Towards Extensible Information Brokers Based on XML

Jianguo Lu, John Mylopoulos, Jamie Ho

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

Abstract. The exponential growth in the number and size of information services available on the internet has created an urgent need for information agents which act as brokers in that they can autonomously search, gather and integrate information on behalf of a user. Moreover, the inherent volatility of the internet and the wide range of information processing tasks to be carried out, calls for a framework that facilitates both the construction and evolution of such information brokers. This paper proposes such a framework named XIB (eXtensible Information Brokers).
Based on descriptions of relevant information services, XIB supports the interactive generation of an integrated query interface, generates wrappers for each information service dynamically, and returns to the user the composed result to a query. XIB depends heavily on XML-related techniques. More specifically, we will use DTDs to model the input and output of the service, use XML elements to denote the input and output values. By using this representation, service integration is investigated in the form of DTD integration, and query decomposition is studied in the form of XML element decomposition. Within the proposed framework, it is easy to add or remove information services on the internet to a broker, thereby facilitating maintenance, evolution and customization.

Keywords: XML, data integration, interoperability, wrapper, multi-agent system, mediator, web-based information system.

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 an usable form. To ease this kind of tedious work, various types of information agents have been proposed, including meta-searchers [28], mediators [11][4], and information brokers [9]. These 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. To meet such demands, the technology we use to build information agents needs to support customizability and evolution.

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

There are three groups of users inside this framework, i.e., wrapper engineers, broker engineers, and end users. Wrapper engineers are responsible for wrapping up a particular service in terms of XIBL, and registering the service in a service server. Broker engineers select services from the service server, and build brokers where they define how to integrate the services. End users use the brokers.

Correspondingly, we provide *WrapperBuilder* and *BrokerBuilder* tools. *WrapperBuilder* is a visual tool that helps a user wrap up a service interactively. Through an interactive session, *WrapperBuilder* produces a service description written in XIBL and gets it registered in a service server. *BrokerBuilder* is a visual tool that interact with users to define a broker. The tool allows allow broker engineers to select from the service server the services they want to integrate, and to define the logic of the integration. Again, through an interactive session, a broker is generated automatically, without writing program code. Brokers will typically accept more complicated queries then any individual services, decompose the query into sub-queries, and compose the results from 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 matchmaking capability of the service server. More details of the system can be found at `www.cs.toronto.edu/km/xib`.

In the following we first introduce the information service description language XIBL, which allows the description of websites or databases. Next we describe how a broker engineer defines or customizes an information broker, based on a set of such information service descriptions. Wrapper generation is described next, while section 5 discusses result composition. The paper concludes with a review of the literature and a summary of the key issues addressed.

2 Information Service Description Language XIBL

We classify web services into three categories, i.e., static, dynamic, and interactive. Static web services are those static HTML web pages. Dynamic services typically allow users to provide input on a HTML form and get a dynamically generated webpage. One example of such web services is a generic search engine. Interactive web services are a special class of dynamic web services that allow for the change of the state on the web server side and accomplish the service through multiple layers of interaction. E-commerce websites usually provide interactive services.

```
<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>
```

Fig. 1. Amazon description

This paper focuses mainly on dynamic web services. This kind of service could be modeled as a function. There are four layers of description for such services:

1. 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.
2. What queries can it answer. For a database, this would usually be 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 needs to provide a grammatical notation for specifying the set of queries that can be submitted.
3. What information can it provide. This is the output data we expect from the service, specified in a XML DTD.
4. Where is the data exactly located. For a database, this is specified in 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 call these four components of a service description as `INPUT BINDING`, `INPUT`, `OUTPUT`, and `OUTPUT BINDING`, respectively. Figure 1 is an example of an Amazon search service description.

2.1 Input and Output Descriptions

Information sources usually allow only 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 associate values with, and their corresponding range specification. One design goal of the description is to model the HTML form, so that a description can be generated from an HTML form or vice versa.

The input description takes the form of an XML schema expressed in XML-data [15], an extension of the XML DTD that can be embedded in an XML document. Figure 2 shows an example of a more complicated input/output description. The input part here means that the input variable `model` can be any string (which corresponds to the *text input* control in an HTML form), `cpu` can take values `PII350` or `PII400` (which correspond to the *menus* control in the HTML form), and `memories` can take values `32M`, `64M`, or both (which corresponds to the *menus* control with multiple selections in an HTML form).

```
<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>
```

Fig. 2. computerSearch1 input/output description

For the output descriptions, it is not adequate to use a variation of the relational data model as in [26]. Instead, we also use a syntax similar to DTD to allow for the description of a tree-like data structure.

