document repository to support engineers to design for the aftermarket [24]. Semantic Web technologies are used for this project because of their ability to easily integrate distributed resources. In the aerospace industry, maintenance documents are created by service teams located all over the world, at airports and overhaul facilities. Design documents are created by multiple design teams, which can also be based at multiple sites. During the design and implementation of this document repository, it became apparent that deficiencies in the W3C recommended query language SPARQL [19] makes extracting required data from RDF [16] stores unnecessarily difficult. Specifically, SPARQL lacks support for aggregation, grouping and SELECTed variable operations. These functionalities are essential to industrial knowledge applications, where the information sought is not entirely textual in nature, but often contains a numerical element.

The contribution of this paper is threefold:

- To argue that in industrial applications, not all knowledge sought are entirely and completely textual. Even if the data stored is textual and the questions asked relate to semantic information stored within the data, some degree of numerical and data summarization is often required.
- To demonstrate that SPARQL cannot answer questions that require data summarization and simple arithmetic operations.
- To demonstrate that SQL-styled aggregation, grouping and variable operations are easy to implement in SPARQL, and that the addition of these features open SPARQL, and thus RDF stores, to industrial knowledge applications.

The paper is organized as follows. Section 2 gives an overview of industrial hypermedia applications, and its relevance to the Semantic Web. Section 3 then reviews other literature that compares Semantic Web technologies with traditional relational databases. Section 4 explains the motivation behind our document repository project, and the objectives we are trying to achieve. Section 5 describes the documents available and the ontology developed for our application. This is followed by Section 6, which describes the type of knowledge design engineers specifically sought from our document repository. Section 7 then explains in detail why SPARQL is ill-equipped to handle the questions asked by the engineers in our application. A comparative implementation in SQL is presented in Section 8. Section 9 generalizes the problem with SPARQL to other industrial knowledge application domains. This section outlines the categories of questions that SPARQL is ill-equipped to answer. Then in Section 10, we explain why these shortcomings can be easily solved with technologies today's Semantic Web infrastructure are built upon. The paper finishes with conclusions in Section 11.

2. INDUSTRIAL HYPERMEDIA AND THE SEMANTIC WEB

The concept of using open hypermedia for information and document management within manufacturing organizations was first proposed by Malcolm et. al. [15]. They argued that hypermedia systems had to evolve beyond the standalone application and allow the integration of resources across an enterprise. Since then, a number of hypermedia applications including engineering applications [2], medical record keeping [12], historical archives [5] and education [4].

The aim of the Semantic Web is to augment the existing Web into a web of data [20]. This contrasts to the original Web where information is provided and delivered to be read by humans, and not for automated processing by software agents. RDF (Resource Description Framework) is the standard for information representation and knowledge exchange in the Semantic Web. RDF is designed to allow easy integration over distributed sources of information. It allows anyone to make statements about any resource.

While the Semantic Web aims primarily at providing a generic infrastructure for machine-processable content on the Web, it has direct relevance to hypermedia [22, 1]. In particular, the Semantic Web provides an open-standard infrastructure that are compliant to the definition of open hypermedia systems defined in [3] and [6]:

- A system that does not impose any mark-up upon the information. The absence of mark-up ensures that other processes that are not part of the information delivery system can access the information.
- the links are separated from the information objects.
- A system that can integrate with any tool or process that runs under the host operating system.
- A system in which information and processes maybe distributed across a network and across hardware platform.

3. QUERYING THE SEMANTIC WEB

There are several query languages for RDF in existence [8], for example, RDQL, RQL and SPARQL. SPARQL has emerged as the de-facto RDF query language, and currently has W3C Recommendation status. RDF query languages offer developers a simple way to combine the results of queries over disparate resources. Documents on corporate intranet and web sites are mostly stored inside relational databases and presented to users via server side scripts. Therefore the use of RDF for data exchange can be seen as a way to unleash the isolated data stored inside these distributed relational databases, and integrating these databases into a unified resource for sharing.

Siberski et. al. had also recommended an extension to the SPARQL language [21]. The current SPARQL specification offers only limited facilities for scoring and result ordering. Specifically, it is not possible to search with a list of scored preferences. For example, a user wants to search for a list of dentists, preferring ones close to home, and with appointments outside rush hours. However, if preferences are combined with a logical `AND`, the search is likely to return an empty result table if the list of preferences is long. The work of Siberski et. al. allows scoring of preferences in a search. In the dentist example, the user would be able to say that finding a dentist with appointments outside rush hours is more important than finding ones close to home.

Drawing parallels between relational databases and RDF stores, or SQL and SPARQL is nothing new. For example, Noy and Klein discussed the similarities and differences

between database schema and ontology evolutions [18]. By comparing and contrasting database schemas and ontologies, Noy and Klein built on and transferred knowledge from existing research in schema evolution and applied them to ontologies. Similarly, the purpose of this paper is not to dismiss SPARQL. It is to draw useful features from SQL and show how they can be used to improve SPARQL for industrial applications of the Semantic Web.

