

# RDF Representation of Metadata for Semantic Integration of Corporate Information Resources

Tom Barrett<sup>1</sup>, David Jones<sup>2</sup>, Jun Yuan<sup>2</sup>, John Sawaya<sup>1</sup>, Mike Uschold<sup>2</sup>, Tom Adams<sup>1</sup>,  
Deborah Folger<sup>2</sup>

<sup>1</sup>The Boeing Company, MS P29-99, PO Box 16858, Philadelphia, PA 19142-0858

<sup>2</sup>The Boeing Company, PO Box 3707, Seattle, WA 98124-2207

{thomas.m.barrett, david.h.jones, jun.yuan, john.b.sawaya, michael.f.uschold,  
thomas.l.adams, deborah.h.folger}@boeing.com

## Abstract

In this paper, we will discuss the use of RDF-based metadata to achieve the semantic integration of corporate information resources. This approach uses the Resource Description Framework (RDF) and Resource Description Framework Schema (RDFS) for the representation of all metadata characterizing the content of the information source as well as an interlingua for standardizing all communications between system components. We will discuss the integration architecture and describe the metadata structures used. We will also describe how metadata facilitates information browsing and a practical approach to query optimization.

## Introduction

Integration of information resources continues to be a high priority in businesses today. Companies have wrestled with the integration of legacy systems for some time, but with the explosion of web-based resources, the interoperability of information has become an even greater problem. The essence of this problem is the implicit and frequently inconsistent semantics of the information. Because the web has made access to information much easier, the potential for companies to leverage information has grown tremendously. But, for the most part, each information resource was created for a single purpose, and the power of integration is in the merging of information, particularly in unanticipated ways.

Most commercial tools to date have concentrated on *access* level integration. For the last several years, research has been focused on interoperability of information which requires *semantic* level integration. The InfoSleuth project [1] used an agent architecture for semantic integration of information sources. Somewhat more recently the OntoBroker project [2] employed the Resource Description Framework (RDF) [3] to represent semantic level metadata and the DAML project [4] has extended that representation and applied it to semantic integration of services. This distinction between access level and semantic level integration is the same as Pollock's [5] *application* integration vs. *information* integration. The tools we have to help us integrate at this level are from the wealth of research done on database integration, the latest web standards, as well as the previous work done on knowledge representation.

In order to get easy access to company information, we wanted to create a toolset that would allow information browsing and ad hoc queries in a decision support role. We were interested in addressing some of the performance issues related to the integration of what McIlraith, et al [6] refer to as *information-providing* services, which include resources like structured databases and analytical tools, rather than *world-altering* services, such as on-line ordering systems which change the state of the world (in this case your credit card balance).

Our approach is similar to previous efforts, however we did not implement full problem-solving capabilities typical of agent implementations, which would be required to deal with world-altering services. Our focus was on helping users find information quickly and improving performance levels through techniques such as run-time query analysis, dynamic constraint passing, and information browsing.

We have applied the toolset to two domains so far. The first was the aircraft maintenance domain and the second was in aircraft design. It was a requirement that the toolset be able to access legacy systems without any changes to those systems. Most of the data was in conventional RDBMS systems, but for the design domain we also had to interface to a proprietary PDM (Product Data Management) system.

The paper starts with an overview of the architecture of our information integration toolset. Then we will discuss several of the more important implementation issues we had to deal with. First we describe the ontology and other metadata structures that define the interchanges between components. Next we describe how the components support several forms of information browsing. Finally we discuss our strategy for query optimization.

## **Architecture**

The architecture we used to achieve semantic integration had two major components: Resource Brokers and Domain Brokers (see Figure 1). Every resource that we wish to make available has an associated Resource Broker. The role of the Resource Broker is to handle the communication to and from the resource. It has three specific jobs:

1. translate queries to the target format (e.g., SQL),
2. convert ontological terms to terms used by the resource (e.g., database schema), and
3. translate responses back into the ontological terms used in the query.

The role of the Domain Broker is query handling. Its specific jobs are:

1. accepting queries from users (via the query generator) or other applications,
2. partitioning the query into sub-queries,
3. distributing the sub-queries to the appropriate Resource Brokers, and
4. merging the results from the various Resource Brokers and passing the combined response back to the requestor.