In the example shown in figure 2, 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 1, the `INPUT` component is simply a `query` that can take arbitrary strings as its value. The `OUTPUT` component, on the other hand, declares that the result is `books`, and that `books` consists of zero or a more `book`. Each `book` consists of elements `author`, `title`, `publisher`, `year`, and `price`.

```
<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>
```

Fig. 3. ComputerSearch2 input/output description

2.2 Input and Output Bindings

An input binding provides necessary information for the dynamic construction of an URL. For our example of figure 1, the input binding 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` uses the markup algebra introduced in [17] to define the location of the data inside a HTML documents.

3 Broker Synthesis

Once web descriptions are available, 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 can be handled by a matchmaking agent [24].

To synthesize the broker with selected services, there are three things to be defined by the broker engineer. 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 to compose 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 defined and how the results are composed, while name mapping issues are discussed throughout these two subsections.

3.1 Definition 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 via 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 to 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.

Normal Form The schema should be normalized so 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 definition of element types should be removed.

```
<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>
```

Fig. 4. Computer search broker input XML schema

The steps required to construct the broker input schema are as follows. Given two service input descriptions A and B, first construct the integrated schema (DTD) as <!ELEMENT X ((A|B)|(A,B))>, which means that the broker input schema could be either A, B, or the sequential composition of A and B. Obviously, this schema satisfies condition one. The next step is to apply a set of schema transformation rules to simplify the schema so that element types, attributes, or values inside the integrated schema are not defined in multiple places. Each transformation rule will preserve the equality of the schemas. The process continues until no transformation rule is applicable.

Let's look at the example of generating a computer search interface from the descriptions in figures 2 and 3. To keep things simple, we suppose that the same entities in the two descriptions are denoted 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 have produced the `INPUT` description as in figure 4. We notice that in the broker schema the valid values of the `cpuValue` is obtained by combining the corresponding valid 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.

3.2 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 use them to search for books that appear in both places, for books that can be shipped within 24 hours, etc. Here we are concerned with the comparison of the prices of the books in these two places. Hence the broker engineer needs to interact with the XIB to define the output XML template.

Given the output XML schemata for `AmazonSearch` and `GlobeChaptersSearch`, the broker engineer can define the output XML template as in figure 5.

```
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"
```

Fig. 5. XML template

This kind of template uses a simplified form of the XML query language XML-QL [10]. The major difference in the syntax of 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 which consist of elements `<author>`, `<title>`, `<AmazonPrice>`, and `<GCPrice>`. The `WHERE` part defines 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 do the currency conversion between US dollars and Canadian dollars, we need to use another information service `ConverterService` whose input and output definitions are as in figure 6.

4 Wrapper Generation

Wrapping a system is the process of defining and restricting access to a 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.

```
<SERVICE Name="ConverterService"/>
<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>
```

Fig. 6. Currency Converter

4.1 Wrappers for Dynamic Web Services

A wrapper for a web-based information source is a special kind of wrapper in that the 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 webpage given a URL, extracting the relevant information and returning the resulting XML DOM object to the broker.

When the output XML template is defined, the wrappers for each service will be generated dynamically. Wrappers are not developed a priori due to the fact that each information source has a vast array of services, while different broker or mediator will only use some of these service. For example, the `AmazonSearch` service also provides information on `publisher`, `publish year` etc. However, since the broker will not need this information, there is no need for the wrapper to produce it.

The process of generating XML wrappers for websites is described in figure 7.

