Proceedings of IDETC/CIE 2009
International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
August 30-September 2, 2009, San Diego, USA

DETC2009-86544

NAVIGATION OVER A LARGE ONTOLOGY FOR INDUSTRIAL WEB APPLICATIONS

Richard M Crowder[*] **Max L Wilson, David Fowler, Nigel Shadbolt, Gary Wills and Sylvia Wong**
School of Electronics and Computer Science
University of Southampton
Southampton, UK
Email: rmc@ecs.soton.ac.uk

ABSTRACT

Ontologies for industrial semantic web applications are often very large. This is especially true in scientific and engineering applications where there exists a large pool of technical terminology necessary for operation within the domain. In this paper we look at the problem of presenting this domain ontology to users for navigation within web applications. The conventional tree view can be considered to be cumbersome and awkward to navigate for ontologies that have a very large breadth and/or depth. We present three approaches to this ontology presentation problem — content dependent filtering, autocompletion text box and partial segments using drop-down lists. All the approaches attempt to *limit* the ontology presented to users at one time. We implemented two of the proposed methods of ontology presentation in our demonstrators and have received positive and valuable feedback from engineers.

INTRODUCTION

One of the goals of the Semantic Web is to bring structure and meaning to Web pages, beyond the human-centric content that is prevalent on the Web today [1]. The Semantic Web is aimed at computational agents, so that *programs*, and not just humans, can interpret the meaning of documents on the Web. This allows the Web to be used for more than a human-browseable repository of information. Documents in the Semantic Web are expressed in RDF [2], which is a language for representing information about resources. An ontology, in RDFS [3] and/or OWL [4], then provides interpretation of terms and concepts found in RDF documents. The structure provided by an ontology forms the backbone of knowledge interpretation for Web applications.

The ontology needed for industrial and scientific web applications are usually, and some might argue inevitably, large. This is because of the size and complexity of the underlying domain knowledge. For example the Gene Ontology [5] for bioinformatics has over 25,000 terms. Of course, not all applications have to use the full ontology of the targeted domain [6]. Several approaches have been introduced to automatically reduce the size of ontologies for applications. The goal of ontology trimming is to reduce hosting cost and increase tractability. For example, Seidenberg and Rector analyzed links in ontology structures to extract relevant segments [7], while Alani et. al. performed simplification by analyzing queries used in targeted applications [8].

However, the size of the technical vocabulary captured in an ontology is often difficult to reduce. One might even argue that simplification is not viable as it might remove features that may be required. For example, in the aero-engine component ontology we have previously used for our document repository in Wong et al [9], there are 160 direct subclasses of *Part*. These 160 classes have a further 1096 direct subclasses. All these subclasses represent parts of an aero-engine, and must be made available to engineers using the web application. Furthermore, engine components are not the only domain concepts we require in our application.

In many Semantic Web applications, there is a need for users to navigate over the underlying ontology. When querying doc-

[*]Address all correspondence to this author.

uments, an application might wish to guide users in entering search terms from the ontology. When composing new documents, an application might capture ontological concepts entered by users to a document, or it might allow users to tag the document with ontology based keywords.

Perhaps the most widely acceptable approach to the ontology navigation problem is to present the user with a tree representation, similar to the one available in ontology editors, Figure 1. However, as the size of the ontology increases, it becomes more and more difficult to navigate with a tree view.

