Contact Author	Gregory Cobena
Address	Gregory.Cobena@inria.fr INRIA Rocquencourt Domaine de Voluceau BP105 78153 Le Chesnay FRANCE
Phone	+33 1 3963 5662
Fax	+33 1 3963 5674

Paper Number	———
Title	Model, Design and Construction of a Service-Oriented Web-Warehouse
Authors	Serge Abiteboul, Vikas Bansal, Gregory Cobena, Benjamin Nguyen, Antonella Poggi
Area	——

Model, Design and Construction of a Service-Oriented Web-Warehouse

Serge Abiteboul Vikas Bansal Grégory Cobéna Benjamin Nguyen Antonella Poggi

INRIA
Domaine de Voluceau - Rocquencourt 78153
Le Chesnay
France

Abstract

We propose a new methodology, a language and tools for the design and construction of Web data warehouses. Our approach is Service Oriented, in that our framework makes an extensive use of Web Services and semi-structured data (XML) to define the data structures, the services and the connections between them. We present an experimental version of the system that has been built using this framework. It uses external Web Services such as Google's WebAPI, and Web Services we implemented, including a scalable crawler, a classification and clustering engine based on links semantic, a Pagerank module and a versioning system. An important aspect in our work is that our architecture allows the warehouse data to change in a continuous way, when Web data changes, or when the users refine their choices.

Introduction

The construction of a Warehouse of Web resources on a specific topic is a major issue nowadays. This problem has various aspects: (i) the construction of the warehouse by finding relevant resources on the Web, (ii) the organization of the data and meta-data about these resources, (iii) the querying of the warehouse, (iv) its maintenance. Our approach is based on the use of XML to store the information, and meta information of the documents, and on Web services that perform various processing operations such as crawling or classification. All this is integrated in a simple declarative language, by our prototype system called `SPIN`.

We will demonstrate the following contributions:

- *A declarative specification:* We propose a two part language to easily specify a dynamic warehouse of documents: the *data model*, a language to describe the data, and the *service model*, a language to describe the services and queries that manage the data.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

- *GUI*: We will present a graphical user interface to design and construct a dynamic warehouse. The GUI gives a graphical representation of both our *data model* and *service model*, It provides the logical view of the content of the warehouse in terms of data and services. It also helps us define some XML-based dynamic types to describe its meta-data.

- *A system:* The SPIN system constructs and maintains the warehouse based on its declarative specification and user feedback. The system consists in a *Warehouse-Compiler* and an execution platform. The platform is based on standard tools (e.g. Java runtime) and on more specific tools, such as ActiveXML [1]. ActiveXML is a system that enables the use of Web service calls inside XML documents, in the spirit of Sun's JSP or Microsoft ASP. The compiler generates an instance of the warehouse (initial content and XML schemas) and the program that will manage the warehouse and calls to the Web Services. We provide both a Java and ActiveXML platform, and we provide a high level specification of requirements for a platform to support our system (e.g. the platform needs in terms of storage, querying and update mecanisms).

- *User Interaction:* An application server based on an XML query language is used to browse the warehouse, select and process content. Users may provide feedback by updating some of the warehouse data. This feeback may be used by other services.

- *Evolution:* The warehouse should try to reflect the actual state of the changing Web. In this spirit, the system supports change control (versions, query subscription) that grant evolution capabilities to the warehouse.

- *A library of warehouse services:* We provide a set of services that are frequently used by warehouse designers such as Web-crawling, classification, Web searching, Pagerank, version management. We also provide some that are specific to our application *e.dot*. The /em e.dot project is sponsored by the RNTL program of the French Government [4].

1 Motivations

The context of our work is the *e.dot* project : the construction of a warehouse on the topic of *food-risk assesment*. More precisely, the goal of our work is to create a warehouse containing data to be used by biologists to conduct their studies. Some