MacRae-Spencer and Shadbolt compared the query languages SQL and SPARQL for a large-scale digital library indexing application [17]. The aim of the reported comparison was to investigate whether RDF/SPARQL was better (or at least equal) in performance with a repository built upon relational database and SQL. They found that updates and inserts took longer in relational databases because of the need to recalculate the indexes that were needed for efficient table JOIN operations. On the other hand, JOINs were not needed in RDF stores because the data model is designed specifically for questions of relationships and integration of disparate sources of information. MacRae-Spencer and Shadbolt noted that one of their sample queries *"how many distinct authors are there in this system"* was more efficient in SQL than SPARQL. This was because the SPARQL statement had to query the entire RDF store to find all instances that matched the search pattern and then formed a distinct list of the matches. Summation was then calculated with an external program. However, they did not provide any suggestions for overcoming this deficiency. In this paper, we provide a broader analysis of the shortcomings of SPARQL with regard to a large-scale industrial application. We generalize the type of user questions that are hard to answer in SPARQL. We then make suggestions on how SPARQL can be modified to support these user questions.

4. MOTIVATION

As is well recognized in engineering design, the use of past experiences and previously acquired knowledge, either from the designer's own experiences or from resources within their organization forms an important part of the design process. It has been estimated that 90% of industrial design activity is based on variant design [7], while during a redesign activity up to 70% of the information is taken from previous solutions [14]. A cursory consideration of these figures identifies two immediate challenges – how to capture knowledge, and how to retrieve it. The purpose of our document repository is thus to enable the transfer and retrieval of knowledge across the organization to support design activities.

Figure 1 shows the key information flow for the different stages in the life of an engine. Concept design is the first stage of an engine's life cycle. Given a set of broad requirements, such as thrust, range of the target aircraft and fuel burn, engineers determine the approximate dimensions, weight, power and other physical characteristics for the engine design. The engineers also make estimates of the manufacturing costs of the engine. In the design stage, engineers transform the preliminary abstract design into a set of concrete plans that can be used in production. In production, engines are built according to the design plans. Traditionally, after production and sales, responsibility for the engine passes from the manufacturer, to the airlines, who own the engines. The airlines are responsible for maintaining the engines. This maintenance activity is supported by the manufacturer's technical support and operations team. To assist maintenance engineers to identify problems before a breakdown occurs, engines are commonly equipped with sensors for engine monitoring. This monitoring information can be analysed for abnormal operating conditions, such as temperature or pressure. However until fairly recently, the monitoring data was only used to support maintenance activities, even though it is a rich source of information for the designers of future engines.

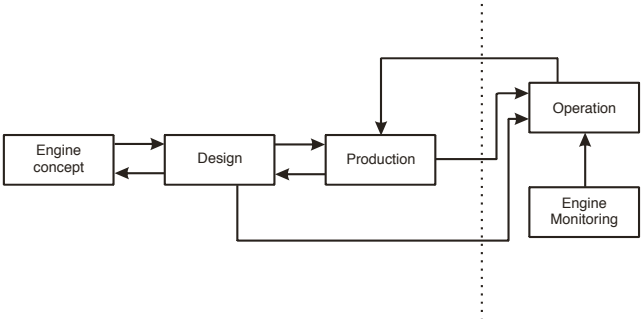

Figure 1: Information flow between the different stages in the life of an aero-engine. The vertical line between production and operation represents the transfer of the engine from its manufacturer to an airline. Operations is the generic term for maintenance and aftersale support.

As can be seen in Figure 1, there are interactions and information flow between neighbouring stages in the production maintenance process. This is due to the iterative nature of engineering processes. Design knowledge is also passed to operations in the form of 'owner's manuals'. Sometimes, information also passes between unconnected stages, for example, between operations and engine concept. However, the flow is weak and may take the form of informal and personal networking between engineers.

While the process works very well, it does have significant disadvantages. In particular, design engineers are remote from the problems experienced in the field by operations. Due to the importance of increasing operational reliability and minimizing maintenance costs in the new market paradigm of product support, information gained in the operation of a fleet of engines needs to be fed back to the designers of subsequent engines. However, the current information infrastructure makes this difficult as concept design engineers do not have access to maintenance knowledge. Similarly, design engineers should consult existing maintenance documents to help design parts with more predictable maintenance costs.

As a result, we need to strengthen and help formalize the information flow between the company's aftermarket operations and the design teams. Figure 2 shows the information and knowledge flows our research aims to build. This would allow the knowledge gained during the design, production and operation of an engine to advise the design of the next variant.

The following scenario[2] illustrates the potential use and benefit from such a document repository. The scenario involves three separate and different groups of users that are

[2]The scenario is entirely fictitious and does not derive from any real event.

