A Semantic Matching Approach for Distributed RDF Data Query on a Knowledge Bus

Tao Guan, David W. Fowler, Richard M. Crowder, Feng (Barry) Tao, Nigel R. Shadbolt, Gary B. Wills
School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
tg2,dwf,rmc,ft,nrs,gbw@ecs.soton.ac.uk

Abstract

During the past several years, semantic web technologies have been applied to facilitate the sharing of data in new and unexpected ways. Various heterogeneous data are assigned well-defined meaning by ontologies and expressed with RDF triples so that they are are able to be integrated and extracted across multiple sources. As the order of magnitude of triples hosted is higher and higher, and considering other essential issues (e.g. copyright, security) in application domains, a distributed RDF data management approach is more appropriate for knowledge sharing and integration. In this paper, we present a knowledge bus infrastructure - a general solution of locating and extracting knowledge elements from distributed sources on-demand rather than loading all of RDF triples into a large central triple store in advance. A semantic matching approach is discussed to support the key function of automatic knowledge source location. The knowledge bus infrastructure as well as the semantic matching approach has been adopted in the CFMS project, enabling a rapid information search and access for the engineering domain.

1 Introduction

During the past several years, semantic web technologies [3] [9] have been applied to facilitate the sharing of data in new and unexpected ways. Various heterogeneous data are assigned well-defined meaning by ontologies and expressed with RDF triples so that they are are able to be integrated and extracted across multiple sources to support high-level intelligent applications. Generally speaking, there are two approaches to managing a large volume of RDF triples: centralized and distributed RDF data management. In most application domains (e.g. bioinformatics), all kinds of knowledge and experimental data are encapsulated into a unified format and copied into a central data repository. However, as the order of magnitude of triples hosted is higher and higher, the performance of central triple stores becomes the bottleneck of the data integration system, for example, it normally takes several hours to load a billion triples. Furthermore, because of copyright and security issue in many application domains, sensitive data sources are required to be kept separately - only authorized users are able to access them.

In a distributed semantic data integration system, data providers wrap data into RDF format and publish the access information of data sources. Data consumers may need to connect with a number of different data sources on-demand to process a data query request. Three aspects need to be considered when implementing the interaction between data providers and data consumers: the first is a method that describes various data source features; the second is a mechanism for locating data sources so that required data can be found and selected given a query request; the third is the way of data access. Essentially, data source description, location and usage are interdependent: data source description is a prerequisite for data source location; the mechanism of locating required data sources determines how data sources should be described; data source usage depends on data source location and selection.

In this paper, we present a semantic system to support the data integration from distributed sources. Heterogeneous data resources are encapsulated into RDF triples and are connected with a knowledge bus infrastructure. The knowledge bus, the core component of the semantic system, is a general solution that serves as an intersection for both knowledge providers and knowledge consumers, making it possible

to locate and extract knowledge elements from distributed sources on-demand rather than load all of RDF triples into a large central triple store in advance. A key function supported by the knowledge bus infrastructure is to locate required knowledge sources based on query requests because the answer to a data query usually needs to exploit a number of different knowledge sources. A description model for data sources and the semantic matching algorithm are designed and developed to realize the function of automatic knowledge source discovery.

The paper is organized as follows. Section two briefly introduces the knowledge bus infrastructure. Section three presents the knowledge sources description and the semantic matching algorithm. The implementation of knowledge source location mechanism and the CFD user case are discussed in section four. Section five introduces the related work and section six concludes our research work with the discussion of further directions.

2 Infrastructure for Data Integration

2.1 Knowledge Bus Overview

The concept of a knowledge bus is borrowed from the concept of the data bus in computer architecture, which is a subsystem that transfers data between computer components inside a computer or between computers. Similar to the data bus, the Knowledge Bus provides the transportation media for knowledge communication, and allows new sources of knowledge and new consumers of knowledge to be connected together. Figure 1 shows a conceptual model of the Knowledge Bus, in which unique identifiers are assigned to bus nodes so that each entity can communicate with any other entities on the same bus by specifying the unique identifier of the target entity. A typical Knowledge Bus system consists of two kinds of knowledge node: Bus Controller, Knowledge Terminal.