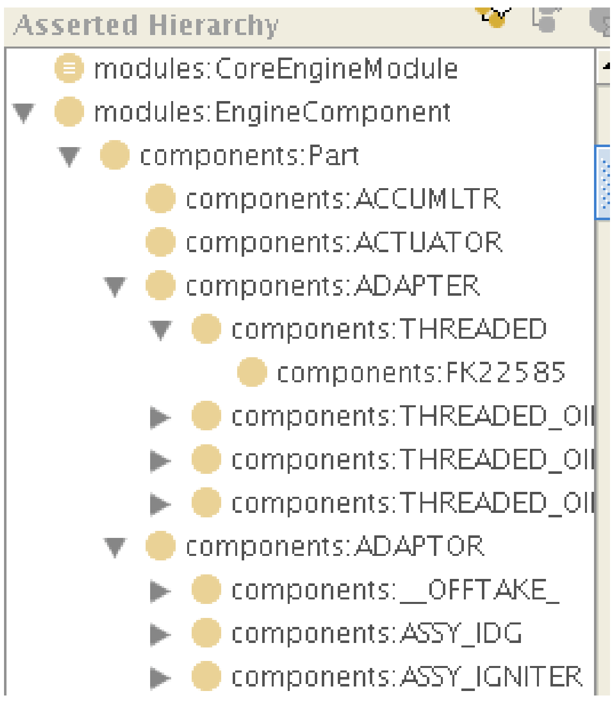

Asserted Hierarchy
modules:CoreEngineModule
modules:EngineComponent
components:Part
components:ACCUMLTR
components:ACTUATOR
components:ADAPTER
components:THREADED
components:FK22585
components:THREADED_OI
components:THREADED_OI
components:THREADED_OI
components:ADAPTOR
components:__OFFTAKE_
components:ASSY_IDG
components:ASSY_IGNITER

Figure 1. Tree view of class hierarchy in the Protege ontology editor.

After we have briefly described the target web application, this paper presents three approaches we investigated for navigation over a large ontology in an industrial application. Firstly we present a method to prune tree menus based on content stored in the triplestore. Secondly, instead of presenting a class hierarchy to users, we use text boxes with autocompletion. The autocompletion is done using an ontology of the target domain. Finally we presents an alternative way to display a class hierarchy using interconnected drop-down lists. Due to the operational nature of drop-down lists, only a partial hierarchy is presented to users at one time. Feedback from user evaluations, if available, is presented in the section of the relevant methodology.

THE DESIGNER DESKTOP

It is well recognized in engineering design, that 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 [10], while during a redesign activity up to 70% of the information is taken from previous solutions [11].

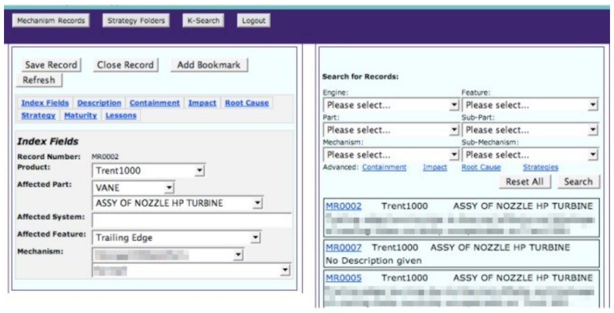
In [12] we considered future systems to support the engineering designer, where there tends to be a presumption that all processes should be based on IT systems; in our discussions with the designers this was strongly resisted. They are only willing to move to automated systems if there was a clear advantage in doing so. One of the key issues for them was for any system to be accurate and reliable. If these are not achieved, the user community will not trust the system, and hence not use it during their daily activities. The introduction of such systems is a major organizational issues. Currently the interactions in the design process are mainly human, and the use of technology would require a change in the normal ways of working. Our work can be seen as an extension on digital libraries. Digital libraries concentrates on the problem of searching for documents distributed over multiple repositories. For example, Priebe and Pernul [13] developed a portal over multiple document repositories by using an integrated metadata store. As a result, users can search on both the content of the documents and their metadata. In contrast, our approach is service based, and search functionality does not form part of our proposed infrastructure. However, global document searches can be provided to our knowledge repository via web services that implement document indexes and metadata indexes. In the scenario we only consider a subset of the available documents. Petrelli et al [14] identified over a dozen different textual documents across Rolls–Royce, any of which could be used to populate a record. As the documents are generated at different stages of a problems solving activity, they need to be cross referenced through the use of metadata.

In response to these challenges, we are developing a number of *Designer Desktop* to harvest and feed back knowledge gained from both previous design and service documents to help (a) engineers design modifications to existing engines, and (b) design engineers in designing the next variant engine, [9, 15].

Specifically, the *Designer Desktop* is a Semantic Web application that serves the following two main purposes:

1. Retrieve relevant existing design and maintenance documents
2. Assist in the creation of new semantically enriched documents

For our *Designer Desktop* demonstrator, we want to present the domain ontology to users so they can create queries using vocabulary from the ontology. In our discussions with the user community it was possible to identify four parameters they would commonly use to search for data. They are *component, engine, feature* and *mechanism*. Each of these parameters are represented by a class hierarchy in the ontology. For example, any subclass of `Part` is valid for the part parameter. Therefore, our task is to reduce or present relevant segments of the ontology to users, so they will not feel overwhelmed by the vastness of the full ontology. Figure 2 shows two screenshots of the current implementation of the system, these are primarily concerned with the development of the record. To ensure good user interaction the screen is divided into two, with the entering of search terms and the results being presented on the right, while data entry takes place on the left of the screen.



