EXTENSIBLE INFORMATION BROKERS

JIANGUO LU, JOHN MYLOPOULOS

Department of Computer Science, University of Toronto
{jm, jglu}@cs.toronto.edu

ABSTRACT The number and size of information services available on the internet has been growing exponentially over the past few years. This growth has created an urgent need for information agents that act as brokers in the sense that they can autonomously search, gather, and integrate information on behalf of a user. To remain useful, such brokers will have to evolve throughout their lifetime to keep up with evolving and ever-changing information services. This paper proposes a framework named XIB (eXtensible Information Brokers) for building and evolving information brokers.

The XIB takes as input a description of required information services and supports the interactive generation of an integrated query interface. It also generates wrappers for each information service dynamically. Once the query interface and wrappers are in place, the user can specify a query and get back a result which integrates data from all wrapped information sources. The XIB depends heavily on XML-related techniques. More specifically, we use DTDs to model the input and output of each service, and XML to represent both input and output values. Based on such representations, the paper investigates service integration in the form of DTD integration, and studies query decomposition in the form of XML element decomposition. Within the proposed framework, it is easy to add or remove information services to a broker, thereby facilitating maintenance, evolution and customization of information brokers.

Keywords: web services, data integration, mediators, information brokers.

1 Introduction

The availability of information sources, services and deployed software agents on the internet is literally exploding. To find relevant information, users often have to manually browse or query various information services, extract relevant data, and fuse them into a usable form. To ease this kind of tedious work, various types of information agents have been proposed, including meta-searchers [36], mediators [15][4], and information brokers [12]. Such agents provide a virtual integrated view of heterogeneous information services, and perform a variety of tasks autonomously on behalf of their users.

Two issues are critical in building such software agents: extensibility and flexibility. The internet is an open and fast changing environment. Information sources, internet connections, and the information agents themselves may appear and disappear unpredictably, or simply change with no warning. Any software agent that operates within such an environment needs to be easily adaptable to the volatile internet environment. Likewise, in such an open environment there will always be new users who have different requirements for their information processing tasks. Under such circumstances, any

technology that is proposed for building information agents needs to support in a strong sense both customizability and evolution.

The XIB (eXtensible Information Broker) is a framework intended to facilitate the construction of information brokers that meet such extensibility and customizability requirements. The basic idea of the framework is to make web services that are currently only available to users, also accessible from within other applications. To enable this, we define an XML-based service description language called the XIBL. More specifically, the input and output descriptions for each service are represented as DTDs, while input and output data are represented as XML elements. Due to the widespread adoption of XML notation, and the extensibility of XML itself, the XIBL is flexible enough to describe various services including web services, databases, and even Java remote objects.

The XIB framework recognizes and accommodates three types of users: wrapper engineers, broker engineers, and end-users. Wrapper engineers are responsible for wrapping up a service in terms of the XIBL, and register the service in a service server. Broker engineers select services from the service server, and define the logic of an integrated service composed from existing services. Finally, end users use the brokers to access information.

To support these users, the XIB offers the *WrapperBuilder* and *BrokerBuilder* tools. *WrapperBuilder* is a visual tool that helps a user wrap a service interactively. The result of a session with *WrapperBuilder* is a service description expressed in the XIBL and registered in a service server. *BrokerBuilder* is a visual tool that interacts with users to define a new broker. Broker engineers are allowed to select from a service server the services they want to integrate, and to define the logic of the integration. The outcome of a session with *BrokerBuilder* is a broker that will typically accept complex queries, rewrite them into sub-queries to be handled by individual services, and compose the results of these sub-queries into a coherent response to the user query. As well, facilities are provided so that brokers can replace a source that is out-of-service with the help of a matchmaking capability of the service server. More details of the system can be found at http://www.cs.toronto.edu/km/xib.

In the following sections, we first describe a typical scenario we propose to address with the XIB, as well as its overall architecture. Then we introduce the information service description language (XIBL for short), which allows the description of websites or databases. Section 4 offers details on how a broker engineer defines or customizes an information broker, based on a set of information service descriptions. Wrapper construction is described next, while Section 6 discusses query result composition. The paper concludes with a review of the literature and a summary of the key issues that have been addressed.

2 The Problem

Suppose we want to buy books from the web. There are many websites that provide such services, notably `www.amazon.com` based in the United States and `www.globeChapters.com` based in Canada. Our objective is to find a website that provides the best price for the books we are interested in. Accordingly, we browse the two websites, retrieve the prices of these books in each website, and go to the currency converter service provided in `www.xe.net/currency` to convert the prices from USD

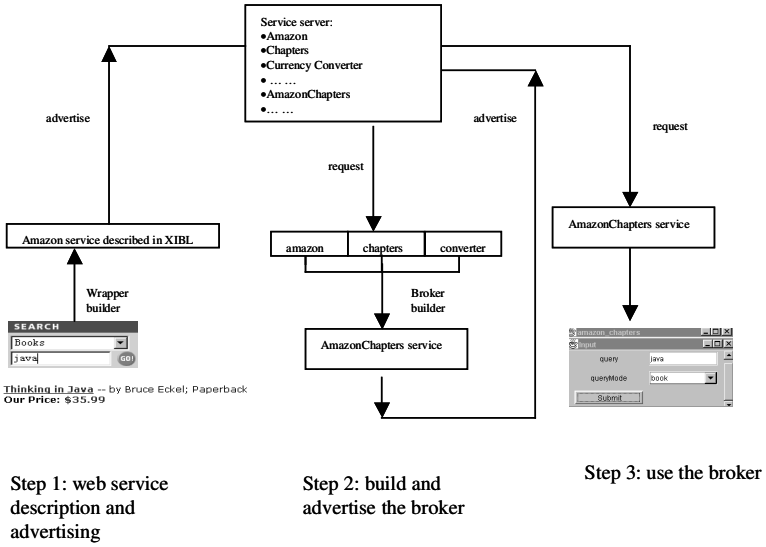

Figure 1: Overall architecture

to CAD for each book. In doing so, we are shifting back and forth between different websites and need pencil and paper to do the comparisons.

There should be a better way to do this. There are information broker websites that will automatically collect information from different places and find a suitable solution for us based on user-defined criteria. In such websites, the user specifies the criteria for selecting a commodity, and the broker presents a prioritized listing of products from different vendors.

