Silva, B., Cardoso, J., "Semantic Data Extraction for B2B Integration", International Workshop on Dynamic Distributed Systems (IWDDS), In conjunction with the ICDCS 2006, The 26th International Conference on Distributed Computing Systems July 4-7, 2006 - Lisboa, Portugal. IEEE Computer Society. ISBN: 0-7695-2541-5. ISSN: 1545-0678.

Semantic Data Extraction for B2B Integration

Bruno Silva, Jorge Cardoso
Department of Mathematics and Engineering
University of Madeira
9050-390 Funchal, Portugal
bmpsilva@hotmail.com, jcardoso@uma.pt

Abstract

Business-to-business (B2B) data exchange and integration is a common daily operation in today's organizations. These operations are crucial since they affect organizations' capability to compete in today's marketplace. Data exchange and integration has been proven to be a challenge due to the heterogeneity of the information systems involved."This paper described a Syntactic-to-Semantic (S2S) middleware which, when based on a single query, integrates data residing in different data sources possibly with different formats, structures, schema, and semantics. The middleware uses an ontology-based multi-source data extractor/wrapper approach to transform syntactic data into semantic knowledge.

1. Introduction

As organizations grow and change, their needs to manage and access information increases exponentially. In many situations, "data supporting architectures have shifted from a centralized to a distributed approach due to the advantages in the cost and flexibility". *While these trends have resulted in many advantages for organizations, they have also introduced a large gap in the ability to integrate data between applications and organizations.*

A middleware for data integration should allow users to focus on 'what' information is needed and leave the details on 'how' to obtain and integrate information hidden from users. Thus, in general, data integration systems must provide mechanisms to communicate with an autonomous data source, handle queries across heterogeneous data sources, and combine the results in an interoperable format. Therefore, the key problem is to bridge syntactic, schematic and semantic gaps between data sources, thereby solving data source heterogeneity.

At least three types of data heterogeneity may occur when integrating information from heterogeneous, autonomous, and distributed data sources: *syntactic heterogeneity*: the technology supporting the data sources differs (e.g. databases, Web pages, XML streams, etc); *schematic heterogeneity*: data sources schema have different structures; and *semantic heterogeneity*: data sources use different meanings, nomenclatures, vocabulary or units for concept.

Our approach uses a Syntactic-to-Semantic (S2S) middleware approach to resolve data source heterogeneity problems and offers the advantages of using a common shared structured format represented with an ontology. Thus, by the interpretation of a query (single point of entry), S2S generates ontology instances that permits having the retrieved data in a conceptual representation. This way, besides offering a solution for B2B integration issues (through standardised business models), it enables semantic knowledge processing.

2. S2S Middleware Architecture

Approaches to the problems of semantic heterogeneity should equip heterogeneous, autonomous, and distributed software systems with the ability to share and exchange information in a semantically consistent way [1]. A suitable solution to the problem of semantic heterogeneity is to rely on the technological foundations of the semantic Web; or more precisely, to semantically define the meaning of the terminology of each distributed system data using the concepts present in a shared ontology to make clear the relationships and differences between concepts.

The S2S approach introduces the ability to extract data from various data source types (unstructured, semi-structured, and structured) and wrap the result in OWL (Web Ontology Language) format [2], providing a homogenous access to a heterogeneous set of information sources.

The decision to adopt OWL as the ontology language is based on the fact that this is the World

Wide Web Consortium (W3C) recommendation for building ontologies.

Figure 1 presents a high level illustration of the S2S architecture. Two key areas can be identified. The first concerns the extractor (*Extractor Manager*) used to connect to the different data sources registered in the system and to extract data form them. The extracted data fragments are then compiled in order to generate ontology instances. The second key area is the mapping result between an ontology schema and the data sources (*Mapping Module*). This information is produced when the ontology attributes and classes are intersected with the data sources forming an extraction schema used by the extractor to retrieve data from the sources.