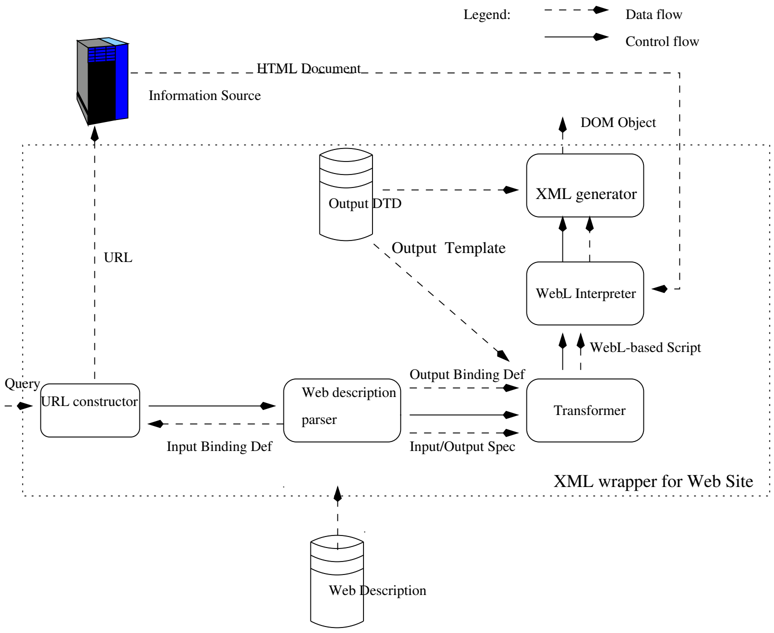

Fig. 7. XML Wrapper generation

The description parser parses the web description. From the input binding, the *URL constructor* is able to generate the URL to get the corresponding webpage in HTML format. In the `AmazonSearch` example, suppose the query string to the user interface is "`XML`", to search for books about XML, the generated URL is `http://www.amazon.com/cgi-bin?keyword-query="XML"`.

At the same time, given the output template, the output bindings, and the input/output descriptions, the *Transformer* will generate the *WebL* [17] scripts that can extract the relevant data. In our example, according to the `AmazonSearch` description, we can get information about publisher and publish year. However, since this information is not needed according to the output template, the generated *WebL* scripts won't extract it. The *WebL interpreter* will interpret the *WebL* scripts and extract the pertinent data. Finally, the *XML generator* transforms the output of the WebL interpreter to the XML format according to the output XML schema.

4.2 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 and its description is as in figure 8. In this description the `INPUT` and `OUTPUT` specifications are the same as in figure 6. Inside the binding part, there are definitions for the database URL, the database driver, user account, password, and the database query.

5 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 look at an example in the computer search case. Suppose that the following query is submitted through the broker input interface specified in figure 4:

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

This asks for the selection of computer 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.

Suppose I and O are the input and output schemata of service S. 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.

```
<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>
```

Fig. 8. Database description

In our example, since Q is not an instance of the input schema of computerSearch2, it is decomposed into

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

and

Q_2= <memory>32M</memory>,

which are instances of the input and output schemata of computerSearch2, respectively. Hence this query is acceptable to computerSearch2.

Once such decomposition is obtained, Q_1 is sent to the computerSearch2 and Q_2 is used as a filter condition inside the broker.

With the XML documents produced from the wrapper and the output template provided by the broker engineer, the task of result composition becomes easier. 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 can get the result XML document.

One complication that can arise here is that some input variables may not be instantiated beforehand. 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.

6 Related Work

6.1 Information Integration

Much work has been done on query planning and rewriting for information mediators[16]. Recently, XML-related issues have also been studied in this area. For example, MIX[4] is an XML-based DTD driven mediator prototype. In MIX, data exchange and integration relies 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 XIB in that MIX mediates between static web pagesm, while XIB is intended to integrate dynamic services that are modeled as functions.

Metawrappers [26] are components within a mediator architecture which decompose user queries and compose wrapper responses. The assumption here is that there are hundreds of information sources in a dynamic WWW environment, so there is a need 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 metawrapper schemata. An input/output relation is used to describe the limited capability of a web source. Unlike the XIB, where the input and output could be XML schemata, the input and output in a metawrapper are sets of attributes. Moreover, the metawrapper assumes the existence of wrappers which 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 metawrapper does not include the `BINDING` part in XIB.

WebSemantics[18] proposes an architecture to describe, publish, register, discover, and access relevant data over the internet using XML and XML-data. The focus is on data instead of the services available on the internet.

WIDL [27] is an XML application that tries to describe web sources so that they can interoperate. 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 interface and the result composition based on certain data schema.

6.2 Wrapper Construction

Wrapper construction or generation is the task of producing wrappers from source descriptions. The 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 a declarative language, e.g., [27][22]. They could also be represented as executable scripts [17]. The input/output descriptions may be provided manually[27], or obtained with the help of tools [22]. Alternatively, they may be induced automatically using machine learning techniques [20][13]. Our work has not addressed yet the important issue of acquiring automatically or semi-automatically such descriptions.

```
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"
```

Fig. 9. Computer search template

Compared to other wrapper construction proposals, our description language is very 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 Conclusions

The key design features of the proposed eXtensible Information Broker (XIB) is 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 several steps of interaction with the XIB. Moreover, the evolution and maintenance of such brokers is simple, since one can easily add or remove sources or services without consulting the code. Finally, information brokers built through the XIB can be built for reliability by including several redundant sources and services. For example, the broker engineer can prepare several currency conversion services in case one of them breaks due to a bad internet connection.