Index Fields

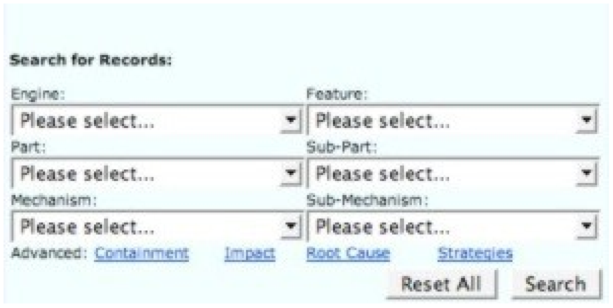
Record Number: MR0002
Product: Trent1000
Affected Part: VANE
ASSY OF NOZZLE HP TURBINE
Affected System:
Affected Feature: Trailing Edge
Mechanism: [redacted]

MR0002 Trent1000 ASSY OF NOZZLE HP TURBINE
[redacted]

MR0007 Trent1000 ASSY OF NOZZLE HP TURBINE
No Description given

MR0005 Trent1000 ASSY OF NOZZLE HP TURBINE
[redacted]

(a) The screen used when searching for a record.



(b) An expanded view of the boxes used to define the search terms.

Figure 2. The Designer Desktop developed during the project.

ONTOLOGY AND SYSTEM ARCHITECTURE

To enable machines to interpret meanings stored within the documents, an ontology that captures all the terms and concepts used was created. Moreover, since the document repository was to be used by engineers from across the company responsible for both design and service activities, the ontology captures concepts from engineers working in both areas. The ontology was created by analyzing existing documents and conducting knowledge acquisition interviews with engineers [16]. 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, [17]. Figure 3 shows a UML class diagram for the concepts associated with maintenance events in the application ontology.

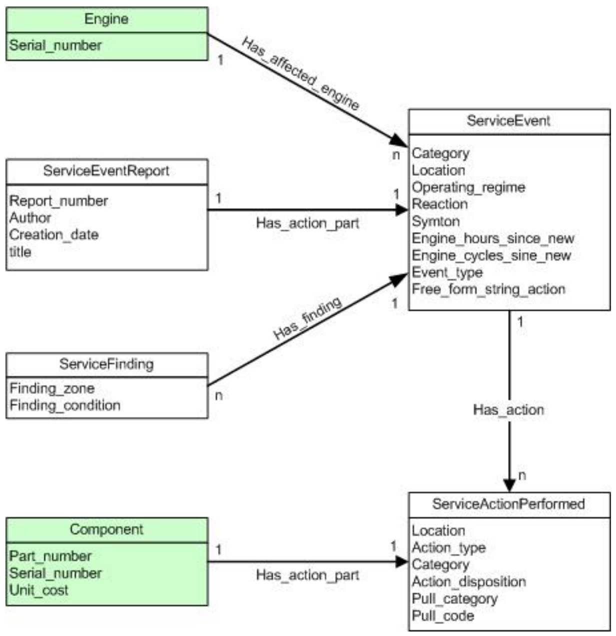


Figure 3. RDF graph for describing maintenance events. This is a simplified view and does not show all properties and classes defined in the ontology.

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 `Part` class contains a taxonomy of aero-engine parts, and the `Engine` class includes a list of existing engine types.

In the development of this application existing web standards are used wherever possible, to maximize tool reuse, compatibility and portability. Documents in the system are stored in the form of RDF triples. Both the RDF triples and associated OWL ontologies are stored in a Sesame 1.x triplestore [1].

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

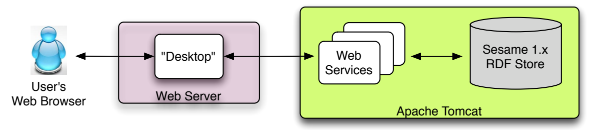


Figure 4. A conceptual view of the *Designer Desktop* work flow components.