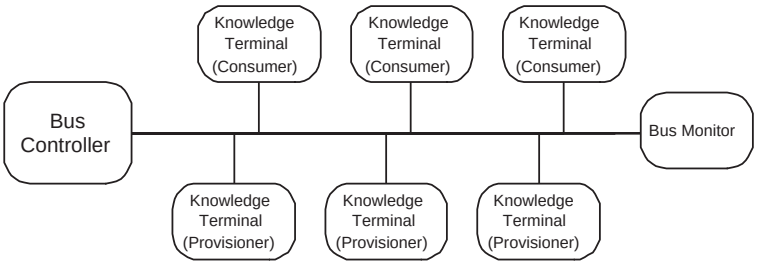

Figure 1: Concept Model of Knowledge Bus

2.2 Knowledge Communication

The communication protocol for the Knowledge Bus is to use HTTP for transportation with the content wrapped in XML/RDF. HTTP is lightweight and based on TCP/IP sockets which are widely supported in different platforms. Many HTTP libraries are available which will help the rapid development of knowledge nodes on the bus. A balance is considered between data source integration and independence: it is assumed that knowledge consumers are not expected to update the content of knowledge providers through the knowledge bus. Hence, the "GET" method of the HTTP protocol is mostly required.

A global URI is assigned to every concept on knowledge providers, so that RESTful interfaces for knowledge providers can be exposed on the knowledge bus. The syntax of a normalized URI is:

```
http://3clix.org/<ks>/<identifier>
```

where "ks" refers to the name of knowledge providers. For example, the simulation rules of the Computational Fluid Dynamics (CFD) domain on the wiki would be expressed as

```
http://3clix.org/wiki2/rules/
```

This global strategy ensures the addressability of every resources on the knowledge bus and provides a lightweight approach of resource access.

The knowledge consumers are able to extract knowledge elements in two ways. The first is to execute a "graph query". For example, the following request will be sent out to acquire the RDF statements of the "wing3759" resources:

```
Get  http://3clix.org/wiki/wings/wing3759
```

The second is implemented with the standard SPARQL query language [10]. The knowledge sources could be configured to be SPARQL endpoints, and the knowledge consumers are able to query knowledge elements as long as the URLs of the SPARQL endpoints are located. The contents requested are wrapped in the HTTP envelope and returned in the HTTP response message.

2.3 Workflow of Processing a Knowledge Query

Figure 2 shows a general workflow which takes place when users or external applications submit a query request through a knowledge node on the bus. It is assumed that the bus controller knows the information of all knowledge nodes (to be discussed in the next section).

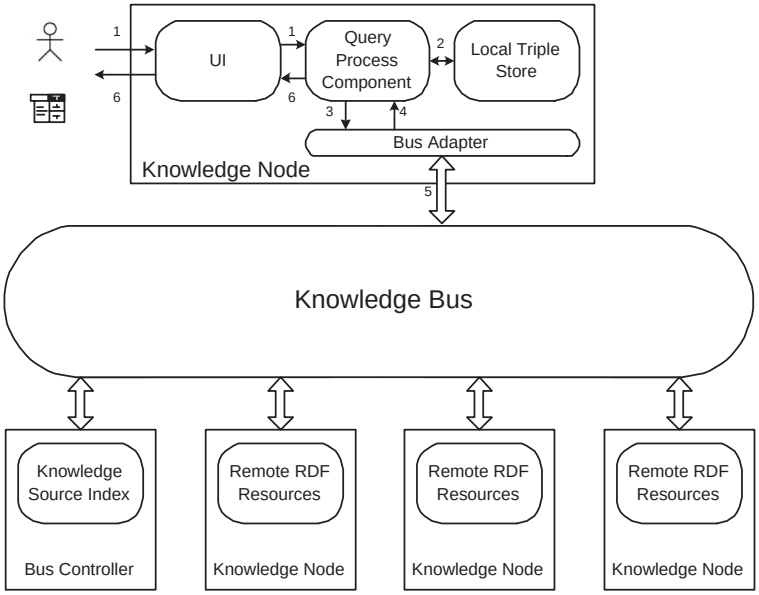

Figure 2: Answering a query in Knowledge Bus Infrastructure

An integrated process of accomplishing a knowledge query request is described as followings:

1. Query Submission: Users or external applications submit a query to a knowledge node.
2. Local Processing: the query processing component of the knowledge node receives the query, parses it and checks whether required knowledge elements can be found in the local knowledge repository.

3. Knowledge Source Location: If required knowledge elements cannot be found in the local knowledge repository, the processing component will contact the Bus Controller by sending a knowledge source location request.

4. Matching Knowledge Source Return: The bus controller looks up knowledge nodes on the bus and returns the information of matching knowledge sources to the processing component.