Unfortunately, most broker websites suffer from two problems. Firstly, they require a lot of maintenance to keep data from different sources up-to-date. A broker website does not have any data of itself. It simply offers collections of information retrieved from vendor websites. Since web services are highly volatile, such hardcode brokers break frequently and need maintenance. A better way to accomplish this integration of information would be to have vendors publish their service in a well-known format so that brokers can automatically obtain it and collate it.

Secondly, most broker websites are programmed manually. Tasks such as source selection, querying, and assembling of retrieved data are all hardcoded into programs. Such hard-coded brokers cannot satisfy a diversity of user needs.

To overcome the first problem, we use XML to encode the information provided from different sources, and use an XML schema to describe the type of inputs and outputs for each source. There is a wrapper for each information source that can transform the HTML documents or database tables to XML documents. Thanks to the information contained in XML tags, it is easier for the data integrator to assemble returned results from each vendor website. For dynamic web services, the wrapper description specifies the queries that can be

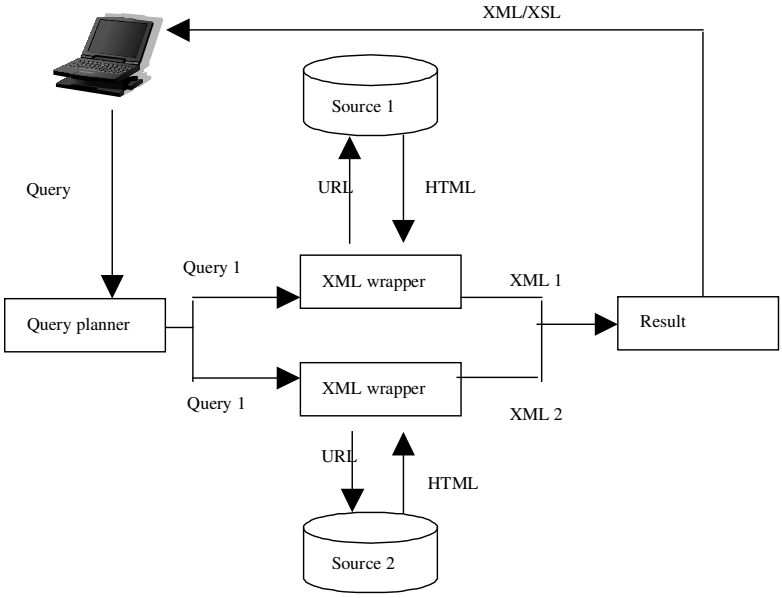

Figure 2: Broker usage

asked, the format of the results, also the location of the service. In particular, the description includes directions for extracting relevant information from HTML documents. Wrappers and XML integrators can be generated automatically from such descriptions, thereby enhancing the extensibility of XIB brokers. Obsolete service can be removed and new services can be added to a broker's domain of expertise whenever the need arises.

To overcome the second problem, the XIB provides a service server where users can select the services they want to integrate. Our framework uses a DTD inference mechanism to combine query interfaces, and an XML query language to integrate the query results. Returning to the book comparison example, we can easily add an additional online bookstore, `http://www.barnesnoble.com` or find a substitute for `www.xe.net` when it goes out of service. This means that when we try to extend the information broker, there is no need to manually modify the underlying program code.

The workflow of the three types of roles provided for in the XIB framework is presented in Figure 2. A *wrapper builder* describes an XML interface for a service and registers the service in the service server. A *broker engineer* is responsible for the selection of appropriate services to be integrated, and interacts with the XIB to come up with a set of definitions that customize the intended broker. These definitions include the output XML schema, the mappings between the tag names of the output XML and the tag names of the individual services provided by the wrappers, also the query interface presented in HTML form.

End users are the people who use the customized information broker defined by the broker engineer. As illustrated in Figure 2, the end user inputs queries through the query interface,

```
<XIB>
 <SERVICE NAME="AmazonSearch"/>
 <INPUT>
     <elementType id="query"> <string/> </elementType>
 </INPUT>
 <OUTPUT>
     <elementType id="author"> <string/> </elementType>
     <elementType id="title"> <string/> </elementType>
     <elementType id="publisher"> <string/> </elementType>
     <elementType id="year"> <string/> </elementType>
     <elementType id="price"> <string/> </elementType>
     <elementType id="book">
       <element type="#author" />
       <element type="#title" />
       <element type="#publisher" />
       <element type="#year" />
       <element type="#price" />
     </elementType>
     <elementType id="books">
       <element type="#book" occurs="ZEROORMORE" />
     </elementType>
 </OUTPUT>
 <INPUTBINDING>
   <BASE method="POST" action="cgi-bin"> http://www.amazon.com</BASE>
   <BINDING variable = "query" mapsTo="keyword-query" />
 </INPUTBINDING>
 <OUTPUTBINDING>
   <script>
    titles = Elem(P, "a") inside Elem(P,"dt");
    dd     = Elem(P,"dd");
    title  = Text(titles[i]);
    ... ...

   </script>
 </OUTPUTBINDING>
 <DESCRIPTION> search for book information from Amazon.
 </DESCRIPTION>
</XIB>
```

Figure 3: Amazon Search

and gets an integrated response that contains the results of several sub-queries. The XIB is responsible for the decomposition of the query submitted by a user, the transmission of the query to relevant services, the extraction and transformation of data from HTML documents or databases to XML documents, and finally the composition of the XML documents into a single integrated result. During composition, the XIB may require additional information and produce new queries for XML wrappers. For example, in the book comparison case, an additional query is generated to the currency exchange service.

3 The information service description language XIBL

Web services can generally be classified into three categories: static, dynamic, and interactive. Static web services simply offer static HTML web pages. Dynamic services typically allow users to provide input on a HTML form and get a dynamically generated

web page. Vanilla search engines are obvious examples of this kind of service. Interactive web services, on the other hand, constitute a special class of dynamic web services that allow state changes on the web server side and provide their service through multiple layers of interaction. E-commerce websites usually fall in this category.

This paper focuses mainly on dynamic web services. Such services are modeled as functions that specify the following:

- *What queries can be answered.* For a database, this is usually determined by the database query language (SQL or other); however, the queries that can be answered by a particular website are usually very limited. The XIB provides a grammatical notation for specifying the set of queries that can be submitted.
- *What information it returned*. This is the output of the broker service, specified in terms of a XML DTD.
- *Where is the service*. For our purposes, this may be the URL of a cgi script for a website, or the URL address of a database server.
- *Where is the data located*. For a database, this is specified by the database schema; for a website, on the other hand, pertinent data is usually hidden inside an HTML document, so we need to specify the exact location of those data.