Sesame provides a Java API for graph based access to its repositories. Functionalities such as document creation, deletion, retrieval and query are provided with Java-based Web Services that operate directly on the Sesame triplestore via the Java API. The Web Service interfaces are defined in WSDL [18], which is the standard language for defining Web Service interfaces. Specifically, we used the open source Apache Axis Java [2] for the Web Services communication stack. Since Web Services interoperate using a text-based XML protocol, consumers of these services can be written in any programming language and deployed on any OS platform. During the development of the various versions of our *Designer Desktop*, the server-side user interface code has been implemented variably on C# .NET and Java Servlet/JSP. Finally, users access the application via a web browser. The client-side user interface code is a mixture of Javascript and HTML. Figure 4 shows the work flow components of the developed *Designer Desktop*.

CONTENT DEPENDENT PRUING FOR TREE VIEW

A tree menu presents a hierarchical view of information. An example tree menu is shown in Figure 1. With a large ontology, a tree view of the ontology is often difficult to navigate. In [8], Alani et al. performed ontology winnowing by analyzing queries made by applications. Taking this approach a step further, we can prune an ontology for presentation by *anticipating* queries that will be performed by users. We achieve this by removing ontology concepts that will not be returned in any search results. For each of the chosen parameters, we ran queries to find all classes that are explicitly or implicitly referred to in the data stored in the triplestore. The tree view was then pruned by removing classes that are not used by actual data stored in the repository. Note that classes removed from the user's tree view still exist in the application ontology. If new data become available on the removed classes, the simplified tree view will need to be reconstructed.

There are three main problems with this approach. First, the pruning has to be pre-processed and cannot be done in real time. Therefore, if data become available for a class that has no data previously, the user will not be able to recall this new data via the tree menu. Second, if the underlying triples cover a large percentage of the classes in the ontology, there will not be much pruning, and the resulting simplified tree will remain large and unmanageable. Third, and most importantly, engineers who participated in our evaluation found it confusing to see only a partial listing of components. They questioned the validity of the ontology, and whether we actually used the ontology built for the project.

AUTOCOMPLETION TEXT BOX

A text box is created in HTML using the `input` element with the attribute `type=text`. It is a one-line text input area which allows users to enter free form text inside a web browser. An autocomplete text box attempts to anticipate users' intention and complete their input using vocabulary from a predefined dictionary of words. A semantic autocomplete text box is made by using an ontology as its dictionary. Whenever user enters text in the text box, a triplestore is queried to find ontological concepts that best match the input. Figure 5 summaries the interaction process for autocomplete text boxes.

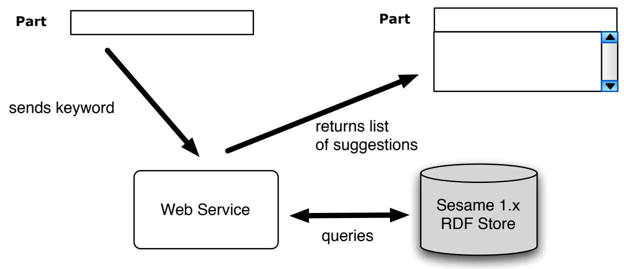


Figure 5. Interaction diagram for autocomplete text box.

We implemented a prototype of the semantic autocomplete text box. Figure 6 shows a screenshot of this prototype.

The text box is implemented on the client-side using the script.aculo.us ([3] AJAX library. Inputs from users are matched using the SeRQL query in Figure 1.