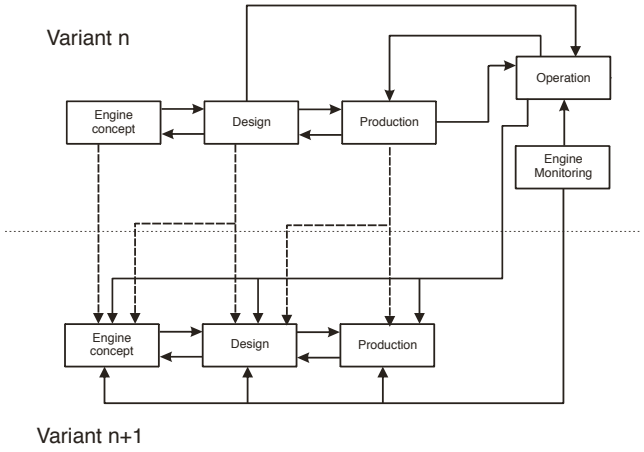

Figure 2: We aim to facilitate the flow of information gained during the life-cycle of one engine variant to inform the design of the next variant. The dotted arrows indicate the flow of design rationale and similar knowledge. The solid arrows represent all other information flows including real time engine information and design documentation.

involved in the life of a jet engine. Front-line maintenance engineers are involved in the day to day servicing of the engines, and thus responsible for populating the document repository with maintenance reports and other similar documents.

> *During the regular pre-flight checks, a flight crew reported a problem with a leak from an engine's bleed air system. Subsequent inspection which required the removal of the engine revealed that a duct had failed at a joint due to vibration. After repair, the engine was returned to service, and a full maintenance event report submitted to the document repository.*

The document repository can then be used by technical support and operation engineers, who are responsible for improving the performance of existing engines. They can use information collected in the repository to monitor trends that develop over a fleet of engines. Modification can then be designed to mitigate any problems found:

> *Following a review of the maintenance events relating to a specific engine fleet, a trend was noticed in the high than expected number of failure of an air duct joint due vibration. To maintain the reliability of the engine fleet, a modification was developed and implemented.*

The same information in the repository will also be used by design engineers working on a new engine:

> *The design team for the next variant of this engine reviews the performance of the air bleed system across the fleet to learn from previous design rationale and operational history. Finite element analysis showed that a joint failure could occur due to vibration if certain operational conditions were met. It was therefore decided that the future variant of the engine would both eliminate the joint and reroute the duct work. The revised design costs 50% more than the original. However, the saving over the life of the engine will be substantial due to lower likelihood of in-service failure.*

The goal of our work can thus be summarized as follows: To feedback and harvest knowledge gained from the aftermarket operations documents to help (a) operations engineers in designing modifications to existing engines, and (b) design engineers in designing the next variant engine for the aftermarket.

5. DOCUMENTS AND ONTOLOGY

For each maintenance event, maintenance engineers document information surrounding the event for later reference. A maintenance event occurs whenever actions are performed on an engine. Usually, each maintenance event involves multiple actions. Information documented includes the circumstances of the event, actions taken, parts installed and removed and any other findings they observed. The engineers who created these documents are located in numerous sites around the world, in the manufacturer's or third party repair bases or at airports. Most of the maintenance engineering documents reside in a centralized Service Data Manager (SDM).

To enable machines to interpret meanings stored within the documents, we need an ontology that captures all the terms and concepts used. Moreover, since the document repository is to be used by both design and service engineers, the ontology should capture concepts from engineers working in both areas. The ontology is created by analyzing existing documents and conducting knowledge acquisition interviews with engineers [23]. The engineers interviewed are carefully selected and are domain experts from several different specialization. The result of these interviews enabled us to identify, by specialism, the main concepts and the associated keyword for these concepts used by the particular type of engineer when searching for information.

The resulting ontology contains concepts ranging from engine deterioration mechanisms, engine models and parts to airport locations. Figure 3 shows a UML class diagram for the concepts associated with maintenance events in the application ontology. In the diagram, the square boxes are concepts, or *classes*, in the ontology. Underneath the name of the class is a list of attributes for the class. In the ontology, a UML attribute is modeled as an OWL property which has the specified class as its domain. The lines connecting two classes in the diagram are associations. Associations represent relationships between classes. The arrow on the line shows the direction of the association. A UML association is modeled with an OWL property which has the linked classes as its domain and range. In addition, the *Component* class contains a taxonomy of aero-engine parts, and the *Engine* class includes a list of exiting engine types. In total, the ontology has 896 classes and 133 properties. Most of the classes represent parts within the engine taxonomy.

We populated this ontology with maintenance records from the SDM database. The maintenance records used are for the Trent 500 engine over a period of 5 years. There are ap-

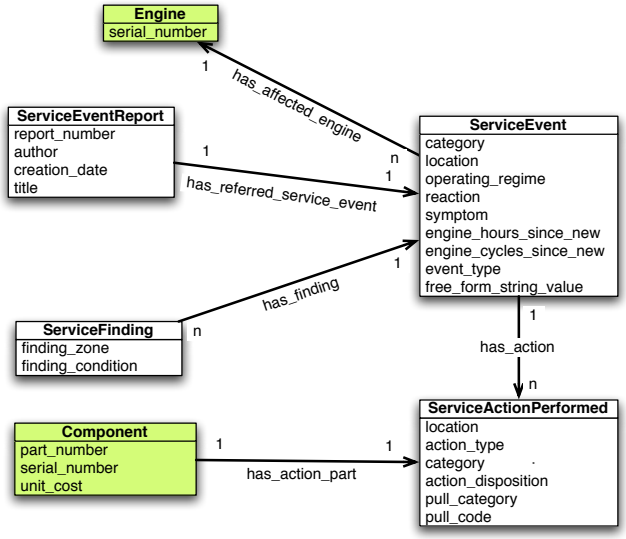

Figure 3: UML class diagram for the ontology used to describe maintenance events. This is a simplified view and does not show all properties and classes defined in the ontology.

proximately 3250 maintenance events, with around 31,000 actions. The populated RDF store currently contains 316,240 ground triples and 213,330 inferred triples.