We shall refer to these four components of a service description as INPUT, OUTPUT, INPUT BINDING, and OUTPUT BINDING, respectively. Figure 3 is an example of an Amazon search service description. We will explain the description in the following subsections. Please note that the more recent WSDL[38] is in many ways similar to the XIBL.

3.1 Input and output descriptions

Information sources usually only allow a limited number of query forms to be submitted. The input description defines the set of queries acceptable to a particular service. It consists of a set of variables that a user can specify values for, and their corresponding range specification. One design goal for such descriptions is the ability to model an HTML form, so that a description can be generated from an HTML form or vice versa.

Definition 1 (*Web service*) A Web service is a tuple $S(I, O)$, where S is the service name, I and O are respectively the input and output types, respectively, in the form of an XML schema.

An input description takes the form of an XML schema expressed in XML-data [20][37], an extension of the XML DTD that can be embedded in an XML document. Figure 4 shows a second, more complex, input/output description. Its input part specifies that the variable `model` can take as value any string (which corresponds to the *text input* control in an HTML form), while the variable `cpu` can take values `PII350` or `PII400` (which correspond to the *menus* control in the HTML form). In addition, the variable `memories` can take values `32M`, `64M`, or both (which corresponds to the *menus* control with multiple selections in an HTML form).

For the output descriptions, it is not sufficient to adopt a variation of the relational data model as proposed in [29]. Instead, we use a syntax that is similar to that of DTD to provide for the description of tree-like data structure.

In the example shown in Figure 4, the output consists of zero or more `computer` elements, each consisting of `cpu`, `memory`, `hardDisk`, `price`, and `address` elements. The `address` element, in turn, consists of two other elements, `mail` address and `email` address.

In Figure 3, the `INPUT` component is simply a `query` that can take an arbitrary string as its value. The OUTPUT component, on the other hand, declares that the result is `books`, and that `books` element consists of zero or a more `book` elements. In turn, each `book` consists of elements `author`, `title`, `publisher`, `year`, and `price`.

3.2 Input and output bindings

An input binding provides necessary information for the dynamic construction of an URL. For the example in Figure 3, it consists of the URL of the website in question, the cgi script name, and the mappings between the name used in the description and the attribute name used in the HTML form (the mapping from `query` to `keyword-query`).

The `output binding`, on the other hand, uses the markup algebra introduced in [24] to define the location of the data inside a HTML documents.

4 Synthesizing a broker

Once web descriptions are in place, a broker engineer can interact with the XIB to synthesize a broker as needed. First of all, the broker engineer needs to select a set of services to be integrated. The publication and selection of relevant services is not discussed in this paper. These can be delegated to a matchmaking agent [31], or adopt the more recently proposed XML-based UDDI approach[32].

To synthesize the broker, the broker engineer needs to specify three things. First, the user interface through which a query is submitted. Second, the output of the query, which consists of both the output format and the means for composing the results from each information source. Third, the mappings between the names in the broker and the names in each service. The following two subsections describe how the broker interface is derived and how the results are composed, while name mapping issues are discussed throughout these two subsections.

4.1 Derivation of the broker query interface

The broker query interface is an HTML form through which a user can submit queries. To derive the broker interface, first we must derive a broker XML schema from a set of input XML schemata, one for each service. Then we can generate an HTML form from the broker schema using XSL.

There are several requirements for the broker schema:

- *Generality.* The broker XML schema (DTD) should be capable of accepting queries (XML instances of the DTD) for every service. That is, every instance of each source XML schema should also be an instance of the broker schema.
- *Decomposability.* Every query acceptable by the broker schema (XML instance of the schema) should be decomposable into sub-queries that are acceptable to the services. In general, it is not desirable for the interface to let users submit queries that always fail to produce answers.

```
<SERVICE NAME="computerSearch1">
<INPUT>
  <elementType id="model"> <string/> </elementType>
  <elementType id="cpu">
     <attribute name="cpuValue" atttype="ENUMERATION"
          values="PII350 PII400" /
  </elmentType>
  <elementType id="memory">
    <attribute name="memoryValue" atttype="ENUMERATION"
          values="32M, 64M" />
  </elementType>
  <elementType id="memories">
    <element type="#memory" occurs="ONEORMORE"/>
  </elementType>
</INPUT>
<OUTPUT>
 <elementType id="computers">
   <element type="#computer" occurs="ZEROORMORE" />
 <elementType id="computer"/>
    <element type="#cpu"/>
    <element type="#memories"/>
    <element type="#hardDisk"/>
    <element type="#price"/>
    <element type="#address"/>
 </elementType>
 <elementType id="hardDisk">
    <attribute name="hardDiskValue" atttype="ENUMERATION"
          values="6G 8G" />
 </elementType>
 <elementType id="price"> <string/> </elementType>
 <elementType id="address"/>
    <element type="#mail"/>
    <element type="#email"/>
 </elementType>
</OUTPUT>
```

Figure 4: Computer Search 1

- *Minimality*. The schema should not be redundant in the sense that the same element type or attribute name and value will not be defined twice. Since each schema element type or attribute will be transformed into an HTML form control, multiple definitions of an element or an attribute will require a user to duplicate the action to set a value in several places. Besides, to ensure the validity of the schema, multiple definitions of a single element type should be removed.

Based on those requirements, we define the broker schema as follows:

Definition 2 (Broker Schema) Given two web services *S1(I1, O1)*, and *S2(I2,O2)*, a broker input schema is defined as a sequence of unions and sequential compositions of *I1* and *I2*.