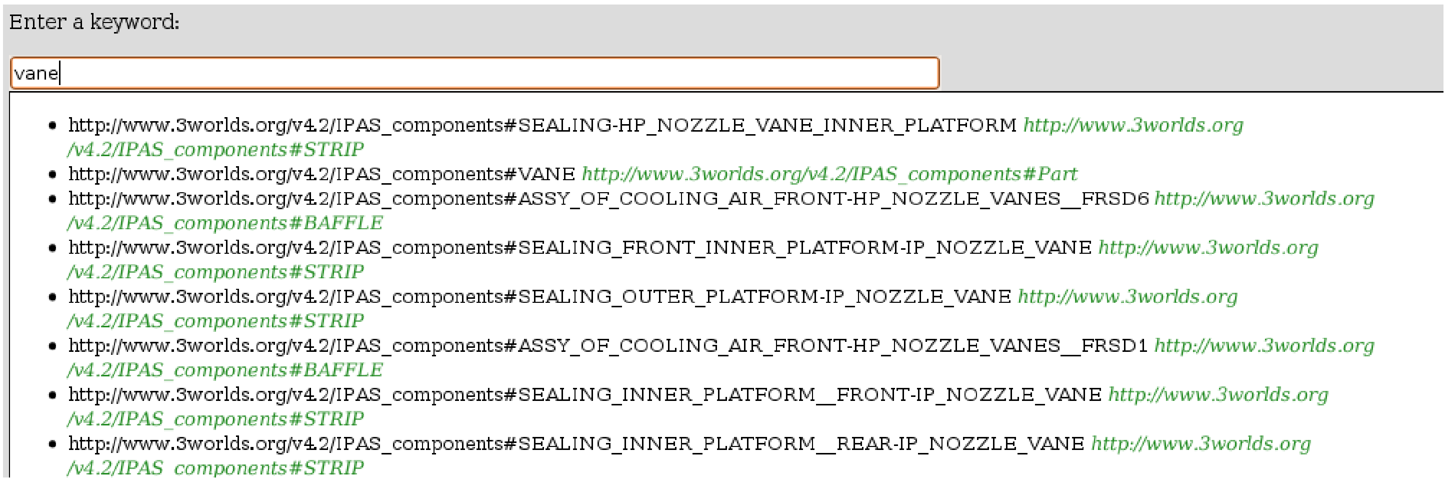
Listing 1. SeRQL query to support a generic autocomplete search box.

```
SELECT  a, b
FROM {a} rdfs:subClassOf {};
         serql:directSubClassOf {b}
WHERE a LIKE "*keyword*" IGNORE CASE LIMIT 25
```

This query searches the triplestore for any class that contains the user input as a substring. The Web Service returns the matched class and its direct parent to the client. The parent is returned to the user to provide some context to the selections chosen by the query. Since we

[2] `http://ws.apache.org/axis/java/index.html`

[3] `http://script.aculo.us`



Enter a keyword:

vane

- http://www.3worlds.org/v4.2/IPAS_components#SEALING-HP_NOZZLE_VANE_INNER_PLATFORM *http://www.3worlds.org/v4.2/IPAS_components#STRIP*
- http://www.3worlds.org/v4.2/IPAS_components#VANE *http://www.3worlds.org/v4.2/IPAS_components#Part*
- http://www.3worlds.org/v4.2/IPAS_components#ASSY_OF_COOLING_AIR_FRONT-HP_NOZZLE_VANES__FRSD6 *http://www.3worlds.org/v4.2/IPAS_components#BAFFLE*
- http://www.3worlds.org/v4.2/IPAS_components#SEALING_FRONT_INNER_PLATFORM-IP_NOZZLE_VANE *http://www.3worlds.org/v4.2/IPAS_components#STRIP*
- http://www.3worlds.org/v4.2/IPAS_components#SEALING_OUTER_PLATFORM-IP_NOZZLE_VANE *http://www.3worlds.org/v4.2/IPAS_components#STRIP*
- http://www.3worlds.org/v4.2/IPAS_components#ASSY_OF_COOLING_AIR_FRONT-HP_NOZZLE_VANES__FRSD1 *http://www.3worlds.org/v4.2/IPAS_components#BAFFLE*
- http://www.3worlds.org/v4.2/IPAS_components#SEALING_INNER_PLATFORM__FRONT-IP_NOZZLE_VANE *http://www.3worlds.org/v4.2/IPAS_components#STRIP*
- http://www.3worlds.org/v4.2/IPAS_components#SEALING_INNER_PLATFORM__REAR-IP_NOZZLE_VANE *http://www.3worlds.org/v4.2/IPAS_components#STRIP*

Figure 6. A screenshot of an autocomplete text back backed by an ontology.

have only implemented a prototype of the autocomplete text box, the list of suggestions is simply the raw URI, with no post-processing done for user display. However, post-processing can be done easily as shown in the context aware drop-down lists example. As seen in Figure 6, we display the list of suggestions as a simple unordered list `ul`. The list is in the form of 2-tuples. The first element is the suggested URI, and the second element (in italics and green) is the parent of the first. When the user clicks on a selection, only the first element of the tuple is copied onto the text box. In other words, the parent is shown purely for context and will not form part of the autocomplete entry.

By changing the query, we can limit the match to only subclasses of specified concepts. This allows us to create text boxes for each of the chosen parameter, instead of a generic search box. As an example, for a text box that expects engine parts, the query should be adapted by replacing the variable `b` in Listing 1 to the URI representing engine parts. This query is shown in Listing 2.