5. Remote Knowledge Source Access: The query processing component exploits remote knowledge sources and extracts required knowledge elements to answer the query. The knowledge elements from one remote knowledge source may be used as the input of exploiting another remote knowledge source.

6. Query Result Return: After obtaining all of the knowledge elements, the processing component makes a final process and returns the final results to users or external applications.

It should be noticed that the whole knowledge query process is transparent for users. They do not need to know detailed knowledge sources that will be exploited to answer the query. The knowledge bus infrastructure provides a function of automatic knowledge source location to realize the vision of distributed RDF data query.

3 Locating Required Knowledge Sources

3.1 Methodology

A semantic knowledge management approach is adopted to support the function of automatic knowledge source location. Knowledge sources are annotated with semantic description models, which are advertised for public access. The matching engine collects the existing knowledge source models and checks whether they are able to satisfy query requests. During such a semantic knowledge lifecycle, two issues are essential: a semantic description model for describing knowledge sources and structuring related domain concepts, and a matching engine for comparing the description models of knowledge sources and query requests. The knowledge source description model and related concepts can be defined using an ontology, which is expressed in a logical language, enabling accurate, consistent, sound and meaningful distinctions among classes, properties and relations. The matching engine can be built based on logic reasoning mechanisms, achieved by ontology supporting tools. As long as requesters present their requirements with terms from the same ontology model used to build knowledge source description models, logic reasoning mechanism can find the similarity between knowledge source descriptions and request requirements.

It is assumed that the knowledge source request attempts to describe expected requirements with terms from the same ontology model used to build the knowledge source description. However, it is impractical that every knowledge source request can acquire the exact desired knowledge source even though the required knowledge elements exist in several knowledge sources which have already been deployed and advertised, because one knowledge source could have a number of description formats so that there may be the deviation in the process of the knowledge source matching. In fact, the responsibility of the matching engine is to obtain all of the knowledge sources which could be related with requests including those that differ from the request to some defined extent. These deviation matches should not be rejected but be classified using a predefined rule (e.g. matching degree), enabling knowledge sources to be selected and exploited to check whether required knowledge elements exist. Our knowledge source location engine takes a request and available knowledge source descriptions as inputs, and outputs a list of candidate knowledge sources as well as their matching degrees.

3.2 Description Model for Knowledge Sources

Each knowledge source has a number of attributes. However, when defining a general description model for knowledge sources, only essential and most common attributes should be included in the high-level models. We define five properties to compose the high-level description model for knowledge sources. Figure 3 illustrates the description model for describing knowledge sources.

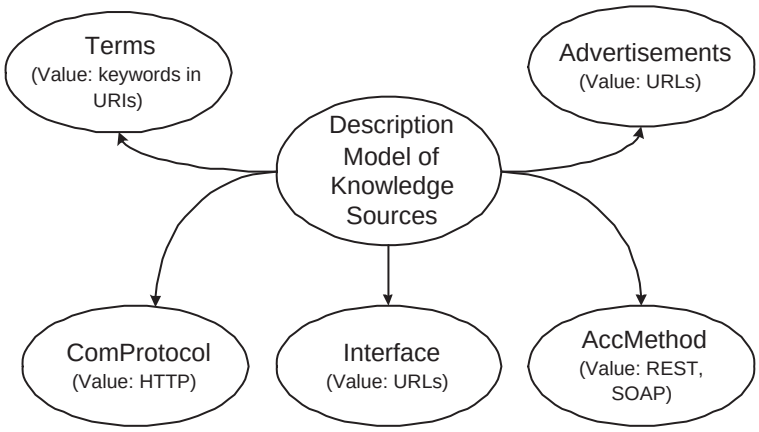

Figure 3: Knowledge Source Description Model

The defined properties are explained as follows:

- Interface: the value of this property is single or multiple URLs, a global address of knowledge source assigned when connecting to the knowledge bus infrastructure.
- ComProtocol: this property indicates the binding communication protocol. Because the current knowledge bus infrastructure only supports HTTP communication protocol, the value of this property is "HTTP".
- AccMethod: this property indicates the binding access method of knowledge sources. As discussed in the above section, the knowledge elements can be extracted from knowledge sources with RESTful interfaces (Graph Query) and/or SOAP protocol (SPARQL), hence the value of this property could be "REST" or "SOAP".
- Terms: a knowledge source could be tagged with a number of keywords, demonstrating roughly what the knowledge source is about. However, unlike most of existing tagging systems, the tags in the knowledge source description are expressed with URIs, which avoid the ambiguity issue.
- Advertisements: the value of this property is URL(s), which points to semantic metadata models of knowledge sources. This advertisement model will be used to check the detailed relationship between a request and a knowledge source.