6. USER REQUIREMENTS

To understand the scope of knowledge our users want to gain from an engine's service history, a questionnaire-based study was conducted with design engineers from Rolls-Royce [13]. In the questionnaire, engineers were presented a list of questions relating to maintenance experience with a product. They were asked how often they might ask them when designing a new product. They were also asked what other questions they might want to ask. The result of this questionnaire tells us what are the most important and most common life cycle information design engineers seek from maintenance documents.

The study identified 39 questions commonly asked by design engineers. Only a small number of the 39 questions involve complex mathematical and numerical simulations, which cannot be answered from semantic analysis of maintenance documents. Most of the questions are concerned with textual and semantic information stored within the documents. However, to answer even these textual based questions requires some degree of simple arithmetical manipulation, as demonstrated by the following examples:

1. What are the common deterioration mechanisms associated with this part?
2. Are there any other mechanisms, which only occur rarely?
3. How many engine removals have been caused by a deterioration of this part?
4. Which parts dominate the reliability and cost drivers in this engine?
5. How critical is it if and when this part fails?

These five questions can be broadly classified as belonging to two categories – *ranking* and *scoring*. To answer questions 1 to 3, we need to first find the occurrence of a pattern within the documents. Afterwards, we rank the occurrences to obtain a list in ascending or descending order. Specifically, in a SPARQL query, the pattern is a RDF triple, or a series of RDF triples. We say that questions 1 to 3 is a *ranking problem*. Question 4 contains two sub-questions, sorting parts that are serviced most by frequency and by cost. Obtaining a list of parts by service frequency is a *ranking problem*. Sorting by cost requires the multiplication of unit costs of parts with their service frequencies. We say that it is a *scoring problem* in addition of being a *ranking* one. For question 5, criticality is measured by the reaction taken by front-line maintenance engineers. Each reaction is assigned a criticality score. Therefore, question 5 is also a *scoring problem*.

In the next section, we explain in detail how to answer question 4 using SPARQL. Question 4 is used as an example because it contains both *ranking* and *scoring*. Then in Section 8, we present a comparative implementation of the same question using SQL. The purpose of the comparison is to demonstrate how *ranking* and *scoring* problems are impossible to answer using SPARQL alone, but extremely simple with SQL.

7. SPARQL IMPLEMENTATION

When asking question 4, the engineer wants a sorted list of parts ordered by (1) service frequency, and (2) total accumulated cost of maintenance. As input, the question requires the engineer to select an engine model that interests him or her. Inside the maintenance documents, engine models are recorded using their sub-model numbers. For example, the Trent 500 engine has two sub-models, the Trent 553 and the 556. Therefore, we made the Trent 553 and 556 subclasses of the Trent 500 in the ontology. Users can use the term Trent 500 to search for documents on both sub-models. Similarly, within the maintenance documents, parts are only referred to using their part numbers, such as ET10935. It should be noted that parts with different part numbers can be identical in functionality. To generate results meaningful to the engineers, we combine information from both design and maintenance and associate part numbers with their types. By correctly instantiating the parts involved in the *Component* hierarchy of the ontology, we can group functionally identical parts together within a query. Using the ontology, we can return a list of parts based on functional groupings, and not simply distinct part numbers.

The first step in answering part 1 of the question is to obtain a complete list of part installation and removal using the SPARQL query shown in Figure 4. This query looks for all triples that contains the `has_action_part` predicate, and obtains the functional type of the part used in that action.

The result returned by this query is simply a complete list of parts that are named in all service actions. There is no grouping, and if a part is named in multiple service actions, it will appear multiple times in the result set, as shown in Figure 5. The number of results returned depends on the number of service actions stored inside the target triplestore. With 31,000 actions stored, the result table returned will have 31,000 rows.

```
PREFIX ipas:<http://www.3worlds.org/ipas.owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ep:<http://www.3worlds.org/parts#>

SELECT ?type
WHERE
{
  ?a ipas:has_action_part ?part .
  ?part a ?type .
  ?type rdfs:subClassOf ep:Trent500_Components
}
```

Figure 4: SPARQL query to obtain a complete list of part installation and removal.

part
http://www.3worlds.org/parts#Blade_Assy
http://www.3worlds.org/parts#Blade_Assy
http://www.3worlds.org/parts#Blade_Hp_Turbine
http://www.3worlds.org/parts#Blade_Assy
http://www.3worlds.org/parts#Sav
http://www.3worlds.org/parts#Eec
http://www.3worlds.org/events.owl#Blade_Hp_Turbine
http://www.3worlds.org/parts#Ignition_Unit
http://www.3worlds.org/parts#Blade_Rotor
http://www.3worlds.org/parts#Sav

Figure 5: First 10 rows of the result table returned from a SPARQL query for obtaining a complete list of part installations and removals (Figure 4).