Step 4 is accomplished by storing the responses to the sub-queries in a temporary store and executing the original query against this information. Source information is included as part of the response so that the requestor can differentiate redundant or conflicting information. There is a simple publish and subscribe mechanism that enables new resources to register with domain brokers; alternately a resource registry could be used. Our architecture is similar to that used on InfoSleuth but, as mentioned previously, we didn't require full agent capabilities. Our query generator corresponds to their user agent, our Domain Broker performs the functions of their broker agent and multi-resource query agent, and our resource broker serves as their resource agent. We do not use an ontology agent although, for our purposes, this function could be provided by a metadata repository.

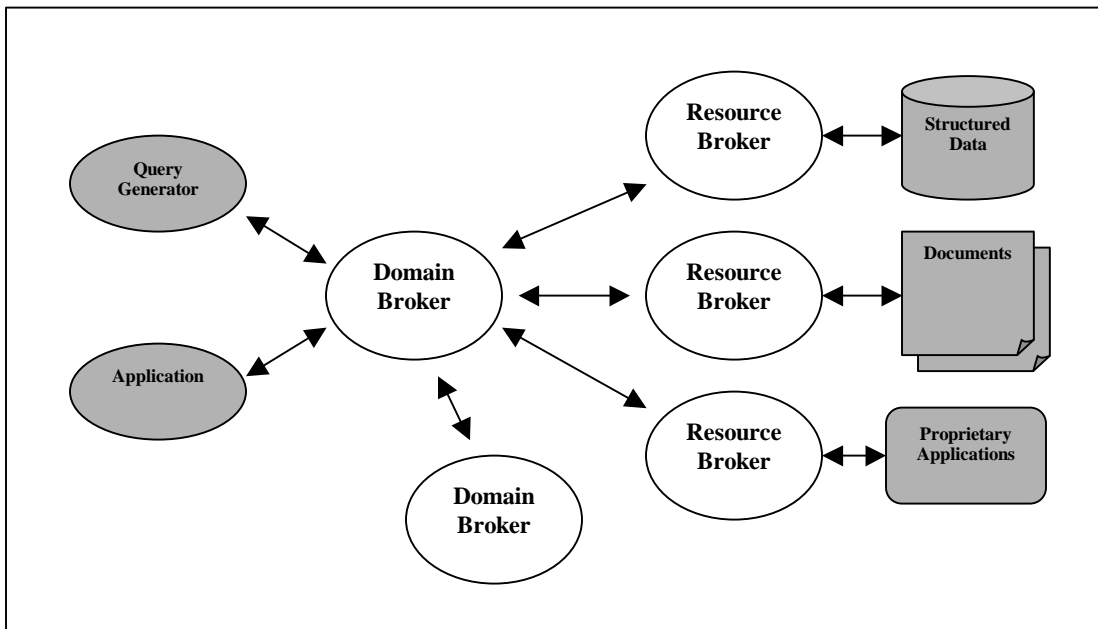


Figure 1 – Integration Architecture

Much of the metadata was created using Stanford University's Protégé tool [7], which was used to create the ontologies as well as specify the mappings of the database schema to the ontology. We have two versions of the Domain Broker with different underlying query engines. One uses the Java Expert System Shell (JESS) [8] developed by Sandia Labs and the other uses an Oracle Database. The JESS version allows developers to perform inferencing, if desired. This would be useful for more sophisticated levels of information conversion or combinations (e.g. data fusion) or in implementing the problem solving behavior mentioned earlier.

We modified the Stanford RDF API [9] to use permanent storage if the model exceeds available memory and are currently evaluating Jena [10] developed by HP to potentially replace the Stanford API. Longer term we expect commercial tools that support W3C standards to augment or replace some of the functions of the toolset. We now discuss several key aspects of the toolset.

## Ontologies and Metadata Structure

In order to provide an integrated view of two sources, there must be a set of shared concepts, even if these concepts only cover a limited part of the information stored in the two resources. The concepts could even be referenced by different terms as long as they have the same semantics and a mapping is provided. Ontologies offer a uniform set of terms with which to refer to concepts throughout a domain. They are organized in line with how users conceive of their domain rather than the data structures created to conveniently store the information. The ontology is used as a neutral interchange format as described by Uschold et al [11]. That is, rather than map terms from every source to every other source, we map them from each resource to and from the ontology. These pre-specified mappings are used to compute the ontological representations of resource information at run-time.