The knowledge source description model is expressed with RDF language. The following is a brief example of knowledge source description in RDF.

```
<KnowledgeSource:presents>
   <des:DesModel rdf:ID=‘‘KnowSource1’’>
     <des:hasInterface rdf:resource=‘‘http://www.3clix.org/ks1’’/>
     <des:hasComProtocol rdf:resource=‘‘#HTTP’’/>
     <des:hasAccMethod rdf:resource=‘‘#REST’’/>
```

```
      <des:hasTerms rdf:resource=''http://3clix.org/concepts/Wings''/>
      <des:hasTerms rdf:resource=''http://3clix.org/concepts/Aerofoils''/>
      <des:hasAdvertise rdf:resource=''http://3clix.org/meta/ks1''/>
      < ......More properties......  >
    </des:DesModel>
  </KnowledgeSource:presents>
```

3.3 Knowledge Source Advertisements and Requests

Apart from interfaces, communication protocols and access methods, other features of knowledge sources are also represented in the knowledge source description, which are indicated and structured by advertisements - semantic metadata model of knowledge sources. The features of knowledge sources are either concepts or restrictions for existent concepts. The value of knowledge source advertisement is URL(s), which could be separated from the detailed knowledge source contents. Different knowledge source advertisement URLs can be made and published for one knowledge source with one knowledge source interface, enabling the knowledge sources to be reused for several purposes. For example, a knowledge source contains both knowledge elements about aerofoils and about their designers. Hence, this knowledge source can have two metadata URLs: one's topic is about "People", and the other's topic is "Aerofoil".

Similar to the knowledge source advertisement, a knowledge source request often consists of a number of individual requirements, specifying the features to be expected in a knowledge source. In order to ensure the matching process to succeed, knowledge source advertisements and requests must be described in an appropriate manner. The selection of description language is an important issue because it affects the way of describing knowledge source advertisements and knowledge source requests, and the matching algorithm between advertisements and requests. In our approach, the Web Ontology Language (OWL) is adopted to describe knowledge sources because it enables us to employ the benefits of semantic matching. OWL is often used by computer applications that need to process the content of information instead of simply presenting information to humans, and there are many existing tools supporting its editing, parsing and reasoning.

All of features that could describe knowledge sources will be specified in the knowledge source advertisements, which can be shown in the form of:

$$Advertisement \subset (A_1) \cap (A_2) \cap (A_3) \cap \cdots \cap (A_n)$$

where A_i is a named concept or an existential restriction or a complementary restriction between two values. For example, an advertisement for a knowledge source that includes data about aerofoils, contributors, and their relationship (e.g. designers have contribution to aerofoil design) is described as follows:

$$\begin{aligned} Advertisement \subset\ & KnowledgeSource \cap \\ & \exists hasElements.Aerofoil \cap \\ & \exists hasElements.Designers \cap \\ & \exists hasInternalRestriction.Res \\ Res \equiv\ & Designers \cap \\ & \exists hasContribution.Aerofoil \end{aligned}$$

A knowledge source request consists of a number of individual requirements, taking the form of:

$$Request \subset (R_1) \cap (R_2) \cap (R_3) \cap \cdots \cap (R_n)$$

where R_i is an individual requirement, taking the form of:

$$R_i \subseteq (= 1 hasReqDes.RD) \cap (= 1 hasExpMacLel.MatchingLevel)$$

where RD is the detailed requirement description, which could be a concept or an existential restriction or a internal restriction between two concepts. The *MatchingLevel* indicates matching degree of concepts in the description models that can meet this requirement. Its values are "Exact", "Substitute", "Cover", "Fuzzy" and "Close". The following example shows a knowledge source request expressed in description logic notation:

$$
\begin{aligned}
&Request \subset \exists hasReq(Req \cap KnowledgeSource \cap hasExpMacLel.Exact) \cap \\
&\quad \exists hasReq(Req \cap hasReqDes.RD1 \cap hasExpMacLel.Cover) \cap \\
&\quad \exists hasReq(Req \cap hasReqDes.RD2 \cap hasExpMacLel.Substiute) \cap \\
&\quad \exists hasReq(Req \cap hasInternalRestriction.Res1 \cap hasExpMacLel.Fuzzy) \\
&RD1 \equiv \exists hasElement.Aerofoil \\
&RD2 \equiv \exists hasElement.Person \\
&Res1 \equiv Person \cap \\
&\quad \exists hasContribution.Aerofoil
\end{aligned}
$$