The union (denoted as $\oplus$) and sequential composition operators (denoted as $\otimes$) on DTDs are defined as follows. Remember that we use the terms DTD and XML Schema interchangeably. While XML Schema is a new standard and is encoded in XML, DTD is more succinct and easy to discuss.

```
<SERVICE NAME="computerSearch2">
<INPUT>
  <elementType id="cpu">
    <attribute name="cpuValue" atttype="ENUMERATION"
          values="PII266 PII350" />
  </elmentType>
  <elementType id="hardDisk">
    <attribute name="hdValue" atttype="ENUMERATION" values="4G 8G"/>
  </elementType>
</INPUT>
<OUTPUT>
  <elementType id="computers">
    <element type="#computer" occurs="ZEROORMORE" />
  </elementType>
  <elementType id="computer"/>
    <element type="#model"/>
    <element type="#cpu"/>
    <element type="#memories"/>
    <element type="#hardDisk"/>
    <element type="#price"/>
  </elementType>
  <elementType id="memory">
    <attribute name="memoryValue" atttype="ENUMERATION"
          values="32M 64M 128M" />
  </elementType>
  <elementType id="price"> <string/> </elementType>
  <elementType id="model"> <string/> </elementType>
</OUTPUT>
```

Figure 5: ComputerSearch2 input/output description

Definition 3 (DTD transformation) Let α, β ... denote element declarations, A, B, C, ... denote element types or content models, $\equiv$ denote syntactic equivalence. The DTD element declaration <!ELEMENT A (B)> is abbreviated as $A \leftarrow B$.

Union 1

$$\frac{\alpha \equiv \alpha_1\ (A \leftarrow B)\ \alpha_2,\ \beta \equiv \beta_1\ (A \leftarrow C)\ \beta_2}{\alpha \oplus \beta = (\alpha_1\ (A \leftarrow B \mid C)\ \alpha_2) \oplus (\beta_1 \beta_2)}$$

Union 2

$$\frac{\alpha \equiv \alpha_1\ (A \leftarrow B)\ \alpha_2,\ \beta \equiv \beta_1\ (C \leftarrow D)\ \beta_2}{\alpha \oplus \beta = (\alpha_1\ (A \leftarrow B)\ (C \leftarrow D)\ \alpha_2) \oplus \beta_1\ \beta_2}$$

Sequential 1

$$\frac{\alpha \equiv \alpha_1\ (A \leftarrow B)\ \alpha_2,\ \beta \equiv \beta_1\ (A \leftarrow C)\ \beta_2}{\alpha \otimes \beta = (\alpha_1\ (A \leftarrow (B, C))\ \alpha_2) \otimes (\beta_1 \beta_2)}$$

Sequential 2

$$\frac{\alpha \equiv \alpha_1\ (A \leftarrow B)\ \alpha_2,\ \beta \equiv \beta_1\ (C \leftarrow D)\beta_2}{\alpha \otimes \beta = (\alpha_1\ (A \leftarrow B)(C \leftarrow D)\ \alpha_2) \otimes (\beta_1 \beta_2)}$$

```
<elementType id="model"> <string/> </elementType>
 <elementType id="cpu">
   <attribute name="cpuValue" atttype="ENUMERATION"
          values="PII266 PII350 PII400" />
 </elmentType>
 <elementType id="hardDisk">
   <attribute name="hdValue" atttype="ENUMERATION" values="4G 8G"/>
 </elementType>
 <elementType id="memories">
   <element type="#memory" occurs="ONEORMORE"/>
 </elementType>
 <elementType id="memory">
   <attribute name="memoryValue" atttype="ENUMERATION"
          values="32M 64M" />
 </elementType>
```

Figure 6: Computer search broker input XML schema

Once we derived a DTD using the union and sequential composition operations, we need to simplify the content model that was derived. We define the following axiom system to simplify the content model of DTD.

Definition 4 (Content model axiom system) The axioms for the content model are as follows:

1) $A \mid (B \mid C) = (A \mid B) \mid C$
2) $A, (B, C) = (A, B), C$
3) $A \mid B = B \mid A$
4) $A, (B \mid C) = (A, B) \mid (A, C)$
5) $(A \mid B), C) = (A, C) \mid (B, C)$
6) $A \mid A = A$
7) $A+ = A^* \mid A$

The inference rule is the usual substitution rule for algebraic systems: Given that A=B, suppose A' is identical to A except each occurrence of C in A is replaced by D, also B' is identical to B except each occurrence of C in B is replaced by D. Then we can infer that A'=B'.

Notice that with the axioms above, if A and B are content models of DTD, we can deduce the following:

$A, A+ = A+$
$(A^*, B)^*, A^* = (A \mid B)^*$

By definition 2, 3 and 4, the broker input schema could be either *I1, I2*, or the sequential composition of *I1* and *I2*. Obviously, this schema satisfies conditions 1 and 2.

Let's generate a computer search interface from the descriptions in Figure 4 and Figure 5. To keep things simple, we suppose that the same entities in the two descriptions are denoted

```
CONSTRUCT
   <newbook>
       <author>$a </author>
       <title> $t </title>
       <AmazonPrice> $p3 </AmazonPrice>
       <GCPrice> $p2 </GCPrice>
   </newbook>
WHERE <book>
       <author>$a </author>
       <title> $t </title>
       <price> $p1 </price>
     </book> IN "http://cs.toronto.edu/XIB/amazonSearch"
          CONDITION "amazonSearch.INPUT.query=newSearch.INPUT.queryString"
AND
  <chapterBook>
       <authors>$a </authors>
       <BookName> $t </bookName>
       <ourPrice> $p2 </ourPrice>
  </chapterBook> IN "http://cs.toronto.edu/XIB/globeChaptersSearch"
       CONDITION
"globeChaptersSearch.INPUT.query=newSearch.INPUT.queryString"
AND
  <Converter>
     <amount> $p1 </amount>
     <result> $p3 </result>
  </Converter> IN "http://cs.toronto.edu/XIB/ConverterService"
     CONDITION "ConverterServive.INPUT.from=USD;
               ConverterService.INPUT.to=CND;
               ConverterSerivce.INPUT.amount=$p1"
```

Figure 7: XML template

by the same name. When they are denoted by different names, the broker engineer needs to construct a mapping between those names.

By integrating the XML schemata in `computerSearch1` and `computerSearch2`, we can produce the `INPUT` description as shown in Figure 6. We notice that in the broker schema allowable values for `cpuValue` are obtained by combining the corresponding allowable values from `computerSearch1` and `computerSearch2`. The correspondence between this XML schema and an HTML form is as follows. Element type `model` will produce an HTML *text input* control, element types `cpu` and `hardDisk` will produce *menus* controls, and `memories` will produce *menus* control that allows for multiple selection.

4.2 Definition of the output XML template

When the services are selected, there are numerous ways to integrate them. Given the two services `GlobeChaptersSearch` and `AmazonSearch`, we can proceed to search for books offered on either site, or only for books that can be shipped within 24 hours, etc. Here we are concerned with price comparison for available books. Hence the broker engineer needs to interact with the XIB to define the output XML template.