The ontologies are modular, layered, and possibly overlapping (equivalent terms are legal). Ontology extensions are also supported for the specific mapping requirements of individual information resources. The Resource Description Framework Schema (RDFS) was chosen to represent all schema-level metadata (both domain and infrastructure) and the Resource Description Framework (RDF) was used to represent all instance information.

In addition to the domain ontology, several infrastructure ontologies were defined in RDF(S) to support component interaction. These included descriptions of database structures, application services, and query structures. These infrastructure ontologies were not intended to be comprehensive. They were only designed to meet the minimum required functionality of the application. It was our intention to incorporate accepted standards for these ontologies as they become available (e.g. DAML+OIL).

The basic technique for mapping structured data sources to the ontology was to link database tables to classes defined in the ontology and corresponding columns to class properties. As an example, consider mapping a table in one database called AIRCRAFT to the ontology term CH-47 and the column AC\_SERIAL\_NUMBER to the ontology property AircraftSerialNumber. The RDF for this example is in Figure 2.

```
<db:DatabaseTable rdf:about="&LDMDEV;AIRCRAFT"
  db:Label="AIRCRAFT">
  <db:PartOfDB rdf:resource="&LDMDEV;LDMDEV" />
  <db:AssociatedClass rdf:resource="&am;CH-47" />
</db:DatabaseTable>
<db:DatabaseColumn
  rdf:about="&LDMDEV;AIRCRAFT.AC_SERIAL_NUMBER"
  db:Label="AC_SERIAL_NUMBER">
  <db:PartOfTable rdf:resource="&LDMDEV;AIRCRAFT" />
  <db:AssociatedSlot rdf:resource="&uo;AircraftSerialNumber" />
</db:DatabaseColumn>
```

Figure 2 – Database Table and Column Mapping

AircraftSerialNumber is actually a property of the class Aircraft and is inherited by its subclass CH-47. However, Aircraft is a very general term and is defined in a different ontology than CH-47 (hence the different namespaces, “uo” and “am”).

Schema mappings are not required to be one-to-one (ontology concept to database table) and some properties values can be derived. Mappings allow references to the content of all resources through common ontological terms. Class inheritance is supported and one-to-many relationships between database tables can be modeled in both directions (e.g. part\_of and has\_parts). We also support content conversion so that information represented differently in two or more resources can be directly compared or “joined”.

A common use of content conversion is to deal with the use of codes in the database. For example, in the maintenance database there was a column that contained data referring to part status. The value “0” meant the part was installed on the aircraft and the value “2” meant it had been removed. In this case, the Resource Broker converted these codes to the common terms of “Installed” and “Removed”. Another use of content conversion is when different data sources use different formats or data types for the same content. Content conversion is handled by the Resource Broker and can be calculated from either a value substitution table (part of the mapping metadata) or a simple conversion function. More sophisticated conversions could be handled as inferences in the Domain Broker or as separate services.

Another infrastructure ontology is used to describe queries. Rather than requiring users to input queries in a serialized form (i.e. using a string-based syntax), we defined a query structure in RDFS and specified the individual queries in RDF. The construction of the query itself is done primarily through a point-and-click interface such that the users were shielded from the underlying structure. RDF is used to transfer the queries from the query generator to the Domain Broker and then the sub-queries to the Resource Brokers. By using RDF, there is no need to do any additional parsing beyond what is provided by the RDF API. The query structure supports popular constructs like logical operators (AND, OR, NOT), comparison functions (EQ, NE, GT...), and aggregations. A simple example of a query is shown in Figure 3.

We use indirect references to classes and properties (ClassReference and PropertyReference) because a query may contain different references to the same type of class or property, particularly when specifying constraints (not shown in example). ClassReferences and PropertyReferences serve the same function as variables in certain query languages.