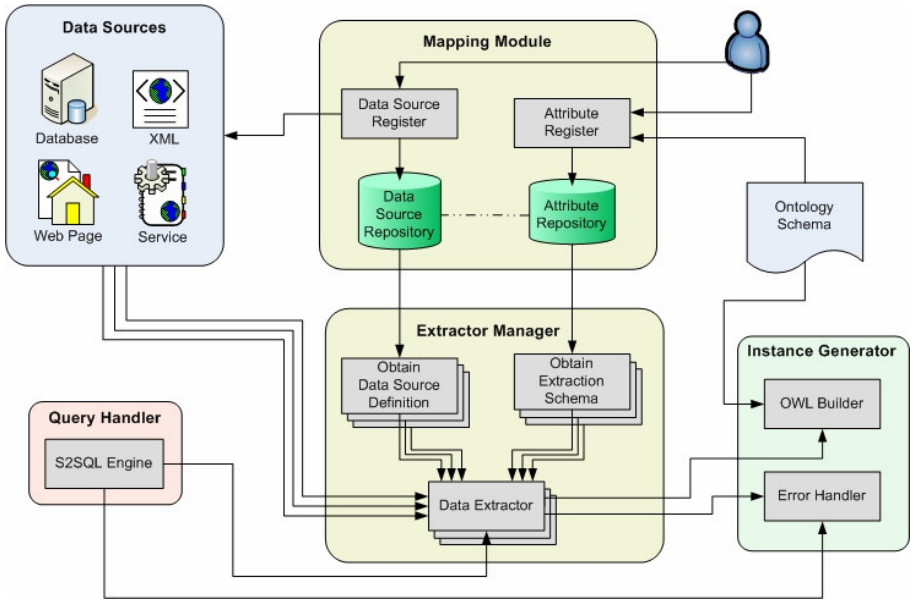

Figure 1 – Syntactic to Semantic architecture

Other areas also play an important role in the architecture. This is the case of the *Query Handler*, which handles the queries to the data sources, the *Instance Generator*, which is responsible for providing information about any error that has occurred during the extraction process or in the query, and finally the *Ontology Schema* that plays a major role in data mapping.

2.1 Data sources

The data sources define the scope of the integration system, thus data source diversity provides a wider integration range and data visibility. S2S middleware can connect to B2B traditional data source formats, such as structured (e.g. relational databases), semi-structured (e.g. XML) and unstructured (e.g. Web pages and plain text files). The supported data source types can easily be increased to support other formats.

2.2 Ontology schema

To conceptualize a domain in a machine readable format an ontology is necessary. In B2B applications, ontologies play an important role in order to promote and facilitate interoperability among systems, enable intelligent processing, and to share and reuse knowledge. From a data integration point of view, ontologies provide a shared common understanding of a domain.

S2S middleware represents ontologies using the Web Ontology Language (OWL), a semantic markup language for publishing and sharing ontologies on the World Wide Web. Other alternative formal languages can also be used to express ontologies, for instance CycL [3], KIF [4] and RDF [5].

Since the ontology schema defines the structure and the semantics of data (Figure 2) it is understandable that there is a need for the schema in the extraction process. The ontology is used to create mappings between data sources and the schema. Another important role of the ontology schema is to define the query specification process.

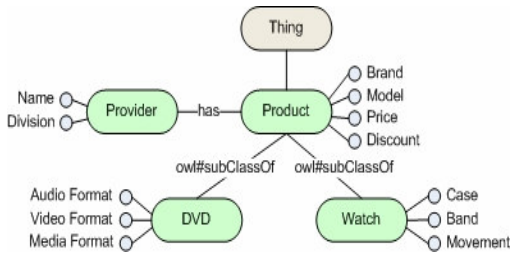

Figure 2 –Ontology schema example

2.3 Mapping Module

To enable the extraction from distributed and heterogeneous sources it is necessary to formally denote the notion of mapping between remote data and the local ontology. The mapping is the result of information crossing between the ontology schema and the data sources in order to provide information about ontology's attributes in the extraction process.

Depending on data source characteristics, two data extraction scenarios may emerge. This is because data sources might have 'one' data record (for instance a Web page describing a watch) or might have 'n' data records (for instance a database of watches). The data source scenario defines how the mapping is made and how data is extracted (in order to support the existence of an infinite number of records).

According to our approach, the mapping procedures are carried out manually. This task is time consuming but offers the highest degree of data extraction accuracy and domain consistency. This fact is very important when integrating data since the integrity and correlation between the sources and the ontology must be very accurate so that the "meaning" of the data is not lost. Although time consuming, the mapping should not need substantial maintenance after being created. Data sources do not normally change their structures (except perhaps Web pages), so few mapping updates should be necessary.

2.3.1 Attribute registration. In order to register an attribute we need information about the ontology schema and how to extract the information from a specific data source. The objective is to have a mapping specification that relates information about attributes, data sources and extraction rules.