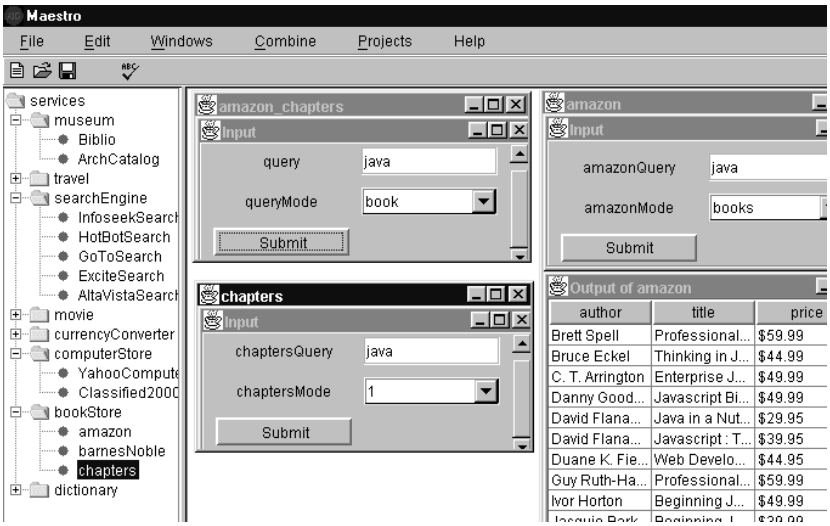

Figure 8: User interface

Given the output XML schemata for `AmazonSearch` and `GlobeChaptersSearch`, the broker engineer can define the output XML template as shown in Figure 7.

This kind of template uses a simplified form of the XML query language XML-QL [13]. The major difference in the syntax of the XIB templates is that the `IN` clause contains wrapper information, instead of an URL that points to an XML file. The `Construct` component of the template defines the intended output, i.e., a list of `<newbook>` elements that consist of elements `<author>`, `<title>`, `<AmazonPrice>`, and `<GCPrice>`. Finally, the `WHERE` clause specifies how to compose results from different information services. Note that the strings preceded by the sign $ denote variables. In this example, the `AmazonSearch` and `globeChaptersSearch` are joined on the `author` and `title`, and the `AmazonSearch` and `ConverterService` are joined on the price (amount) in USD. To calculate the currency conversion between US and Canadian dollars, we need to use another information service `ConverterService`.

Once the mapping and the template are defined, a broker is generated and can be used in the same way as other services. Figure 8 shows the user interface for selecting services, integrating them into a new broker, also the broker is used to query data.

5 Wrapper Generation

Wrapping a system is the process of defining and restricting access to the system through an abstract interface. A wrapper for information services accepts queries in a given format, converts them into one or more commands or sub-queries understandable by the underlying information service and transforms the native results into a format understood by the application. In the following we discuss wrapper generation for websites and relational databases.

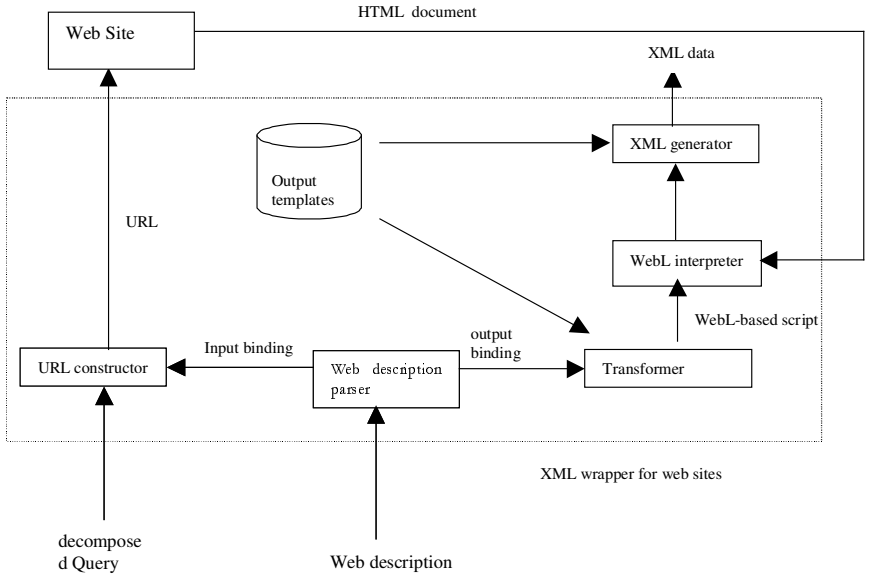

Figure 9: XML wrapper generation

5.1 Wrappers for dynamic web services

A wrapper for a web-based information source is a special kind of wrapper in that the underlying source involves websites and applications, while the native results are usually in the form of HTML documents. The basic functionality of such wrappers includes accepting a query and constructing the corresponding URL; also accessing a web page given a URL, extracting the relevant information and returning the resulting XML DOM object to the broker.

Once the output XML template is defined, a wrapper is generated for each service. Wrappers are not developed a priori for an information source because such a source can offer a vast array of services, while different broker or mediator will only use a specific part of the service. For example, the `AmazonSearch` service also provides information on `publisher, publish year` etc. However, since our broker will not need this information, there is no need for the XIB wrapper to produce it.

The process of generating XML wrappers for websites is described in Figure 9.