If the database mapping were such that the class AircraftPartDesign corresponded to the table “Part” and the properties PartDescription and PartOf corresponded to the Columns “description” and “part\_of” respectively, then the SQL translation of this query would simply be “SELECT description, part\_of FROM Part;”

The RDF represented query structures are translated to the target query language by the Domain and Resource Brokers at run-time. These brokers support translations to query languages for JESS, RDBMS (SQL), Jena, or proprietary applications.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY a 'http://mct.pw.boeing.com/qry#'>
  <!ENTITY b 'http://mct.pw.boeing.com/pdm#'>
]>
<rdf:RDF xmlns:a="&a;"
  xmlns:rdf="&rdf;"
  xmlns:b="&b;">
<a:Request rdf:about="&a;Request1">
  <a:RequestedClassReference rdf:resource="&a;CR1"/>
</a:Request>
<a:ClassReference rdf:about="&a;CR1">
  <a:HasClass rdf:resource="&b;AircraftPartDesign"/>
</a:ClassReference>
<a:PropertyReference rdf:about="&a;PR1">
  <a:HasClassReference rdf:resource="&a;CR1"/>
  <a:HasProperty rdf:resource="&b;PartDescription"/>
</a:PropertyReference>
<a:PropertyReference rdf:about="&a;PR2">
  <a:HasClassReference rdf:resource="&a;CR1"/>
  <a:HasProperty rdf:resource="&b;PartOf"/>
</a:PropertyReference>
</rdf:RDF>
```

Figure 3 – Example Query

## Information Browsing

Beyond query processing performed by the Domain Broker, there are a number of ways metadata can be used to assist someone in formulating queries. When trying to locate the appropriate concepts, users can browse the ontology. The query generator supports subclass navigation as well as range navigation (selecting classes that represent the range values of properties of the currently selected class). It is also possible to group concepts under general subject categories to aid in location. Additionally, display labels, equivalent terms, and Wordnet [12] synsets can be used to help locate desired concepts. These could be included as part of the schema information and used by the query generator to help in the identification of desired concepts.

In addition to browsing schema information, users can browse the data itself. When a data record references other records, possibly in other tables, the users have the ability to easily navigate to those records. They can do this by navigating instances of the concepts in the ontology. The instances can be generated dynamically by including two pieces of information in the ontology. First, sets of properties that uniquely identify instances of the class (analogous to key fields) are specified. Second, for properties that link classes, a mapping of object class properties to subject class properties is given.

For example, suppose the following two classes were defined:

```
Class: AircraftPartDesign
  Property: PartNumber {Range: Literal}
  Property: HasDesignGeometry {Range: DesignGeometry}
  Property: GeometryDesignNumber {Range: Literal}

Class: DesignGeometry
  Property: DesignNumber {Range: Literal}
```

The identifying properties are PartNumber for AircraftPartDesign and DesignNumber for DesignGeometry. Also, the property GeometryDesignNumber in AircraftPartDesign maps to the property DesignNumber in Design Geometry (analogous to a foreign key). The property HasDesignGeometry would have as its value a URI corresponding to the instance of DesignGeometry related to the part. The URI generation scheme we employed, however, is dependent on the data record (each record roughly corresponds to an instance), so property values other than Literals are normally not returned for each instance because it would require retrieving all associated records to calculate the URIs. But, by knowing the identifying properties and property mappings between the classes, we can automatically construct a query to retrieve the associated instance of DesignGeometry when desired.

With this information, as the user navigates from one instance to the next, the query generator creates queries to retrieve the requested records. This same metadata is used to implicitly specify joins between classes (if there are more than one, the user would have to choose), which greatly simplifies query generation. It could also be used to constrain queries to a specified context. For example, if the user wanted to retrieve the geometry for an aircraft transmission, and they had previously set a context parameter Aircraft.Model to CH-47, the following information would be included in the query.

User specified information (from point-and-click interface):

```
Desired Value: DesignGeometry.3Dmodel
Constraint: AircraftPartDesign.PartDescription = AIRCRAFT TRANSMISSION
```

Constraints added by the query generator:

```
Implied join: DesignGeometry.DesignNumber =
              AircraftPartDesign.GeometryDesignNumber
Previously set context parameter: Aircraft.Model = CH-47
Implied join from context: Aircraft.Model = AircraftPartDesign.Model
```

## Query Optimization

In order to support ad hoc querying, we had to be concerned with queries that would require excessive computation. The Domain Broker determines the complexity of a query

based on the number of responses from the Resource Brokers and how these responses must be combined. If the computations would be excessive, an analysis of the query is returned to the user with suggestions as to which classes should be further constrained.

One of the problems of combining information from distributed resources is the difficulty in passing constraints. For example, suppose one resource holds information on aircraft configurations (parts installed) and another resource holds information on repair tasks performed on individual parts. If a user wanted to find all repairs performed on a type of part in a given aircraft, the query might look like this:

User specified information (from point-and-click interface):

Desired Value: MaintenanceTask.TaskDescription

Constraint: AircraftConfiguration.PartDescription = AIRCRAFT TRANSMISSION

Constraint: AircraftConfiguration.AircraftSerialNumber = 8984567-1

Constraints added by the query generator:

Implied join: AircraftConfiguration.PartNumber = MaintenanceTask.Part.Number

The sub-queries produced by the Domain Broker would look like these:

Sub-query #1

Desired Value: AircraftConfiguration.PartNumber

Constraint: AircraftConfiguration.PartDescription = AIRCRAFT TRANSMISSION

Constraint: AircraftConfiguration.AircraftSerialNumber = 8984567-1

Sub-query #2

Desired Value: MaintenanceTask.PartNumber

Desired Value: MaintenanceTask.TaskDescription

Since the part number is not known when the sub-queries are issued, the second resource must return all of the repair task records. If this query were extended to retrieve information from yet another resource, such as the cost of labor expended on the repair tasks, the amount of data transferred and the number of combinations tested could become unnecessarily large. In fact, if the number of records returned in this matter is deemed to be excessive by the Domain Broker, it will attempt to pass constraints from the resource with the least number of responses to the next resource it references. In other words, the part number returned from the first resource is a constraint on the second sub-query and similarly, the tasks returned from the second resource are constraints on the sub-query to return labor costs from the third resource.

## Conclusion

RDF(S) has proven to be a very useful way to represent arbitrary forms of metadata for integration. We have successfully modeled ontologies, database structures, schema mapping, content conversion, service descriptions, and queries using this standard. RDF(S) has eliminated the need to do additional parsing (e.g. query syntax) and has made model manipulation and translation much easier. We anticipate that the way in which these constructs are modeled in RDF(S) will become more standardized in time. Part of the power of RDF(S) results from the ease with which we will be able to convert to (or translate between) these standard representations.

## References

- [1] Jerry Fowler, Brad Perry, Marian Nodine, and Bruce Bargmeyer, Agent-Based Semantic Interoperability in InfoSleuth, *SIGMOD Record* 28:1, March, 1999, pp. 60-67. <http://www.argreenhouse.com/InfoSleuth/publications/sigmod-record99.pdf>
- [2] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In *R. Meersman et al. (eds.), Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, Boston, 1999. <http://www-db.stanford.edu/~stefan/paper/1999/ds8/ontobroker.pdf>
- [3] Resource Description Framework <http://www.w3c.org/RDF/>
- [4] DARPA Agent Markup Language <http://www.daml.org/>
- [5] J. T. Pollock, The Big Issue: Interoperability vs. Integration, *eAI Journal*, October 23, 2001. <http://www.eaijournal.com/Article.asp?ArticleID=441&DepartmentId=5>
- [6] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng, *Semantic Web Services*, IEEE Intelligent Systems, March/April 2001  
<http://www.computer.org/intelligent/ex2001/pdf/x2046.pdf>
- [7] Protégé Project Home Page <http://protege.stanford.edu/index.shtml>
- [8] Java Expert System Shell (JESS) <http://herzberg.ca.sandia.gov/jess/>
- [9] Stanford RDF API <http://www-db.stanford.edu/~melnik/rdf/api.html>
- [10] Jena from HP Labs <http://www.hpl.hp.com/semweb/>
- [11] M. Uschold, R. Jasper, P. Clark, Three Approaches for Knowledge Sharing, KAW 1999 <http://sern.ucalgary.ca/KSI/KAW/KAW99/papers/Uschold1/final-nr-story.pdf>
- [12] WordNet <http://www.cogsci.princeton.edu/~wn/>