After obtaining the list of part installation and removal shown in Figure 5, we count the number of occurrences of each part within the list. This gives us a list of occurrences, as shown in Figure 6. Sorting this table in decreasing order according to the occurrence column gives us the list of most serviced part. The counting and sorting cannot be done using SPARQL and must be performed with custom software written in a language such as Java.

The second part of the question asks for a list of parts sorted by accumulated cost of maintenance. To achieve this, we issue a SPARQL query for each part that appears in the occurrence table. If there are 1000 distinct parts in the occurrence table, 1000 queries are issued. The query in Figure 7 retrieves the cost of a `Blade_Assy`. We then multiply the unit cost for each part with the number of occurrences. The result of the multiplication is the total accumulated cost of service for the part. Lastly, we sort the part list in descending order according to the total cost. This gives us a

part	occurrence
http://.../parts#Sav	138
http://.../parts#Blade_Assy	1081
http://.../parts#Blade_Rotor	12
http://.../parts#Blade_Hp_Turbine	169
http://.../parts#Eec	97

Figure 6: A list of the total number of times a part has been serviced. This is obtained by counting the number of occurrences of a part in the list in Figure 5 using a custom program.

```
SELECT ?c
WHERE
{
  <http://www.3worlds.org/parts#Blade_Assy>
       ipas:has_unit_cost ?c
}
```

Figure 7: SPARQL query to obtain the unit cost of a part.

list of parts that cost most in servicing. The iteration of the result table, the multiplication, the summation and the sorting are all performed using external software.

8. COMPARISON WITH SQL

For the comparative SQL implementation, we use a simple mapping from ontology to relational database schema. We create a database table for each class in the UML class diagram in Figure 3. Attributes for each class are mapped as column headings in the corresponding table. Associations are mapped to primary and foreign keys.

However, with a relational database, no ontology is used to add semantic meanings to terms and concepts within the documents. Therefore a SQL query searching for the term Trent 500 would not automatically expand the query to include the sub-models of the engine. Similarly, we cannot directly query engine parts using their functional names, as they do not exist in the original documents. We could extend the database tables to include design information such as sub-model and part functional names. We can then perform JOINs on these design tables with the maintenance record tables. However, this is not implemented here because the purpose of this section is to demonstrate the grouping, aggregation and variable operations of SQL. We believe introducing JOINs in the examples will only confuse the issue.

To find the list of parts ordered by service frequency, we use the SQL query in Figure 8. This single SQL query finds

```
SELECT action_part,
       COUNT(action_part) AS occurrence
FROM ServiceActionPerformed
GROUP BY action_part
ORDER BY occurrence DESC
```

Figure 8: SQL query to obtain the list of parts ordered by service frequency.

all the parts referenced in service actions, counts the number of occurrences of each part, and sorts the result by the number of occurrences. Unlike the SPARQL implementation, no additional code is needed to group, count and sort the result returned by the query engine. The reason behind the difference is because of data grouping and aggregation support in SQL. In this query, we use the GROUP BY command to group all entries with the same `action_part` together. We then apply the COUNT aggregate function to count the number of occurrences for each `action_part`. The result is then sorted by occurrence using the ORDER BY command.

SPARQL also supports the ORDER BY command. However, you cannot group the result and count the number of occurrences within SPARQL. As a result, the variable we want to ORDER BY is not available within SPARQL.

Therefore, external software must be used to sort the result.

To find the list of parts sorted by total cost of servicing, we use the SQL query shown in Figure 9. The query for ob-

```
SELECT c.action_part,
     (c.action_part * c.unit_cost) AS cost
FROM Cost c INNER JOIN
  (SELECT action_part,
       COUNT(action_part) AS occurrence
     FROM ServiceActionPerformed
     GROUP BY action_part) o
   ON c.action_part = o.action_part
ORDER BY cost DESC
```

Figure 9: SQL query to obtain the list of parts ordered by cost of servicing.

taining the number of occurrences for each part (Figure 8) is repeated here as a nested inner query. The only difference is that the nested query does not sort the result using an ORDER BY. This is because we are interested in sorting the result by cost, not by occurrence. The inner query provided us with a list of part occurrence, similar to the one in Figure 6. This occurrence table is then joined with the `Cost` table. We calculate the total cost of servicing each part by multiplying the number of occurrences for each part by its unit cost. The result is sorted from the most costly to the least using ORDER BY.

SQL supports arithmetic operations on SELECTed variables. We use this feature in the second query to multiply the number of occurrences with the unit cost for each part found. In comparison, this is not possible in SPARQL. The multiplication must be performed with external software written for this purpose. It should be noted that SPARQL does support arithmetic operations, but only in the FILTER command.

9. IMPLICATIONS

Without the basic numerical support as described in the previous section, it is difficult for SPARQL to answer common knowledge questions about the data beyond simple textual inquiries. These basic numerical questions also commonly appear outside the application of maintenance experience transfer in aerospace manufacturers. As a result, this problem has a much wider and general impact on the suitability of SPARQL, and thus RDF triplestores, in knowledge systems.