3.4 Algorithm for Locating Knowledge Sources

A knowledge source request is composed of a number of individual requirements, specifying various attributes to be expected. The matching engine takes a knowledge source request and a group of knowledge description models as inputs, and is responsible for determining whether a knowledge source is a matching one for the request. The comparison between requests and description models consists of two stages. Initially, the matching engine checks whether values of "hasTerms" property in a knowledge source description model include similar vocabularies with the requirements whose expected matching level is "Exact" (concepts or values of existential restrictions in the request expression). This is a basic rough matching process to narrow down the possible candidates for further matching. If not all of "Exact" requirements can find similar terms in the values of "hasTerms" property in a knowledge source, this knowledge source will be dismissed. The result of the rough matching process is a list of candidate knowledge sources, which will be used as inputs for the next stage comparison. The second stage is an elaborate matching process: the matching engine makes a comparison between advertisements and requests of knowledge sources. As both of them are expressed in OWL language, graph matching, triple matching and then concept matching will be implemented.

During the matching process, an important issue is how to check semantic similarity of vocabularies. An assumption is made that the similarity of conceptual attributes can be judged using logic reasoning based on the taxonomic relation in ontology definition. Otherwise, other available similarity measurement approaches such as [12] and [13] will be adopted to acquire the knowledge of similarities. Same as the definition of expected matching level in the knowledge source, five expected matching levels of vocabularies are used:

- "Exact" indicate that the user expects to find a concept in the knowledge source description which is equal to the concept in the requirement.
- "Substitute" indicate that the user expects to find a concept in the knowledge source description which is the direct superclass of the concept in the requirement.
- "Cover" indicates that a concept which subsumes the concept in the request is expected to be found.
- "Fuzzy" means this requirement is of little importance for matching. As long as a concept in the description can be found which has the subsumption relationship (either superclass or subclass) with the concept in the requirement, it will be satisfied.

- "Close" indicates that the user expects to find a concept in the description which has the same direct superclass in the defined concept (ontology) structure with the concept in the requirement. This expected matching level is defined for the conceptual attribute, whose similarity cannot be assessed with the subsumption reasoning.

After rough matching and elaborate matching process, the matching engine may find a number of knowledge sources for a specific request. Although the knowledge source matching engine is not responsible for the selection (the processing component is), the matching degree information about each candidate knowledge source is required to be provided for requests. We use the term "MatchingScore" to show the matching degree of the candidate knowledge sources. For a candidate knowledge source, its "MatchingScore" is calculated using the following equation:

$$MatchingScore = \sum_{i=1}^{n} Score_i / n$$

The "$Score_i$" indicates the matching degree of every individual general requirement in the request against the related attribute in the description metadata model. For concepts between which the subsumption relation exists, the score can be obtained based on the semantic distance $||C_r, C_a||$ between the individual requirement (C_r) and the related attributes (C_a) in the ontology structure. The following equations are used to calculate the individual score:

$$Score_i = \begin{cases} 1 & if\ C_a = C_r \\ \frac{1}{2} + \frac{1}{2*(||C_r,C_a||+1)} & if\ C_a\ is\ a\ superclass\ of\ C_r \\ \frac{1}{2*(||C_r,C_a||+1)} & if\ C_r\ is\ a\ superclass\ of\ C_a \end{cases}$$

The matching score of each candidate knowledge source is calculated based on $Score_i$, and it will determine the ranking. The higher the score is, the higher ranking the candidate knowledge source has.

4 Implementation and CFD Use Case

The knowledge source location middleware is built to serve as the registry center on the knowledge bus. It provides both web application and web service interfaces for knowledge source owners to publish their knowledge sources on the knowledge bus. During the process of knowledge source publishing, the used domain concepts and restriction in the description models are extracted, and related ontology instances are created and stored in the ontology repository. When a knowledge source request is received, the matching engine only needs to parse the request, and judge similarity between concepts in the request and instances in the ontology repository. This pre-reasoning approach speeds up the time of processing a query request because it saves the time of analyzing a number of knowledge source advertisements.

The knowledge source location middleware has been implemented in Java with the MySQL database, the Jena framework, the Racer reasoning system [8], and other related techniques. Jena provides a programming environment for OWL ontologies which is used to parse knowledge source descriptions and manage required ontologies. The Racer system is responsible for executing the necessary reasoning tasks during the matching process. Knowledge source description models and advertisements are stored in the Jena triple store on the top of the MySQL database, which is captured through publishing interfaces. The knowledge source information middleware is written as both a Java Web Service for knowledge nodes on

the knowledge bus infrastructure to submit location requests and a web application for managing existing knowledge sources manually using the AJAX design mode which can be accessed through a standard web interface.