Figure 3 illustrates the attribute registration process. In the example the data source is a Web page, so the extraction rules were set using a Web extraction language. The attribute registration process requires a set of steps to be completed in order to achieve a correct mapping. The first step is to name the attributes. The second step is to define the extraction rules. The last step maps the attribute with the extraction rule.

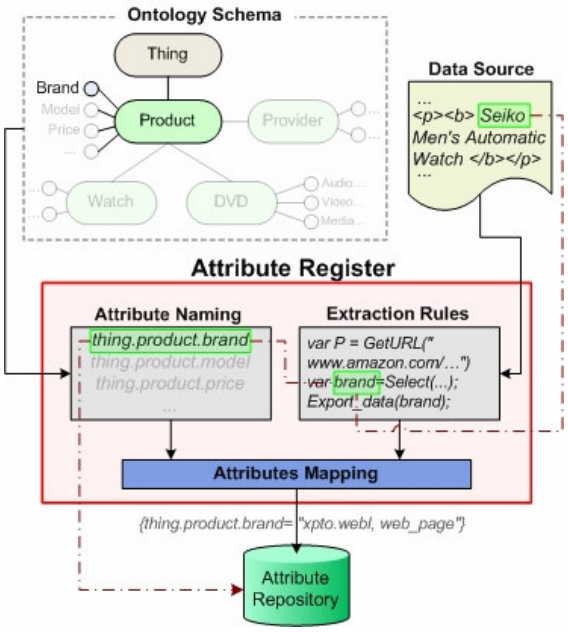

Figure 3 – Attribute Registration

Step 1 – Attribute Naming

The mapping information is supported by attributes; therefore the ontology (Figure 2) must have a corresponding extraction rule for all of its attributes. The mapping is based on ontology attributes rather than classes. The mapping system first selects a unique identifier for each attribute as shown in Figure 4.

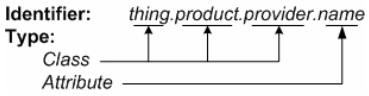

Figure 4 – Attribute naming

Besides having a unique ID to each attribute (since an attribute name may occur in more than one class), it is possible to have a path to the attributes (through the ontology classes) keeping a notion of the ontology hierarchy. We will see that this information is very important to instantiate the ontology with the extracted data.

Step 2 – Extraction Rules

Each attribute is associated to an extraction rule. Extraction rules are basically a segment of code that allows taking out the necessary data from the data source and filling a given attribute. These rules are written according to the data source type. For XML

data sources, XPath and XQuery can be used. For databases, the clear option is to use SQL. In the attribute registration example from Figure 3, the data source was a Web page so the extraction rules were defined in a Web extraction language (*WebL* [6]). Other languages or wrappers (e.g. W4F [7], Caméléon [8]) can be used.

In the example, in Figure 3, the data source is a Web page structured with HTML tags with the following information:

```
…
<p>
<b>Seiko Men's Automatic Dive Watch</b>
</p>
…
```

The following segment of code illustrates an extraction rule written in *WebL* for extracting of the watch brand from the HTML data source. The code connects to the Web site using its URL, retrieves the page and using a set of regular expressions, extracts the watch brand.

```
var P = GetURL("www.amazon.com/watches...");
var pText = Text(P);
var regexpr = "<p><b>" + `[0-9a-zA-Z']+`;
var St = Str_Search(pText, regexpr);
var spliter = Str_Split(St[0][0],"<>");
var brand = Select(spliter[2],0,6);
```

Step 3 – Attribute Mapping

Finally the mapping is completed by adding the mapping information in the attribute repository. This is done by associating the attribute ID with the extraction rule code or module. For example,

```
thing.product.brand = “watch.webl, wpage_81”
```

The attribute ID (thing.product.brand) is associated with the *WebL* file (watch.webl) containing the extraction rules and a data source identifier (wpage_81). This identifier is vital to inform the extractor manager which extractor to use and how to connect to it.

As another example, suppose that the attribute *case* (thing.product.watch.case) were extracted from a database, then the mapping information would have to be set in SQL query language and would be associated with the data source identifier to DB_ID_45. The mapping entry would have the following characteristic:

```
thing.product.watch.case =
    ”SELECT aAtribute
     FROM aTable
     WHERE aAttribute=aValue, DB_ID_45”
```

At this stage the mapping module has information about how to connect to data sources (in the *data sources repository*) and how to extract data from them (in the *attribute repository*). Therefore all mapping requisites are fulfilled, now data extraction may take place

2.3.2 Register data sources. Data sources often need specific connection information. Data source connection information must be specified to every data source used in S2S middleware. The information varies by data source type. For example, Web pages require URLs, files require paths, and databases require location, login, password, and driver type. Registering data sources separately from the extraction rules is useful to create a centralized connection information store, allowing reuse and preventing information redundancy.