Aggregate functions in SQL operate against a collection of values, but return a single value. They are very useful for data summarization. Besides the COUNT function used in the example earlier, SQL has other useful aggregate functions such as MAX and AVG. The following is a list of aggregate functions and the type of questions they answer:

- **COUNT** - *"How many X exists?"* For example, in a restaurant recommendation application, the user might be interested to know how many restaurants in Southampton have a five-star rating. Or for a digital library, we might want to know how many authors there are in the system.
- **MAX/MIN** - *"What is the most/least Y of X?"* For example, what is most or least expensive three bedroom house listed for sale in Southampton?
- **AVG** - *"What is the average Y of X?"* For example, what is the average user rating for this movie?
- **SUM** - *"What is the sum of X?"* For example, what is the total value of non-fiction books sold this week?

Without GROUP BY, the aggregate functions return the appropriate aggregate of all column values in the result table. By using GROUP BY, we obtain the aggregate for each individual group instead. Most of the user questions raised by design engineers in our target application requires both grouping and aggregation. We called this type of question *ranking* problems in Section 6. Ranking problems usually manifest themselves as:

- *"How many of X in each Y?"* For example, how many researches work in each research area in the University of Southampton? Or what type of restaurant can we find in Southampton and what is the average price for each type?
- *"Give me a list of N most/least Y of X?"* By adding a ORDER BY to the grouping function, we can sort the output according to the aggregated values. For example, a list of the top 10 most published researchers or highest user rated movie.

The aggregate functions summarize data in a column in the result table. However, there are times when we want to combine values stored in multiple columns in the same row. In other words, we want to be able to apply operations on multiple SELECTed variables. *Scoring* problems identified in Section 6 require this type of operation. For example, in the question *"which part dominates the cost drivers in this engine"*, we need to multiply the unit cost by the number of occurrences. In a restaurant review website, a user might be asked to rate restaurants according to several aspects, such as food, price, ambience and service. To calculate the overall rating for a restaurant, we will need to sum the separate ratings together.

Since SPARQL does not support grouping, aggregating and variable operations, extra processing in software code is needed to answer the type of questions discussed. First, an extra layer of processing with custom written software code introduces an extra source of error. It is harder to debug a long processing chain in software than a simple, single query. Not to mention that writing software takes time and effort. Second, operations performed by a database is much faster than iterating through result sets in programming lanugages such as Java. This is because the database is optimized to executing query code. As a result, if something can be performed in SQL, it should not be performed using Java [11].

Table 1 shows the speed of execution of the SPARQL and SQL implementations for the first part of question 4 in Section 6, *"which parts dominate the reliability drivers in this engine"*. The experiment is run 10 times for both implementations. A Linux machine with an Intel Xeon 3.00 GHz CPU and 1 GB of RAM is used. For the SPARQL implementation, we used 3store [9] as our RDF store. 3store is a fast, efficient RDF store written in C designed for large datasets. Persistence in 3store is backed by the MySQL relational database. Sorting in Java is performed using `Collections.sort()` in the J2SE standard library. For the SQL implementation, we used the same MySQL installation and issued SQL queries in Java using JDBC. The result for the

Table 1: Time (in milliseconds) to answer the question *"which parts dominate the reliability drivers in this engine?"*

	SPARQL	SQL
Mean	3512.5	16.4
Std. dev.	177.1	9.2

SPARQL implementation is well clustered around the mean, with the standard deviation only 5% of the mean value. For the SQL implementation, the standard deviation is over 50% of the mean. This is because the execution time is so short that the result is highly dependent on the variation in operation system scheduling and processor load during the time of execution.

10. RECOMMENDATIONS

RDF stores commonly use relational databases for persistent storage, for example, Jena[3], Sesame[4] and 3store. Therefore, SPARQL commands are internally translated to SQL commands by these semantic web toolkits to query the underlying relational database. As a result, it is relatively straightforward to add existing SQL functionalities to an implementation of SPARQL. Due to this close coupling between SPARQL and SQL, and the importance of these features to industrial applications, we suggest that grouping, aggregation and variable operations be added to the specification of SPARQL.

Lets take the example of grouping and aggregation. For the SPARQL query in Figure 4, 3store translates it to the SQL query in Figure 10. Both queries have one SELECT

```
SELECT v0.lexical AS 'type'
FROM (SELECT t1.object AS '_TV0',
      t2.object AS 'type',
      t0.subject AS 'action',
      t0.object AS 'part'
  FROM triples t0, triples t1,
       triples t2, triples t3
  WHERE t3.object=-995897154303950976
  AND t3.predicate=-5275523445234180533
  AND t3.subject=t2.object
  AND t2.predicate=-5275523445234180533
  AND t2.subject=t1.object
  AND t1.predicate=-4085280037482734459
  AND t1.subject=t0.object
  AND t0.predicate=-5286098249812210380
   ) AS 'tmp0_60ff', symbols v0
WHERE tmp0_60ff.type=v0.hash
```

Figure 10: SQL created by 3store to query the underlying relational database from the SPARQL in Figure 4.

variable `type`. This variable matches to all serviced parts in the database using the criteria specified. In the SPARQL query, the matching criteria is specified within the WHERE clause. This WHERE clause is translated by 3store to the SQL statements after the initial SELECT line.