The potential of the knowledge bus infrastructure and knowledge source location mechanism is demonstrated by applying them to construct a semantic system for rapid engineering information access in the Computational Fluid Dynamics (CFD) domain.

In order to meet these requirements of CFMS project [1], a semantic system is constructed based on the knowledge bus infrastructure (Figure 4). A Semantic CFD Wiki, which is wrapped as several wiki triple stores by opening a "back door" of the wiki, and other data sources can be integrated through the knowledge bus. Semantic application gateway is an example of knowledge consumer, able to collect required knowledge elements from heterogeneous knowledge providers and assist to implement high-level applications. The detailed discussion about semantic system for the CFMS project is at [7] [6].

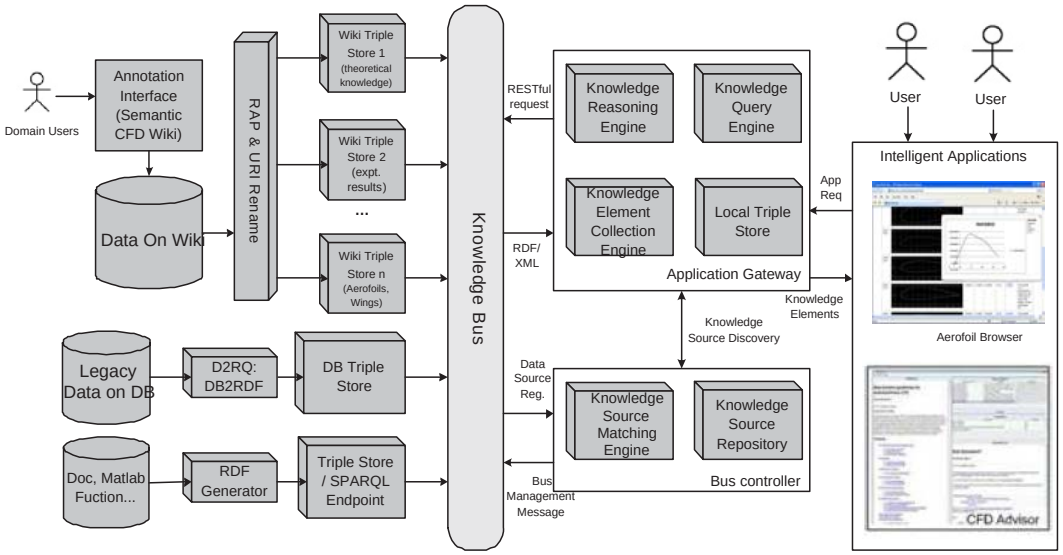

Figure 4: Semantic System on Knowledge Bus

4.1 Aerofoil Design Scenario

One of the existing applications developed is for the aerofoil design scenario, which helps aerofoils designers search and select aerofoils existed in the CFD knowledge space to fit desired performance requirements. The aerofoil design application scenario requires three kinds of knowledge sources. The first is the triple store about aerofoils from the Semantic CFD Wiki, where various existing aerofoils as well as their shape data are published. An "aerofoil" category is defined in the semantic wiki, so that users are able to add or edit the aerofoil individuals, keeping pace with the latest aerofoil update. The second is the engineers' conversation database, where previous use records of aerofoils are stored. The third is the CFD results of aerofoils, in which various aerodynamic constraints (e.g. Mach number, Reynold number) are kept. However, in the CFD domain, there are several triple stores about aerofoils, categorised by the their contributors (e.g. NACA), and many engineers' conversation databases about various topics (e.g. flight engines, wings). It is impossible and unreasonable for users to have the prior knowledge of which data sources to be used.