The main contributions of this work are as follows. Firstly, it provides a language and a tool to model dynamic web services in terms of XML. Secondly, it provides 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 XIB has been implemented and can be accessed at `www.cs.toronto.edu/km/xib`. The implementation adopted the XML parser XML4J developed by IBM. 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 the miniSQL database 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 XIB framework through four groups of examples. One group is concerned with the integration of book store 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 the popular metasearcher like *metacrawler*. The third group is to integrate movie review and local theater information services. This experiment is intended to try out the XIB with complementary, rather than similar, services. The fourth group concerns the integration of information from a set of computer stores. This group of services requires more complicated input and output format, and has allowed us to experiment with sophisticated input DTD integration and query decomposition.

Several issues need further investigation for this framework to scale up:

The vocabulary problem Although XML provides certain semantics for the data and hence facilitates the integration of services, given the assumption that service descriptions may be provided by different people, there may be different 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 bottleneck for the automatic generation of brokers. The 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 statistic methods like co-occurrence and trigger pairs.

Interactive web services Our current framework can only model and integrate dynamic web services. There are many web services that need several layers of interaction, and require the modification of the state on the web server side. XIB is not able to cope with this kind of service.

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 Kenneth Sinn for his contribution in the implementation of the system.

References

1. Vidur Apparao et al, Document Object Model (DOM) Level 1, W3C recommendation, http://www.w3.org/TR/REC-DOM-Level-1/.
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 Bray, Jean Paoli, C. M. Sperberg-McQueen, Extensible Markup Language(XML) 1.0, W3C recommendation, http://wwww.w3.org/TR/REC-xml, 1998.
6. 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.
7. 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.
8. James Clark, Stephen Deach, Extensible Stylesheet Language(XSL), W3C working draft, http://www/w3/org/TR/WD-xsl, 1998.
9. K. Decker, K. Sycara, M. Williamson. Matchmaking and Brokering. Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), Dec-96.
10. 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.
11. 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.
12. 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.

13. Bruce Krulwich, Automating the Internet Agents as User Surrogates, IEEE Internet computing, Vol. 1, No. 4, July/August 1997.
14. Nicholas Kushmerick, Daniel Weld, Robert Doorenbos, Wrapper induction for information extraction, IJCAI'97.
15. Andrew Layman, et al, XML Data, W3C note, http://www.w3.org/TR/1998/NOTE-XML-data-0105.
16. Alon Y. Levy, Anand Rajaraman and Joann J. Ordille, Querying Heterogeneous Information Sources Using Source Descriptions, Proceedings of the 22nd International Conference on Very Large Databases, VLDB-96, Bombay, India, September, 1996.
17. Hannes Marais and Tom Rodeheffer. Automating the Web with WebL. In Dr. Dobb's Journal, January 1999.
18. Mihaila, George and Raschid, Louiqa, Locating Data Repositories using XML, W3C Workshop on XML and Querying the Web, 1998.
19. Makoto Murata, Automatically Constructing the Intersection/Union/Difference of Two Schemas, XTech'99, march 7-11, 1999.
20. I. 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.
21. Y. Papakonstantinou, A. Gupta, L. Haas, Capabilities-Based Query Rewriting in Mediator Systems (Extended Version), in DAPD.
22. Raschid, Louiqa and Vidal, Maria Esther and Gruser, Jean-Robert. A Flexible Meta-Wrapper Interface for Autonomous Distributed Information Sources, Under Review. `http://www.umiacs.umd.edu/users/mvidal/`
23. Arnaud Sahuguet, Fabien Azavant, Wysiwyg Web Wrapper Factory (W4F), 1999.
24. 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.
25. Vasilis Vassalos, Y. Papakonstantinou, Expressive Capabilities Description Languages and Query Rewriting Algorithms, http://www-cse.ucsd.edu/ yannis/papers/vpcap2.ps
26. Maria Esther Vidal, Louiqa Raschid, Jean Robert Gruser, A Meta-Wrapper for Scaling up to Multiple Autonomous Distributed Information Sources, In Proceedings CoopIS'98.
27. 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.
28. `http://www.metacrawler.com`