The description parser parses web descriptions written in the XIBL. Given input bindings, the *URL constructor* generates the URL. Suppose that for the `AmazonSearch` example, the query string to the user interface is "`XML`", meaning that the user wants to search for books about XML. The generated URL for this input is `http://www.amazon.com/cgi-bin?keyword-query="XML".`

```
<SERVICE NAME="ConverterDBService">
<INPUT>
  <elementType id="amnt"> <string/> </elementType>
  <elementType id="frm">  <string/> </elementType>
  <elementType id="to">   <string/> </elementType>
</INPUT>
<OUTPUT>
   <elementType id="conversion">
      <element type="#amount"/>
      <element type="#result"/>
      <element type="#from"/>
      <element type="#into"/>
   </elementType>
</OUTPUT>
<INPUTBINDING>
  <BASE> jdbc:msql://mika.ai.toronto.edu:1114/converterDB </BASE>
  <DRIVER> com.imginary.sql.msql.MsqlDriver </DRIVER>
  <USER> guest </USER>
  <PASSWORD> 12345 </PASSWORD>
  <DBQUERY>
    SELECT amount, result, from, into
    FROM  ConversionTable
    WHERE amount=amnt AND from=frm AND into=to
  </DBQUERY>
</INPUTBINDING>
```

Figure 10: Database description

Along similar line, the Transformer is fed an output template, output bindings, and input/output descriptions, and generates the *WebL[24]* scripts that can extract the relevant data. For our example, according to the `AmazonSearch` description, we can get information on publisher and publication year. However, since this information is not needed according to the output template, our *WebL* scripts won't extract it. The *WebL interpreter* will interpret the *WebL* scripts and extract pertinent data. Finally, the *XML generator* transforms the output of the WebL interpreter to the XML format according to the output XML schema.

5.2 From databases to XML

To allow for data exchange between databases and websites we need to be able to build XML wrappers for database systems. For our book comparison example, suppose there is another converter service provided by a relational database system based on the description shown in Figure 10. In this description the format for `INPUT` and `OUTPUT` specifications is the same with other descriptions. Inside the binding parts, there are definitions for the database URL, the database driver, user account, password, and the database query.

6 Query Planning and Result Composition

Given a query from the broker interface, the broker needs to decompose the query and form a plan to execute the query. Let's focus at the computer hardware example. Suppose that the following query is submitted through the broker input interface specified in Figure 6:

`Q= <cpu>PII350</cpu> <hardDisk>4G</hardDisk><memory>32M</memory>.`

This query requests a search for hardware information satisfying the constraints that `cpu` is `PII350`, `hardDisk` is `4G`, and memory is `32M`.

First, the broker needs to decide which service is capable of accepting this query.

Definition 5 (Acceptable query) A query Q is acceptable to a service $S(I, O)$, if either Q is an instance of I, or there is a decomposition of $Q=(Q_1, Q_2)$, such that Q_1 is an instance of I, and Q_2 is an instance of part of O.

Since Q is not an instance of the input schema of `computerSearch2` for our example, it is decomposed into

Q_1= `<cpu> PII350 </cpu> <hardDisk> 4G </hardDisk>`

and

Q_2= `<memory>32M</memory>`.

Q_1 and Q_2 are respectively instances of the input and output schemata of `computerSearch2`. Hence this query is acceptable to `computerSearch2`. Once such a decomposition is obtained, Q_1 is sent to `computerSearch2` and Q_2 is used as a filter condition within the broker.

The task of result composition is facilitated by the output template provided by the broker engineer. In our implementation, we transform the output template to the XML-QL query with some variables instantiated and the service name replaced by a concrete XML document. Then using the XML-QL engine, we generate the resulting XML document.

One complication that can arise here is that some input variables may not be instantiated at the time a service is initialized. In our book search example, the converter can only be activated after the `AmazonSearch` is completed, i.e., when the value for the variable `$P1` is available from the `AmazonSearch`. Thus the result composer needs to wait for that value, assign that value to the input variable of the `ConverterService`, generate a new query, and send it to the wrapper.

7 Related Work

7.1 Information integration

Much work has been done in query planning and rewriting in federated databases and information mediators [21][15]. Most of those systems are based on relational or extended relational data models, while the XIB is based on XML. The use of XML within the XIB is more than mere notational convenience. Firstly, each information source is modeled as a function, which accepts XML documents as inputs and produces XML documents as outputs. The signature of the function is described by the input and output DTDs. In federated database systems and information mediators each information source is typically described by a local schema, and possibly a subset of the query language that the local database allows. Secondly, we discussed the integration of XML data instead of relational data. Moreover, there has been little research on deducing mediator views from local schemata.

```
CONSTRUCT
 <newComputer>
   <cpu> $cpu </cpu>
   <memory> $mm </memory>
   <hardDisk> $hd </hardDisk>
   <price> $cndPrice </price>
   <address> $addr </address>
 </newComputer>
WHERE
(
  <computer1>
    <cpu> $cpu </cpu>
    <memory> $mm </memory>
    <hardDisk> $hd </hardDisk>
    <price> $cndPrice </price>
    <address> $addr </address>
  </computer1> IN "http://www.toronto.edu/XIB/computerSearch1"
     CONDITION "computerSearch1.INPUT.cpu=newComputer.INPUT.cpu;
      computerSearch1.INPUT.mm =newComputer.INPUT.mm;
      $hd=newComputer.INPUT.hd"
OR
  <computer2>
    <cpu> $cpu </cpu>
    <memory> $mm </memory>
    <price> $usdPrice </price>
    <address> $addr </address>
  </computer2> IN "http://www.toronto.edu/XIB/computerSearch2"
       CONDITION "computerSearch2.INPUT.cpu= newComputer.INPUT.cpu;
         computerSearch2.INPUT.hd = newComputerSearch.INPUT.hd;
         $mm = newComputerSearch.INPUT.mm"
)
AND
  <converter>
    <amount> $usdPrice </amount>
    <result> $cndPrice </result>
  </converter> IN "http://www.toronto.edu/XIB/converterService"
     CONDITION "converterService.INPUT.frm=USD;
          converterService.INPUT.to=CND;
          converterService.INPUT.amnt=$usdPrice"
```

Figure 11: Computer search template

More recently, there has been several XML-related projects that address similar issues to the XIB. MIX [4] is an XML-based DTD driven mediator prototype typical of this line of research. In MIX, data exchange and integration is based on XML. The XML query language XMAS is used to define the integration view, and the graphical user interface BBQ (Blended Browsing and Querying) is used to generate complex queries driven by the mediator view DTD. The view DTD is derived from the view definition and source DTDs. MIX is not comparable with the XIB in that MIX mediates between static web pages while the XIB integrates dynamic services that are modeled as functions.

Meta-wrappers[34] are components within mediator architecture that decompose user queries and compose wrapper responses. The assumption here is that we are dealing with a large number of information sources in a dynamic web setting. Based on this assumption, it is important to group similar information sources, and generate a non-redundant, least-cost plan for a given query. An extension of the relational data model is used to describe the

source and meta-wrapper schemata, while an input/output relation describes the limited capability of a web source. Unlike the XIB, where the input and output could be XML schemata, the input and output of a meta-wrapper are sets of attributes. Moreover, the meta-wrapper assumes the existence of wrappers that are responsible for the direct access to information sources and the translation from different data model to a uniform data model. The source description in a meta-wrapper does not include the `BINDING` part as in the XIB.