[1] https://www.cfms.org

For example, users may like to find information (e.g. shape data, previous experience, CFD results) of aerofoils contributed by NACA. After starting the application, a query string will be generated and transferred to the knowledge query engine in Application Gateway. The query processing component then submits three knowledge source requests to the bus controller, which can be described in description logic notation as:

$$\begin{aligned}
&Request1 \subseteq \exists hasReq(Req \cap hasReqDes.RD11 \cap hasExpMacLel.Exact) \cap \\
&\quad \exists hasReq(Req \cap hasReqDes.RD12 \cap hasExpMacLel.Substitute) \cap \\
&\quad \exists hasReq(Req \cap hasInternalRestriction.Res11 \cap hasExpMacLel.Exact) \\
&RD11 \equiv \exists hasElement.Aerofoil \\
&RD12 \equiv \exists hasElement.NACA \\
&Res11 \equiv NACA \cap \\
&\quad \exists hasContribution.Aerofoil
\end{aligned}$$

$$\begin{aligned}
&Request2 \subseteq \exists hasReq(Req \cap hasReqDes.RD21 \cap hasExpMacLel.Exact) \cap \\
&\quad \exists hasReq(Req \cap hasReqDes.RD22 \cap hasExpMacLel.Fuzzy) \cap \\
&RD21 \equiv \exists hasElement.Conversation \\
&RD22 \equiv \exists hasTopic.Aerofoils
\end{aligned}$$

$$\begin{aligned}
&Request3 \subseteq \exists hasReq(Req \cap hasReqDes.RD31 \cap hasExpMacLel.Exact) \\
&RD31 \equiv \exists hasElement.CFDResult
\end{aligned}$$

The knowledge sources which contain triples about aerofoils and NACA (or its high-level institutions because the expectation match level is Substitute) will be located for the first request; the knowledge sources of engineer conversation about any terms related with Aerofoil (based on the definition of Aerofoil ontology) will be located for the second request because the expectation matching level of "RD22" is fuzzy; as for the third request, only knowledge source about CFD results gets returned.

The matching degree information about each candidate knowledge source is also returned, which is based on the "close" relationship of concepts between requirements and descriptions. For example, for the second request, considering the following two candidate knowledge sources:

$$\begin{aligned}
&Advertisement1 \subset \exists hasElements.Conversation \cap \\
&\quad \exists hasTopic.blade \cap
\end{aligned}$$

$$\begin{aligned}
&Advertisement2 \subset \exists hasElements.Conversation \cap \\
&\quad \exists hasTopic.wing \cap
\end{aligned}$$

The score of the knowledge source one is

$$0.5 + 1/2 * (2 + 1) = 0.667,$$

which is higher than that of the knowledge source two

$$0.5 + 1/2 * (4 + 1) = 0.6,$$

because the semantic distance between "aerofoil" and "blade" is two and the semantic distance between "wing" and "aerofoil" is four.

After locating related knowledge sources, the query process component communicates with them and extract knowledge elements. Finally, all of the extracted knowledge is organized and displayed in a faceted browser (Figure 5). Designers are able to browse, compare, and select aerofoils according to values of various shape requirements (e.g. Area, Chamber, Angle-of-Attack). The conversation records provide previous use reference of aerofoils. After filtering out a large number of aerofoils, designers make a further selection based on the CFD results.

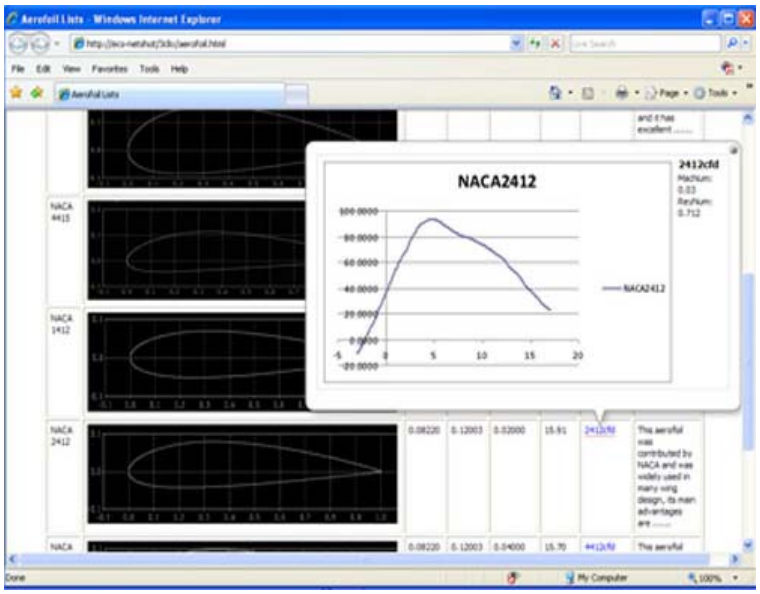

Figure 5: Semantic System on Knowledge Bus

5 Related Work

Semantic Web technologies make various online content to be machine processable, enabling software components to process them and produce value-added knowledge to end users. Semantic Web technologies have been utilized in many fields for data integration and knowledge management, e.g. Watson [5], Bio2RDF [2], Semaplorer [11]. Although existing semantic systems solved several challenges of data integration and knowledge management, most of them were using a centralized RDF data management approach and users have to know the information of targeting data sources in advance. Our approach is to provide a virtual central-storage infrastructure, enabling users to extract knowledge elements from distributed sources on demand without existing data source knowledge required.