2.4 Extractor Manager

This component handles data sources for retrieving the raw data to accomplish query requirements. The extraction method varies by data source so the extractor must support several extraction methods. The extractor and mapping architecture were designed in order to be easily extended to support other extraction methods and languages.

This is the main section of the S2S middleware and it is implemented by three tasks, *Obtain Extraction Schema*, *Obtain Data Source Definition* and *Data Extraction*.

2.4.1 Obtain Extraction Schema. After processing the query, the system must retrieve data in order to answer the query. The extraction is based on attributes, so this area retrieves extraction schemas of the required attributes, thus indicating to the extractor how the extraction is executed.

2.4.2 Obtain Data Source Definition Attributes are associated with data sources and data sources have connection characteristics. Therefore, extractors need to know how to connect to each data source. After retrieving an extraction schema, the extractor fetches the associated data source definition to enable its access. Now extraction can take place.

2.4.3 Data Extraction. This is the hot point in the extraction mechanism. It is supported by a mediator and a set of wrappers/extractors (details will be given in the subsequent sections). The extraction process is carried out in four steps as illustrated in Figure 5.

Step 1 – Know what data to extract

The extracting process starts by identifying what data needs to be extracted. The extraction data must be a set of attributes. This information is determined by the query handler that bases the required attribute list on a query it generates, in order to suit the query.

Step 2 – Obtain extraction schema (rules)

After knowing what attributes need to be extracted, the extractor needs to know how to extract data for them. The *Attribute Repository* has the attribute list and related extraction rules. Thus, based on the attribute list, this element retrieves the information and forwards it to the extractor.

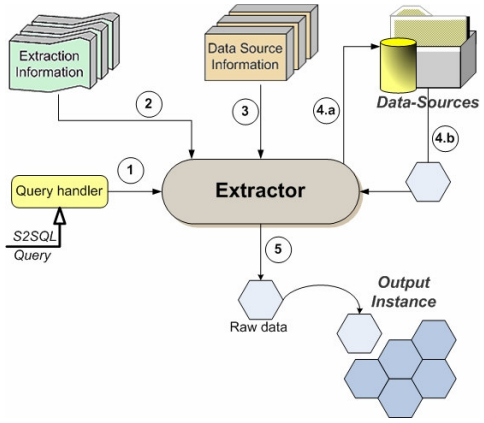

Figure 5 – Extraction process

Step 3 – Obtain data source information

After having the extraction rules, the extractor needs to know how to connect to the data sources. As shown in the attribute registering process, registered attributes have a reference in the Data Source Repository that expresses data source connection information. In this step, all references to the Data Source Repository entries from each attribute are listed and the respective connection information is retrieved. After completing this phase, all requisites to extract data are fulfilled.

Step 4 – Extract data (Data Extractor)

Now data extraction mechanisms begin to gather data. First, the extraction manager delegates a specific extractor for each extraction method depending on the data source type. For Web pages, the extraction rules are delegated to a Web wrapper, for databases to a database extractor, and so on. The extractor executes the extractions rules in the data sources and obtains chunks of data. These data fragments of raw data are then sent to the *Instance Generator* to be compiled in to an ontology instance.

2.5 Query handler

A query is the event that sets the S2S extraction middleware in action. The input is based on a higher level semantic query language. This query is then transformed to represent requests based on ontology classes. The Syntactic-to-Semantic Query Language (S2SQL) is the query language based on SQL supported by the extraction module. It is a simpler version of SQL since data location is transparent from the query point of view. Thus the *FROM* and related operators have no use in S2SQL and are thus not supported. This way, queries are created only with the indication of which data is required. It is not necessary to supply information about data location, data format, extraction method, etc. The syntax of S2SQL is the following,

```
SELECT <ontology class>
WHERE <attribute><operator><constraint>
AND <attribute><operator><constraint>…
```

An example of a query would be,

```
SELECT product WHERE brand=”Seiko”
AND case = “stainless-steel”
```

The output is based on the ontology schema, more precisely ontology classes. The result of the previous example is all products with the brand *Seiko* and case *stainless-steel*, i.e., product classes that have brand *Seiko* and case *stainless-steel*. Thus the query output will have all their associated classes, i.e. all products have a *Provider* (Figure 2), and therefore the output classes will be *Product*, *watch*, and *Provider*.