We need to count the rows in the result table, and order the table according to the number of occurrences for each

[3] `http://jena.sourceforge.net`

[4] `http://www.openrdf.org`

Table 2: Time (in milliseconds) to answer the question *"which parts dominate the reliability drivers in this engine?"*

	SPARQL with Grouping and Aggregation
Mean	2751.3
Std. dev.	72.5

`type`, similar to the query in Figure 8. Figure 11 shows a SPARQL query that supports SQL style grouping and aggregation. The suggested syntax borrows heavily from the original SQL syntax. This SPARQL query can be implemented in SQL as shown in Figure 12.

```
SELECT ?type, COUNT(?type) AS ?occurrence
WHERE
{
  ?a ipas:has_action_part ?part .
  ?part a ?type .
  ?type rdfs:subClassOf ep:Trent500_Components
}
GROUP BY ?type
ORDER BY ?occurrence DESC
```

Figure 11: Suggested SPARQL syntax for grouping and aggregation.

```
SELECT v0.lexical AS 'type',
       COUNT(type) AS occurrence
FROM (SELECT t1.object AS '_TV0',
 ...
 ...
WHERE tmp0_60ff.type=v0.hash
GROUP BY type
ORDER BY occurrence DESC
```

Figure 12: Translation of suggested SPARQL query in Figure 11 to SQL.

We implemented the query in Figure 12 using Java with JDBC and ran the query 10 times over a MySQL relational database. The execution time for this query is shown in Table 2. Compared to counting, grouping and sorting the result table in Java, there is a 21.7% improvement on query speed. However, speed is not the reason why we recommend adding grouping and aggregation to SPARQL. By adding these functionalities that already exist in the underlying query language (SQL), it makes it possible for SPARQL and RDF stores to answer non-textual knowledge questions, without the need of additional processing.

To support arithmetic operations on SELECTed variables, a function that returns the lexical form of a literal is needed. This is because literals in RDF can contain an optional language or datatype tag:

- "chat"@fr
- "1.3"^^xsd:decimal

The query language SeRQL from Sesame provides the functions `label()`, `lang()` and `datatype()` for extracting the different parts of literals. `label()` returns the lexical form, ie

chat and 1.3 in the above examples. `lang()` and `datatype()` returns the language and datatype attributes respectively. The results of these functions can then be used in operations on untyped values. Figure 13 shows an example on how two columns in a result table are multiplied using the `label()` function.

```
SELECT ?type, (LABEL(?cost) * LABEL(?num))
WHERE
{
  ?a ipas:has_num_part_used ?num
  ?a ipas:has_action_part ?part .
  ?part ipas:has_unit_cost ?cost .
  ?part a ?type .
  ?type rdfs:subClassOf ep:Trent500_Components
}
```

Figure 13: Suggested SPARQL syntax for variable operations.

11. CONCLUSIONS

In this paper, we described the problems we found in a large-scale application of the Semantic Web for an industrial document repository. The repository is to be used within an aerospace manufacturer, to enable knowledge transfer from front-line maintenance to the design of new engines. The motivation behind this knowledge transfer is the fundamental shift that is occurring in the aerospace industry away from selling products to providing services. This shift in market focus means that new engines must be designed to provide lower and more predictable life cycle costs. To achieve this, engineers must obtain knowledge gained from the entire life of an engine.

Semantic Web technologies are selected for this application because of the ease of integrating distributed sources of information via a shared ontology. This is important to the aerospace industry because maintenance and design engineers are located in numerous different sites around the world. Additionally, semantic reasoning provided by an ontology allows us to find relevant documents that do not contain the words specified in the search, but are merely implied.

However, after we shared and integrated these distributed documents using RDF, it became apparent to us that SPARQL cannot answer the questions sought by our users. This is because SPARQL is designed to answer only textual queries. However, we found that even though our users' queries are concerned with textual and semantic information inside the documents, some degree of simple arithmetic manipulation is required to answer them. We generalized the problem found into other application domains, and summarized the types of questions that SPARQL is ill-equipped to handle.

To demonstrate the problem with SPARQL, we showed an example of the process needed to answer one of our users' questions. We showed that SQL-styled aggregation, grouping and variable operations are necessary to generate the results required. With our SPARQL implementation, we developed custom written software to perform these operations. We compared the speed of execution for the two implementations.

We argued that since most existing RDF stores are built upon relational databases, features available in SQL can easily be made available to SPARQL. We showed such an implementation of aggregation and grouping with a RDF store called 3store. We compared the speed of execution of this modified SPARQL implementation on a user query with the original SPARQL implementation, and found a speed improvement of 21.7%. More importantly, the additional features eliminate the need for extra processing in software, thus allowing for more rapid and error-free development of Semantic Web applications. We also made suggestions on additions to SPARQL needed to support variable operations.