There are also many existing work on distributed RDF storage and retrieval. For example, in [1], the authors present a distributed storage and query infrastructure on top of an existing RDF infrastructure. The triple information indexed by the predicate value is stored at a mediator to implement the function of distributed RDF query. In [4], an RDF triple repository for storing, indexing and querying individual RDF statements is presented, based on a structured peer-to-peer network. Different with these previous work, the index in our matching approach works at the ontology level – a metadata model is defined for data source description, and reasoning is used for locating required knowledge sources.

6 Conclusions and Future Work

The distributed RDF data management approach is more appropriate for knowledge sharing and integration as the size of the central triple store grows with time. This paper presents a knowledge bus infrastructure - a general solution of integrating and extracting knowledge elements from distributed sources on-demand rather than loading all of the RDF triples into a large central triple store in advance. A key function supported by the knowledge bus infrastructure is to locate required knowledge sources automatically because the answer to a data query needs to exploit a number of different knowledge sources. A description model for knowledge sources and the semantic matching algorithm are discussed to realize the vision of automatic knowledge source discovery. The knowledge bus infrastructure together

with the knowledge source location mechanism has been applied in the CFMS project, enabling a rapid engineering information access in the CFD domain.

In the future, we plan to continue our work to improve the automatic knowledge source location mechanism. Numeric attributes were not considered in the current knowledge source description model and the query request. A percent deviation or a fuzzy membership function could be used to judge similarity between numeric attributes. An alternative method for semantic matching which can provide the function of computing concept similarity is the rule-based approach. Although the rule definition is closely coupled with the application domain, presenting rules to check similarity for particular types of requirements while keeping the approach general, is worth investigating. The experiments are also planned to operate to evaluate the performance and optimize the algorithm, especially when different number of knowledge sources are deployed on the bus.

References

[1] G. Adamku and H. Stuckenschimidt. Implementation and evaluation of a distributed rdf storage and retrieval system. In *IEEE/WIC/ACM International Conference on Weeb Intelligence (WI'05)*, pages 393–396, 2005.

[2] F. Belleau, M.A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5):706–716, 2008.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[4] M. Cai and M. Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *Proceedings of the 13th international conference on World Wide Web*, pages 650–657, New York, USA, May 2004.

[5] M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing Knowledge on the Semantic Web with Watson. In *Workshop on Evaluation of Ontologies and Ontology-based tools, 5th International EON Workshop, collocated with the International Semantic Web Conference (ISWC 2007)*, Busan, Korea, 2007.

[6] T. Guan, D. Fowler, R. Crowder, N. Shadbolt, F. Tao, and G. Wills. A semantic system for rapid engineering information access. *Computers in Industry*, Submitted.

[7] Tao Guan, David W. Fowler, Richard M. Crowder, Feng (Barry) Tao, Nigel R. Shadbolt, and Gary B. Wills. A semantic system for rapid information search and access. In *6th European Semantic Web Conference (poster)*, Heraklion, Greece, May 2009.

[8] Volker Haarslev and Ralf Moller. Racer: a core inference engine for the semantic web. In *Proceedings of 2nd International Workshop on Evaluation of Ontology-based Tools*, pages 27–36, Sanibel Island, Florida, USA, Oct. 2003.

[9] D. E. Millard, N. M. Gibbins, D. T. Michaelides, and M. J. Weal. Mind the Semantic Gap. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 54–62, Salzburg, Austria, 2005. ACM.

[10] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Available at: http://www.w3.org/TR/rdf-sparql-query/, January 2008. W3C Recommendation.

[11] S. Schenk, C. Saathoff, A. Baumesberger, F. Jochum, A. Kleinen, S. Staab, and A. Scherp. SemaPlorer — Interactive Semantic Exploration of Data and Media based on a Federated Cloud Infrastructure. In *Billion Triples Challenge at the 7th International Semantic Web Conference 2007*, Karlsruhe, Germany, 2008.

[12] A. Schwering. Hybrid model for semantic similarity measurement. *Lecture Notes in Computer Science*, 3761/2005:1449–1465, 2005.

[13] A. Tverski. Features of similarity. *Psychological Review*, 8(2):327–352, 1977.