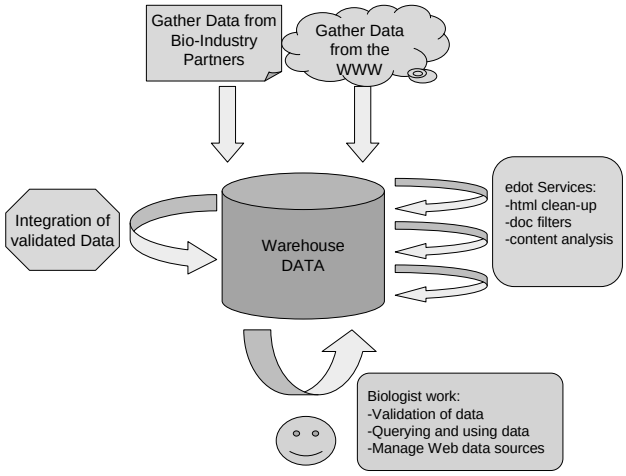

Figure 1: *e.dot Warehouse*

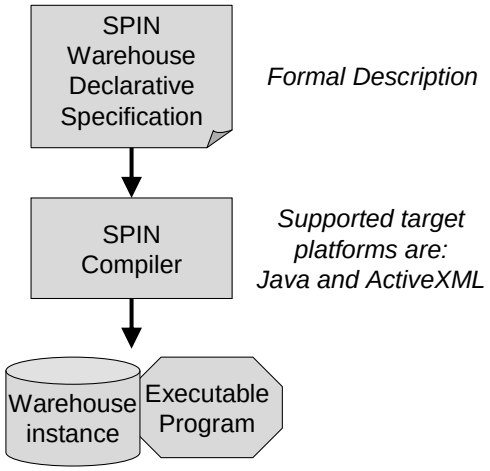

Figure 2: *SPIN Architecture*

of the data is already structured and comes from a relationnal database, managed by a partner in the project. The rest of the data should be discovered and retrieved from the web. The warehouse is used to:

- retrieve data of interest from the Web.
- analyse its content, and extract the numerical values of bio-experiments.
- enable validation of Web data by technicians by providing context information: the data source (e.g. web site), the topic-category of the documents...
- integrate the web data and the relationnal DB.

Figure 1 shows a warehouse that is constructed using the system. It stores Web data and DB data. Several Web Services are used to clean, filter, and enrich the Web data. Users (bio-technicians) are also involved in the dynamics of the warehouse since they (in)validate some of the Web data. For instance, they may approve a well known reliable source of data (*http://www.pasteur.fr/*. Or they may annotate specific keywords (e.g. a list of bacteria) that are used by the crawler to query search engines. Automatic tools are also used to validate the Web data against the relationnal DB. Finally,we provide querying tools to exploit the data.

Previous work is abundant on the topic of data warehousing, mediation and integration of data. Our work does not address the technical aspects of data warehousing, but provides a higher-level method that defines how to use these as services. Some of the technologies (e.g. XML storage and querying) used as an underlying tool. For space reasons, we do not detail these contributions here.

Our work is in the spirit of conceptual modelling languages. Users define their conceptual model according to the data and service specification language. It specifies how to use web services to find, enrich and update the data. The novelty lies in the combination of an XML data model and Web-Service model. For space reasons, we do not detail previous work in this area.

2 SPIN Architecture

In this section, we present the architecture of the system.

The system consists in a language specification that can capture the description of dynamic warehouses. Data in the warehouse is stored in XML. It is dynamic in that the language enables the use of queries and Web service calls to modify and update the content of the warehouse.

As was just mentioned, a number of technologies are needed to construct and maintain such a warehouse. The approach we follow is based on: (i) XML to store and exchange data, (ii) an extensive use of Web services [6, 7], (iii) XML query languages (XQuery and XOQL [2]) and update languages (XUpdate).

The compiler supports two target platforms. One is Java: it creates in the XOQL XML repository a first instance of the warehouse and Java programs that run the queries and call the Web Services. The prefered target platform is ActiveXML. ActiveXML enables simple integration of XML data and Web Services described in WSDL: it is an extension of XML with embedded services calls. The compiler constructs an ActiveXML document that is very similar to the data model specified using the SPIN language. It simply adds service calls to the ActiveXML document according to the specification of our language.

An overview of the architecture is presented in Figure 2.

3 Data and Service Model

In this section, we present the data model that we use to describe warehouse data. Then, we present the service model that organises Web Services to dynamically enrich the warehouse.

3.1 Data Model

The Data Model consists of a *warehouse model* and a *type model*. The type model defines *DataTypes*, i.e. semi-structured data fragments, which will be used to represent pieces of information. The warehouse model defines a tree-hierarchy of data entities (each entities being an instance of a DataType) and collections of some entity. The relation between an entity and its children is *enriched by*. The relation between a collection and its unique child entity is *collection of*.

An important aspect in the model is that each piece of information in the warehouse can be uniquely identified. To do so, one aspect is careful naming of DataTypes and entities (details omitted). The other aspect is that for each collection, a key (on the child entity) is specified that identifies each of the items of the collection. This is in the spirit of [3] (details are omitted).