Listing 2. SeRQL query for autocompletion in a search box for engine parts.

```
SELECT a
FROM {a} rdfs:subClassOf {components:Part}
WHERE a LIKE "*keyword*" IGNORE CASE LIMIT 25
```

Since autocompletion is done using semantic queries over triples stored in an RDF store, a semantic text box can provide suggestions for any triple relationships. In other words, it is not limited to parent and subclass relationship used in our example. Another common relationship found in many industrial ontology is the `partOf` (or `hasPart`) predicate [19]. In aero-engines, a part can be made up by many subparts. These subparts are not subclasses. Simply by replacing the predicate `subClassOf` with `partOf` in the query in Listing 2, we can supply a text box for entering parts that belongs to a specific module. Indeed, this containment concept is found very commonly in applications outside of the industrial domain. For example, in geographical applications, the `partOf` relationship is used to represent information such as "the city Southampton is located inside the county Hampshire". It should be noted that there are several variants of the `partOf` relation [20]. The component ontology deals with the component/integral object variant of `partOf`

We have yet to use any autocompletion text boxes in our *Designer Desktop* demonstrators. This is because we would like to retain some degree of control on the search terms users enter to the system. Despite bringing up a list of suggestions, users are free to ignore the suggestions and enter any text they like. For our application, we would like users to only enter vocabulary from the ontology. This is because the queries we used to search for documents only locate exact matches (asserted or inferred). Therefore, search terms outside the ontology will never retrieve any documents from the repository. This, we believe, defeats the purpose of an application designed to demonstrate semantic web technology.

Furthermore, user studies have shown that exploratory search often produces more expressive queries than keyword search [21]. (Exploratory search are where users compose search terms by browsing or exploring the structure of the collection).

Partial Views with Drop-down Lists

A drop-down list is a user interface widget that allows users to choose one value from a list. The values are not editable. In HTML, a drop-down list is created using the `select` element, with values provided with the `option` element. We built a series of interconnected drop-down lists whose contents are dynamically updated depending on the selection of values. Figure 2(a) shows two interconnected drop-down lists, *A* and *B*.

List *A* contains the values of the parent class, while list *B* is used for children classes. The list of values presented in *B* depends on the value selected in *A*. The values in *B* is updated dynamically whenever users change their selection in *A*. More than two drop-down lists can be connected in a series, representing a deeper class hierarchy.

Figure 7 explains the interaction process behind the interconnected drop-down lists.

Whenever the user selects a value in list *A*, client-side Javascript

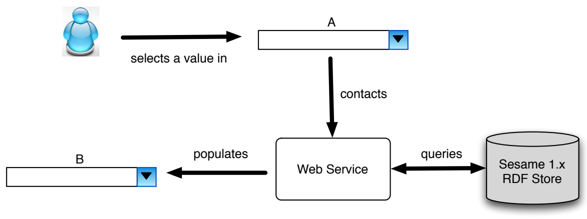


Figure 7. Interaction diagram for interconnected drop-down lists.

code contacts a Web Service with the selected value's ontology URI. The Web Service provides the following operations to support the running of the drop–down lists:

Get a list of parts, features, etc, given the ontology distance from the class's *ultimate* parent
Get the parent of a given URI
Get the list of children of a given URI

The queries are written in SeRQL and make use of the `serql:directSubClassOf` built-in predicate. (SeRQL is one of the RDF query languages supported by Sesame). An example of which is shown in Listing 3.

Listing 3. SeRQL query to get a list of all direct subclasses of a given URI.

```
SELECT children
FROM {children} serql:directSubClassOf {<someUri>}
```

By analyzing the class hierarchy corresponding to the chosen parameters, we identify the levels in the ontology designers would most likely operate within. As an example, we will use the part ontology originally shown in Figure 1. The `EngineComponent` class has only one direct subclass, `Part`. `Part` has 160 direct subclasses. Some of these are leaf nodes, such as `Actuator`; others are branch nodes, such as `Adapter`. `Adapter` has a further four children, each in turn are parents to specific part numbers such as `FK22585`. This small segment of the part hierarchy is demonstrated in the class diagram in Figure 8.

We found that all classes in level 5 represent specific part numbers and are therefore not useful as keywords to a semantic query. This is because engineers are more likely to search using the name of a part instead of its part number. Therefore, we decided to use two interconnected drop-down lists for presenting the component ontology, corresponding to levels 3 and 4 of the ontology. This analysis is carried out for all four chosen query parameters.