12. ACKNOWLEDGMENTS

This research was undertaken as part of the IPAS project (DTI Grant `TP/2/IC/6/I/10292`). The authors would also like to thank the project partners for providing us with data and ontologies. Specifically, we would like to thank Derek Sleeman and David Fowler from the Department of Computing Science, University of Aberdeen for the ontology, and Alymer Johnson and Santosh Jagtap from the Engineering Design Centre, Cambridge University for the user requirement analysis.

13. REFERENCES

[1] L. Carr, W. Hall, S. Bechhofer, and C. Goble. Conceptual linking: Ontology-based open hypermedia. In *Proceedings of 10th International World Wide Web Conference (WWW)*, pages 334–342, Hong Kong, 2001.

[2] R. M. Crowder, W. Hall, I. Heath, and G. Wills. Industrial strength hypermedia: Design, implementation and application. *International Journal of Computer Integrated Manufacturing*, 13(3):173–186, 2000.

[3] H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins. Towards an integrated information environment with open hypermedia systems. In *ECHT '92: Proceedings of the ACM conference on Hypertext*, pages 181–190, New York, NY, USA, 1992. ACM Press.

[4] P. De Bra. Pros and cons of adaptive hypermedia in web-based education. *Cyberpsychology and Behavior*, 3(1):71–77, 2000.

[5] A. Dunlop, M. Papiani, and A. Hey. Providing access to a multimedia archive using the world wide web and an object-relational database management system. *IEE Computing and Control*, 7(5):221–226, 1996.

[6] A. M. Fountain, W. Hall, I. Heath, and H. C. Davis. *MICROCOSM: an open model for hypermedia with dynamic linking*, pages 298–311. Cambridge University Press, New York, NY, USA, 1992.

[7] Y. Gao, I. Zeid, and T. Bardasz. Characteristics of an effective design plan system to support reuse in case-based mechanical design. *Knowledge-Based Systems Knowledge-Based Systems Knowledge-Based Systems*, 10(6):337–350, Apr. 1998.

[8] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *Proceedings of 3rd International Semantic Web Conference (ISWC)*, pages 502–517. Springer, 2004.

[9] S. Harris and N. Gibbins. 3store: Efficient bulk RDF storage. In *Proceedings of 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–15, Sanibel Island, Florida, 2003.

[10] A. Harrison. Design for service – harmonising product design with a services strategy. In *Proceedings of GT2006, ASME Turbo Expo 2006: Power for Land, Sea and Air*, Barcelona, Spain, May 2006.

[11] C. S. Horstmann and G. Cornell. *Core Java 2 Advanced Features*, chapter 4 Database Connectivity: JDBC. Prentice Hall, 2002.

[12] H. H. S. Ip, K. C. K. Law, and S. L. Chan. An open framework for a multimedia medical document system (a multimedia document system framework). *Journal of Microcomput. Applications*, 18(3):215–232, 1995.

[13] S. Jagtap, A. Johnson, M. Aurisicchio, and K. Wallace. Pilot empirical study: Interviews with product designers and service engineers. Technical Report 140 CUED/C-EDC/TR140- March 2006, Engineering Design Centre, University of Cambridge, 2006.

[14] D. V. Khadilkar and L. A. Stauffer. An experimental evaluation of design information reuse during conceptual design. *Journal of Engineering Design*, 7(4):331–339, 1996.

[15] K. C. Malcolm, S. E. Poltrock, and D. Schuler. Industrial strength hypermedia: Requirements for a large engineering enterprise. In *Proceedings of the Third ACM Conference on Hypertext*, pages 13–24, Dec. 1991.

[16] F. Manola and E. Miller. RDF primer. Technical report, W3C Recommendation, `http://www.w3.org/TR/rdf-primer`, Feb. 2004.

[17] D. M. McRae-Spencer and N. R. Shadbolt. Semiometrics: Applying ontologies across large-scale digital libraries. In *Proceedings of Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, Athens, Georgia, USA, 2006.

[18] N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6(4):428–440, 2004.

[19] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Technical report, W3C Working Draft, `http://www.w3.org/TR/rdf-sparql-query`, Oct. 2006.

[20] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[21] W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In *Proceedings of 5th International Semantic Web Conference (ISWC)*, 2006.

[22] J. van Ossenbruggen, L. Hardman, and L. Rutledge. Hypermedia and the semantic web: A research agenda. *Journal of Digital Information*, 3(1), 2002.

[23] G. Wills, D. Fowler, D. Sleeman, R. Crowder, S. Kampa, L. Carr, and D. Knott. Issues in moving to a semantic web for a large corporation. In *Proceedings of 5th International Conference on Practical Aspects of Knowledge Management (PAKM)*, volume 3336 of *Lecture Notes in Artificial Intelligence*, pages 378–388. Springer, 2004.

[24] S. C. Wong, R. M. Crowder, G. B. Wills, and N. R. Shadbolt. Knowledge engineering - from front-line support to preliminary design. In D. F. Brailsford, editor, *Proceedings of ACM Symposium on Document Engineering (DocEng)*, pages 44–52, Amsterdam, The Netherlands, Oct. 2006.