Example: We define the following DataTypes for our *e.dot* warehouse: textcontent (the content of some web page), author, authors, URL, document in the following way:

```
<typeModel:define datatype="textcontent">
  <typeModel:child type="string"
      name="value"/>
</tyeModel:define>
<typeModel:define datatype="URL">
  <typeModel:child type="string"
      name="value"/>
</tyeModel:define>
<typeModel:define datatype="document">
  <typeModel:child type="URL" name="URL"/>
        <typeModel:child type="textcontent"
            name="content"/>
</typeModel:define>

<typeModel:define datatype="author">
  <typeModel:child type="string"/>
</tyeModel:define>
<typeModel:define datatype="authors>
  <typeModel:child type="collection"
       of="author" key="author.value"/>
</typeModel:define>
```

Then, we define the warehouse model as follows: the warehouse is a *collection* of *documents*. The key of the collection has to be specified. We choose to use: `document.URL.value`. Each document may be enriched by an *authors* entity. It may also be enriched by other entities using datatypes that we did not present here: *experiment*, *measure*, *bacteria*. The warehouse model is as follows. Note that we omit the name of some entities: it is by default identical to the name of the datatype.

```
<collection name="TheWarehouse"
            key="document.URL.value">
  <entity datatype="document" >
    <entity datatype="authors" />
    <entity datatype="experiment">
      <entity datatype="bacteria"/>
      <collection name="measures"
                  key="measure.item">
        <entity datatype="measure" />
      </collection>
    </entity>
  </entity>
</collection>
```

Based on this data model, the compiler does three things: it merges the type model and warehouse model into a single XML tree, it generates the corresponding DTD and XMLSchema based on type names and entities names, and it transforms the abstract queries on the service model into queries that run on the warehouse schema.

Note that there is a possible overlap between the type model and the warehouse model. More precisely, the relations between pieces of information can be described using the type model or using the warehouse model. In other words, using some basic types such as *string*, the warehouse model may be sufficient to build all possible warehouses. However, the type model is useful for engineering reasons, it enables reuse of software components in different warehouse and of specific queries and tools. Intuitively, the warehouse model consists in the conceptual model of the warehouse, whereas the type model represents the low-level implementation design.

3.2 Service Model

Using a Web Service is like a function call. The service takes input parameters, and returns some output. In order to use Web Services, there are three important questions: *when* to call the service, *what* is the input, *where* should the output be placed.

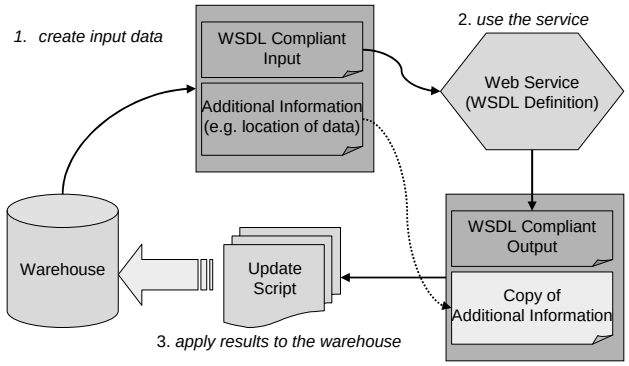

Figure 3: *Service Model*

Here is our answer to the first two questions: In order to call a service, the input parameter is defined as a query (in the spirit of XQuery) over the warehouse data. The system is in charge of calling the Web Service when needed, e.g. when the input data has changed. If the query returns a list of results, the service may be called several times to process each input item. Obviously, this query is specified by the designer of the warehouse since he knows how he wants to use the services. However, our GUI helps simplify the use of services so that the user need not explicit the query. We will demonstrate how this is easily achieved.

It is also up to the warehouse designer to specify where output results should be placed. To do so, for each Web service, the user must specify a list of update queries that describe where the data returned by the serviceshould be placed. More precisely, we apply an update script that is described in the XUpdate (XML Update) language. This update script is itself obtained by the evaluation of a query (XQuery) over the service output. Note that when XQuery will fully support updates, we plan to use this feature.

It is often the case that the output of some service call should be placed close to the input. For instance, if a service call finds the list of bacteria in some document, we want to place that list as a child of the document URL node. This can be done as follows: The query that defines the input of the Web Service may also generate information about the location of the input data. This location information is provided as an identifier using a path descriptor.

The process in shown in Figure 3.

Note that these queries are written based on the warhouse and type model. Thus, we call them *abstract* queries. The compiler translates them into XQuery (or XOQL [2]) queries. In a similar way, the path identifier of some element (obtained by the `PathIdentifier()` function in a query) is translated by the compiler into a call to a similar function that returns an XPath expression. This XPath expression has to use specific techniques of our model, e.g. keys, instead of position numbers. Thus, we use our own implementation of this function in XOQL [2].