This navigational method of using drop-down lists to present a segment of the ontology depending on user interaction was implemented in our latest *Designer Desktop* demonstrator. Figure 9 shows a screenshot of it. The part `Adapter` is selected and its four direct subclasses are shown in the sub-part drop-down list.

The ontology for the project was developed over a couple of years and has been used in previous demonstrators. However, during the evaluation of this latest demonstrator, engineers spotted duplicate entries in the ontology. The duplicates are instances where there are multiple entries for the same concept, and are represented by strings that only

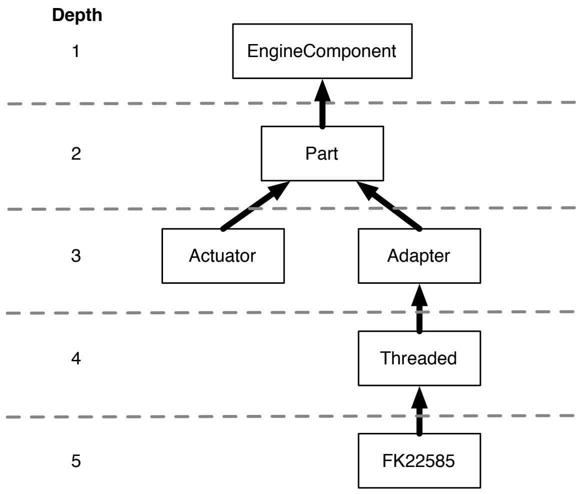


Figure 8. A segment of the *EngineComponent* class hierarchy.

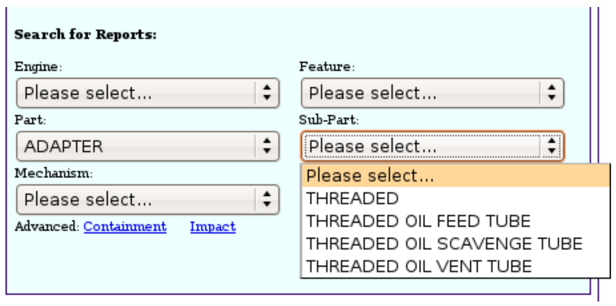


Figure 9. Interconnected drop-down lists in action in the *Designer Desktop*.

varies in the number of underscores (_) and dashes (-) present. The latest demonstrator is the first to make use of the context aware drop-down lists we presented in this paper. We believe that the navigation technique introduced has made it easier for engineers to browse and explore the ontology. Thus they are able to find errors and omissions that have been left unspotted previously.

DISCUSSION AND CONCLUSION

In this paper, we looked at the problem of ontology simplification for presentation in web applications. The ontology needed for web applications in technical domains are often very large. This is because of the vastness and complexity of the technical vocabulary captured in the ontology. In many Semantic Web applications, there is often a need for users to navigate over the underlying ontology, for example, when constructing queries or tagging documents.

We presented three methods for limiting the ontology presented to users. The proposed methods simplify the ontology for the user by presenting a fragment of the entire ontology. The ontology used in querying

Table 1. Comparison of features for the different ontology presentation methods.

	Exploratory	Dynamic	Partial view
Complete Tree View	yes	yes	no
Content Dependent Tree View	yes	no	yes
Autocomplete Text Box	no	yes	yes
Drop-down Lists	yes	yes	yes

the triplestore remains unchanged. The features of the three proposed methods, along with the traditional tree view, are summarized in Table 1.

Traditionally, class and property hierarchies in ontologies are displayed using tree views, (*Complete Tree View* in Table 1). The entire ontology is presented to the user for navigation. As the size of the ontology grows, tree views become awkward and cumbersome to navigate. With a content dependent tree view, ontology concepts that are not referred to in the triplestore are removed from the user's view. Furthermore, the partial view presented can be confusing to domain experts. This is because they might be presented with an illogical segment of the domain ontology that leaves out key concepts and features.

The remaining two proposed methods for ontology presentation are dynamic. In other words, the ontology presented to the user are a direct, real-time reflection of the actual ontology. With an autocomplete text box, users are not presented with a structure to explore the ontology. Instead, users search for information by performing keyword queries. They are also free to enter any text, including ones outside of the underlying ontology. With a series of interconnected drop-down lists, the segment of the ontology presented depends on user interaction. Users can explore the ontology by selecting concepts from any of the drop-down lists. With both the autocomplete text box and the drop-down lists implementation, no concepts are pruned from the ontology shown to users. A disadvantage of using the full ontology is that if the underlying data is sparse, users might find it frustrating after performing many queries without finding results. In our user evaluation, engineers have been able to find inconsistencies in the ontology using the drop-down list approach that are previously unspotted. This demonstrates the ease of exploring ontology structures using this method.