7.2 Web service description, advertisement, and discovery

WIDL[35] may be the first XML application that tries to describe web services so that they can be accessed by applications. In WIDL, the input and output are simply described as a set of variables that have no structure, hence WIDL is unable to support the integration of new query interfaces and result composition that is based on a particular schema.

More recently, there have been industrial efforts focusing on web service description, advertisement, and discovery. For example, WSDL[11] describes services, SOAP[6] helps the XML-based remote procedure invocation, and UDDI [32] facilitates the service registration and discovery. These efforts focus more on access details, instead of service integration. Moreover, such emerging industrial standards prove the necessity of the XIB framework, and pave the way for wider adoption of XIB-like applications. In particular, when WSDL is widely adopted for service description, the wrapper construction bottleneck described in this paper will simply be eliminated.

The description of agent capabilities [5][31] has been another active research area. Using semantic description and inference mechanism, those approaches aim at a higher level of automation and intelligence than our proposal.

WebSemantics [23] proposes architecture for describing, publishing, registering, discovering, and accessing relevant data over the internet using XML and XML-data. Unlike the XIB, the focus of this work is on data instead of the services available on internet.

7.3 Wrapper construction

Wrapper construction or generation is the task of producing wrappers from source descriptions. Approaches to wrapper construction vary with respect to the level of abstraction of the input/output descriptions, the expressiveness of the descriptions, also the degree of tool support for the acquisition of the input/output descriptions and the transformation from the descriptions to executable code.

Input/output descriptions could be represented in declarative languages [35]. They could also be represented as executable scripts [24]. The input/output descriptions could be provided manually [35], or obtained with the help of tools[29]. Alternatively, they may be induced automatically using machine-learning techniques [26][17]. Our work has not addressed yet the important issue of acquiring automatically or semi-automatically such descriptions.

Compared to other wrapper construction proposals, our description language is quite expressive. By using XML schemata to describe inputs and outputs, we allow for the description of complicated queries and richly structured outputs. This is particularly important in view of the increasing complexity of HTML forms.

7.4 Collaborating software agents

There are software agent frameworks that support advertising and requesting of software agents, and the collaboration between them. There are also agent capability advertising and description languages such as KQML, FIPA, and DAML.

The XIB bridges the gap between distributed agents and industrial applications by focusing on one specific application domain, web services. In so doing, we have proposed a specialized capability description language and have developed an elaborate approach to collaboration and integration.

8 Conclusions

Thanks to the widespread use of web based information systems, and the introduction of industrial standards such as WSDL, UDDI, and SOAP, there is a growing demand for integrating web services dynamically.

Unlike many information integration systems[15][13][21][27][29] that have their roots in heterogeneous database systems, we adopt a different approach where each information source is described as a function, rather than a database schema. Moreover, such functions take XML documents as input and produce XML documents as output.

Key design features of the proposed eXtensible Information Broker (the XIB) are its extensibility and flexibility. Given information source descriptions, broker engineers can build and maintain reliable information brokers with ease. Building a new information broker only involves a few steps of interaction with the XIB. Moreover, the evolution and maintenance of such brokers is facilitated by the fact that one can easily add or remove sources or services without consulting the broker code. Finally, information brokers built through the XIB can be designed for reliability by including several redundant sources and services. For example, the broker engineer can prepare several currency conversion services. If one of them is unavailable due to a bad internet connection, other services can be used instead.

The main contributions of this work are as follows. Firstly, we propose a language and a tool to model dynamic web services in terms of XML. Secondly, we provide a platform for making web services accessible not only to users, but also to applications. Thirdly, as an example of such an application, we provide the *BrokerBuilder* that can integrate such services.

A prototype of the XIB has been implemented and can be accessed at http://www.cs.toronto.edu/km/xib. The implementation uses the XML parser XML4J. XSL is used to transform XML documents to HTML presentation, and the XML-data to HTML forms. Also, XML-QL is used to compose results. To construct wrappers for web sources, we used WebL to extract relevant information from an HTML document. For database wrappers, currently we are wrapping miniSQL databases and use JDBC to make the connection. Finally, the service server is implemented using Java RMI, while brokers are served using servlets.

We have experimented with the implemented the XIB framework through four groups of examples. One group is concerned with the integration of bookstore information from sources such as *amazon*, *globalChapters*, along with currency converter services. The second group of services includes generic search engines like *altaVista*, *hotBot*, etc. By

integrating this group of services, we provide new services similar to popular meta-search engines, such as *metacrawler*. The third group integrates movie review and local theater information services. This experiment was intended to try out the XIB with complementary, rather than similar services. The fourth group concerns the integration of information from several computer stores. This group of services required more sophisticated input/output formats, and allowed us to experiment with complex input DTD integration and query decomposition.

Several issues need to be investigated for this framework:

- *The vocabulary problem* Although XML provides a primitive layer of semantics for the data, this is not adequate for dealing with problems of synonymy, homonymy and the like. This problem is particularly important in view of the fact that service descriptions may be provided by different people, resulting in several DTDs for the same task. In our current implementation, broker engineers have to specify the mappings between the DTD tag names. This is one of the bottlenecks for the automatic generation of brokers. This problem will be tackled in two directions. One is to follow industry standards that are emerging, like RosettaNet and BizTalk. Another is to compute the distance between tag names based on WordNet and statistic methods like co-occurrence and trigger pairs.
- *XML Schema inference* There are many issues in the area of XML Schema inference that remain unsolved, such as the subsumption[18] and the combinations of XML Schemas.
- *Interactive web services* Our current framework can only model and integrate dynamic web services. There are an increasing number of web services that support several layers of interaction, and need to modify the state of the web server. The XIB is not able cope with this kind of services yet.

Acknowledgments

The project was funded by the Government of Canada through the Networks of Centers of Excellence and the Institute of Robotics and Intelligent Systems. We would like to thank the anonymous referees for their pertinent and helpful comments, also Jamie Ho and Kenneth Sinn for their help in the implementation of the XIB system.

References

[1] Jose' Luis Ambite and Craig A. Knoblock. *Planning by Rewriting: Efficiently Generating High-Quality Plans*. Proceedings of the Fourteenth National Conference on Artificial Intelligence, Providence, RI, 1997.