Example: Consider a Web Service that retrieves the list of authors from the text of a document. The Web Service that takes the document (a long text string) as input, and returns a possibly empty list of authors. In our warehouse, we may define the input-query of this service like this:

```
SELECT
<query>
```

```
  <input> $a/textcontent
  </input>
  <location> PathIdentifier($a/URL)
  </location>
</query>
FROM $a in //document
```

This location information is appended to the output of the service call. Thus, the output queries can use this information to place output results in some specific location that is input-dependant. In our example, the result would be:

```
<result>
  <output>
    <author>
      John Smith
    </author>
  </output>
  <location>
    /documents/document[URL:text()="..."]
  </location>
</result>
```

In our example, the web service is used to detect authors and institutions appearing in the document content. This information is then placed below the document node.

```
SELECT
<xupdate>
  <xupdate:insert path="$a/authors">
    $b
  </xupdate:insert>
</xupdate>
FROM
$a IN result/location::text(),
$b IN result/output
```

This generates the following update script:

```
<xupdate>
  <xupdate:insert
path="//document[URL:text()="..."]/authors">
    <author>John Smith</author>
  </xupdate:insert>
</xupdate>
```

This script is then used to update the warehouse content.

Remark. In some cases, a Web Service may be called in different parts of the warehouse. Although it is in theory possible to handle the various uses with a single input query, it is more convenient to define several. Each of them corresponds to some specific use of the Web Service. A list of output queries is attached to each input query.

4 Experiments

A first prototype of our system has been implemented. It consists of: (i) a language specification, (ii) a graphical editor to define the data and service model in our, (iii) a compiler that generates executale warehouse programs in Java and/or ActiveXML.

The main difference between our prototype and the language presented here is the management of the service model. Indeed, for time reasons, we did not integrate a fully fledged XQuery processor in our system. Thus, input and output queries for Web Services are defined using XPath expressions to retrieve nodes, and predefined XML templates to construct the query result.

We constructed a warehouse of INRIA... with ... pages ... ... === COMPLETER HERE – METTRE RESULTATS EDOT/XYLEME??? OK mets les greg === Most of the Web Services that we describe here have been implemented. In particular, we have a distributed crawler (Xyleme crawler) that crawled more than a billion web pages. We also have a clustering algorithm that has been presented in [5].

We will demosntrate SPIN using the application developed in the *e.dot* project. The demonstration will consist in:

- showing how a simple warehouse is specified. This will illustrate the SPIN model and specification GUI. The construction of a small warehouse will be shown.
- An existing full fledged warehouse will be exhibited. This will illustrate a real world aplication, and let us demonstrate the change control aspect of our work.

Conclusion

We presented a model, language and tools to design and construct warehouses of Web data. Our system relies on the use of Web Services that provide knowledge and processing to enrich the contents of the warehouse. In our current implementation, we use services that we implemented (e.g. Crawler), as well as services already available on the Web (e.g. Google). The large number of services that are found on the Web nowadays confirms the importance of our choices.

Our current implementation is based on ActiveXML that integrates XML data and Web Services described in WSDL. In the future, we plan to extend our system in different ways. We plan to show how to design and construct peer-to-peer warehouses of Web data. We also plan to improve our current Web Services and implement new ones.

Acknowledgments:

We want to thank Omar Benjelloun and Beiting Zhu for discussions of the topic.

References

[1] Serge Abiteboul, Omar Benjelloun, Ioana Manolescu, Tova Milo, and Roger Weber. Active XML: Peer-to-Peer Data and Web Services Integration (demo). VLDB, 2002.

[2] Vincent Aguilera and al. X-OQL query language for XML. `http://www-rocq.inria.fr/ aguilera/xoql/index.html`.

[3] Peter Buneman, Susan Davidson, Wenfei Fan, Carmen Hara, and Wang-Chiew Tan. Keys for XML. *Computer Networks, Vol. 39*, August 2002.

[4] http://www-rocq.inria.fr/verso/edot/.

[5] Maria Halkidi, Benjamin Nguyen, Iraklis Varlamis, and Michalis Vazirgiannis. THESUS: Organizing Web Document Collections Based On Semantics And Clustering. Technical report, Gemo, July 2002.

[6] W3. Simple Object Access Protocol (SOAP). `www.w3.org/TR/SOAP`.

[7] W3. Web Service Description Language (WSDL). `www.w3.org/TR/wsdl`.