ACKNOWLEDGMENT

This project was/is co-funded by the Technology Strategy Board's Collaborative Research and Development program (`www.innovateuk.org`) under grant number `TP/2/IC/6/I/10292`. The authors would also like to thank the project partners for providing us with data and ontologies. Specifically, Prof. F Ciravegna and R Bhagdev for the University of Sheffield for the generation of the triples used in the demonstration system, and Colin Cadas and Dave Knott from Rolls-Royce for the provision of the case study and technical support.

REFERENCES

[1] Berners-Lee, T., Hendler, J., and Lassila, O., 2001. "The semantic web". *Scientific American,* **284**(5), May, pp. 34–43.

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

[3] Brickley, D., and Guha, R., 2004. RDF vocabulary description language 1.0 RDF schema. Technical Report W3C Recommendations.

[4] McGuinness, D., and v. Harmelen, F., 2004. OWL Web Ontology Language overview. W3C Recommendation, `http://www.w3.org/TR/owl-features`, February.

[5] Ashburner, M., Ball, C., Blake, J., and Botstein, D., 2000. "Gene ontology: tool for the unification of biology". *Nature Genetics,* **25**, pp. 25–29.

[6] Noy, N., and Musen, M., 2004. "Specifying ontology views by traversal". In Third International Conference on the Semantic Web (ISWC-2004).

[7] Seidenberg, J., and Rector, A., 2006. "Web ontology segmentation: analysis, classification and use". In WWW '06: Proceedings of the 15th international conference on World Wide Web, ACM, pp. 13–22.

[8] Alani, H., Harris, S., and O'Neil, B., 2006. "Winnowing ontologies based on application use". In 3rd European Semantic Web Conference (ESWC).

[9] Wong, S., Crowder, R., Wills, G., and Shadbolt, N., 2006. "Knowledge engineering - from front-line support to preliminary design.". In ACM Symposium on Document Engineering (DocEng).

[10] Gao, Y., Zeid, I., and Bardasz, T., 1998. "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), Apr., pp. 337–350.

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

[12] Crowder, R., Bracewell, R., Hughes, G., Kerr, M., Knott, D., Moss, M., Clegg, C., Hall, W., Wallace, K., and Waterson, P., 2003. "A future vision for the engineering design environment: A future sociotechnical scenario". In Proceedings of 14th International Conference on Engineering Design, A. Folkeson, K. Gralen, M. Norell, and U. Sellgren, eds., pp. 249–250.

[13] Priebe, T., and Pernul, G., 2003. "Towards integrative enterprise knowledge portals". In CIKM '03: Proceedings of the 12th international conference on Information and knowledge management, ACM Press, pp. 216–223.

[14] Petrelli, D., Lanfranchi, V., Moore, P., Ciravegna, F., and Cadnas, C., 2006. "Oh my, where is the end of the context?: dealing with information in a highly complex environment". In IIiX: Proceed-

ings of the 1st international conference on Information interaction in context, ACM, pp. 37–41.

[15] Wong, S., Crowder, R., Wills, G., and Shadbolt, N., 2007. "Lesson learnt from a large-scale industrial semantic web application". In 18th ACM Conference on Hypertext and Hypermedia.

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

[17] Fowler, D., Reul, Q., and Sleeman, D., 2008. "IPAS ontology development". In Proceedings of the 3rd International Workshop on Formal Ontolgies meets Industry Workshop (FOMI 2008), pp. 417–444.

[18] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., 2001. Web Services Description Language (WSDL) 1.1. W3C Note, `http://www.w3.org/TR/wsdl`.

[19] Rector, A., Welty, C., Noy, N., and Wallace, E., 2005. Simple part-whole relations in OWL ontologies. Technical Report – W3C editor's draft. `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`.

[20] Winston, M., Chaffin, R., and Herrmann, D., 1987. "A taxonomy of part–whole relations". *Cognitive Science,* **11**(44), pp. 417–444.

[21] Wilson, M., and schraefel, m., 2008. "A longitudinal study of exploratory and keyword searches". In ACM/IEEE–CS Joint Conference on Digital Libaries.