2.6 Instance generator

This module serializes the output data format and handles the errors from the queries and from the extraction phases. The S2S middleware supports the output format OWL, but other outputs can easily be adapted to export plain text to XML, , and so on, being either structured, semi structured or unstructured formats.

The ontology population process (OWL instance generation) is executed in an automatic way. This is because the extracted information (used to map the ontology to the data sources) respects the ontology schema (classes and relationships). Therefore, transforming the unique identifiers of the ontology attributes in a XML format is done naturally. This way it is easier to visualize data hierarchy and how the direct mapping works. Direct mapping is done by transforming the XML structure into the ontology

structure. Data semantics is set in the ontology schema and maintained in the output since the whole extraction process is based on the same ontology schema. This approach has the advantage of providing an ontology-independent system.

4. Related work

There are several research projects which target the same objectives as the S2S middleware. The main differences are that we use semantics and ontologies to achieve a higher degree of integration and interoperability. The World Wide Web Wrapper Factory (W4F) [7] toolkit is a good framework to develop Web wrapper/extractor. It allows the user to create Web wrappers and deploy them as modules in a bigger application.W4F extracts exclusively from Web pages and the output may be in an XML file or a Java interface. The Caméléon Web Wrapper Engine [8] is capable of extracting from both text and binary formats. The engine provides output in XML. Artequakt [9] is an Automatic Ontology-Based Knowledge Extraction from Web documents that automatically extracts knowledge from an artistic ontology and generates personalized biographies. The major drawback of this system is that it is customized to a specific domain. The Architecture for Semantic Data Access to Heterogeneous Information Sources [10] allows heterogeneous data sources to have uniform access through a common query interface based on Semantic Data Model.

5. Conclusion

Creating B2B processes for integrating various organizations are difficult to compose since organization data sources and systems are heterogeneous. One way to increase the degree of integration of B2B links between partners is to use middleware technology. Nevertheless, most current middleware only covers syntactical integration and it has been recognized that semantics are an indispensable approach to support and enhance integration. Therefore, in this paper we have presented middleware architecture for semantic B2B integration. The main goal of the architecture is to offer a common understanding of a domain and assimilate heterogeneous systems (using semantic Web technology). All this is supported by structured data (Ontology schema) thus offering semantic data representation benefits that allow data to be shared and processed by automated tools as well as by people.

We believe that the solution outlined above provides a useful tool for taking better advantage of the future capabilities and benefits of semantic data models and the semantic Web.

7. References

[1]. Sheth, A., Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, in Interoperating Geographic Information Systems, M.F. Goodchild, et al., Editors. 1998, Kluwer, Academic Publishers. p. 5-30.

[2]. OWL, OWL Web Ontology Language Reference, W3C Recommendation. 2004, World Wide Web Consortium, http://www.w3.org/TR/owl-ref/.

[3]. Cycorp, Cyc KNowledge Base - http://www.cyc.com/. 2006.

[4]. Genesereth, M., Knowledge Interchange Format (KIF) - http://logic.stanford.edu/kif/dpans.html. 2006.

[5]. Lassila, O. and R. Swick, Resource Description Framework (RDF) model and syntax specification. 1999, W3C Working Draft WD-rdf-syntax-19981008. http://www.w3.org/TR/WD-rdf-syntax.

[6]. Kistler, T. and H. Marais, WebL - a programming language for the web. Computer Networks and ISDN Systems, 1998.

[7]. Sahuguet, A. and F. Azavant. Building Intelligent Web Applications Using Lightweight Wrappers. in 25th International Conference on Very Large Data Bases. 1999. Edinburgh, Scotland, UK.

[8]. Firat, A., S. Madnick, and M. Siegel. The Caméléon Web Wrapper Engine. in Workshop on Technologies for E-Services (TES 2000). 2000. Cairo, Egypt.

[9]. Alani, H., et al., Automatic Ontology-Based Knowledge Extraction from Web Documents. IEEE Intelligent Systems, 2003. 18(1): p. 14-21.

[10]. Rishe, N., et al. The Architecture for Semantic Data Access to Heterogeneous Information Sources. in 15th International Conference on Computer and Their Applications (ISCA 2000). 2000. New Orleans, Louisiana, USA.