[2] Naveen Ashish, Craig Knoblock, *Semi-automatic Wrapper Generation for Internet Information Sources*, Second IFCIS Conference on Cooperative Information Systems (CoopIS), Charleston, South Carolina, 1997.

[3] P. Atzeni, G. Mecca, P. Merialdo, *Semistructured and Structured Data in the Web: Going Back and Forth,* In SIGMOD Record, Special Issue on the Workshop on the Management of Semistructured Data, 1997.

[4] C. Baru, A. Gupta, B. Ludaescher, R. Marciano, Y. Papakonstantinou, P. Velikhov, *XML Based Information Mediation with MIX*, In Exhibitions Program of ACM SIGMOD 99.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila, *The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*, Scientific America, May 2001.

[6] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, et al, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, May 2000.

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, *Extensible Markup Language (XML) 1.0,* W3C recommendation, http://wwww.w3.org/TR/REC-xml, 1998.

[8] Chen-Chuan K. Chang, Hector Garcia-Molina, Andreas Paepcke, *Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System.* ACM Transactions on Information Systems, vol. 17, no. 1, Jan. 1999.

[9] Chen-Chuan K. Chang, Hector Garcia-Molina, *Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources*. Proc. of the 1999 ACM SIGMOD International Conference On Management of Data, Jun. 1999.

[10] James Clark, Stephen Deach, *Extensible Stylesheet Language (XSL)*, W3C working draft, http://www/w3/org/TR/WD-xsl, 1998.

[11] Christensen E et al, *Web Services Description Language (WSDL) 1.0*, IBM/Microsoft Joint Working Document, see http://www-4.ibm.com/software/developer, 2001.

[12] K. Decker, K. Sycara, M. Williamson. *Matchmaking and Brokering*. Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), Dec-96.

[13] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, Dan Suciu, *XML-QL: A Query Language for XML,* W3C note, http://www.w3.org/TR/NOTE-xml-ql, 1998.

[14] Denise Draper , Alon Y. Halevy , Daniel S. Weld , *The Nimble XML Data Integration System,* Proc. of ACM SIGMOD Conf. on Management of Data 2001.

[15] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G., Philpot, and Sheila Tejada. *Modeling web sources for information integration,* Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, 1998.

[16] David Konopnicki, Oded Shmuedi, *A comprehensive framework for querying and integrating WWW Data and services*, Fourth IFCIS International Conference on Cooperative Information Systems, Edinburgh, 1999.

[17] Bruce Krulwich, *Automating the Internet Agents as User Surrogates*, IEEE Internet computing, Vol. 1, No. 4, 1997.

[18] Gabriel M. Kuper, Jérôme Siméon, *Subsumption for XML Types,* ICDT, 2001.

[19] Nicholas Kushmerick, Daniel Weld, Robert Doorenbos, *Wrapper induction for information extraction*, IJCAI'97.

[20] Andrew Layman, et al, *XML Data*, W3C note, http://www.w3.org/TR/1998/NOTE-XML-data-0105.

[21] Alon Y. Levy, Anand Rajaraman and Joann J. Ordille, *Querying Heterogeneous Information Sources Using Source Descriptions,* in Proceedings of the 22nd International Conference on Very Large Databases, VLDB-96, Bombay, India, September, 1996.

[22] Jianguo Lu, John Mylopoulos, Jamie Ho, *Towards Extensible Information Brokers Based on XML*, CAiSE*00, 12th Conference on Advanced Information Systems Engineering, Stockholm, June 7 - 9 2000.

[23] Mihaila, George and Raschid, Louiqa, *Locating Data Repositories using XML*, W3C Workshop on XML and Querying the Web, 1998.

[24] Hannes Marais and Tom Rodeheffer. *Automating the Web with WebL.* In Dr. Dobb's Journal, January 1999.

[25] Makoto Murata, *Automatically Constructing the Intersection/Union/Difference of Two Schemas,* XTech'99, march 7-11, 1999.

[26] Muslea and S. Minton and C. Knoblock, *STALKER: Learning Extraction Rules for Semistructured, Web-based Information Sources*, AAAI-98 Workshop on AI and Information Integration, 1998, 74-81.

[27] Y. Papakonstantinou, A. Gupta, L. Haas, *Capabilities-Based Query Rewriting in Mediator Systems*, in Proceedings of 4th International Conference on Parallel and Distributed Information Systems. Miami Beach, Florida, 1996.

[28] G. Piccinelli and S. Lynden. *Concept and tools for e-service development*. In 7th Workshop HP Openview Univesity Association (OVUA'00), June 2000.

[29] Raschid, Louiqa and Vidal, Maria Esther and Gruser, Jean-Robert. *A Flexible Meta-Wrapper Interface for Autonomous Distributed Information Sources*, Technical Report AR 309, University of Maryland, Institute for Advanced Computer Studies- Dept. of Computer Science, March 1997

[30] Arnaud Sahuguet, Fabien Azavant, *Wysiwyg Web Wrapper Factory (W4F),* 1999.

[31] K. Sycara, J. Lu, M. Klusch, S. Widoff, *Matchmaking among Heterogeneous Agents on the Internet*, in Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace, Stanford University, USA 22-24 March 1999.

[32] UDDI, *Universal Description, Discovery and Integration*, www.uddi.org.

[33] Vasilis Vassalos, Y. Papakonstantinou, *Expressive Capabilities Description Languages and Query Rewriting Algorithms*, http://www-cse.ucsd.edu/~yannis/papers/vpcap2.ps

[34] Maria Esther Vidal, Louiqa Raschid, Jean Robert Gruser, *A Meta-Wrapper for Scaling up to Multiple Autonomous Distributed Information Sources*, In Proceedings CoopIS'98.

[35] *WIDL: Application Integration with XML*, in "XML: Principles, Tools, and Techniques", the October print issue of O'Reilly's World Wide Web Journal, fall 1997.

[36] http://www.metacrawler.com.

[37] W3C XML Schema Working Group, *XML Schema*, www.w3.org/XML/Schema.

[38] W3C, *Web Services Description Language (WSDL) 1.1*, W3C Note 15 March 200.

[39] Ramana Yerneni, Chen Li, Hector Garcia-Molina and Jeff Ullman, *Computing Capabilities of Mediators*, SIGMOD, 1999.