An XML-based, 3-tier Scheme for Integrating Heterogeneous Information Sources to the WWW

C. Petrou, S. Hadjiiefthymiades, D. Martakos
Department of Informatics, University of Athens, Athens 15784, Greece
{cpetrou | shadj | martakos}@di.uoa.gr

Abstract
The phenomenal growth that the WWW currently experiences necessitates the integration of various types of information sources to its platform. We present an open, extensible multi-agent architecture, which is based on standards like XML, for the efficient retrieval of information from RDBMSs and HTML document collections. We also specify a series of protocols for achieving the synergy between a number of server- and client-side components.

1. Introduction

Nowadays, the WWW is believed to be the ideal software platform for the introduction of online telematic services as well as information systems in corporate intranets. In the majority of cases, legacy systems, such as databases, need to be adapted to this environment. In the past, we have witnessed several efforts, mostly 3-tier schemes, to port structured data to the Web [1]. Additionally, considerable work has been done towards the modeling of semi-structured Web documents, in order to achieve document integration, extract parts of documents and, finally, combine these parts into virtual documents. In the majority of these approaches, a hierarchical document model is proposed [2], [3], [4] along with a query language for retrieving and combining results even from multimedia sources [5], [6] (see also [7] for a review of WWW database techniques). It is worth noticing that even though the intrinsic hierarchical nature of XML specification could serve as the underlying model in such approaches, none of these approaches takes advantage of XML.

Web indexing, in the sense of spiders' construction and activation, is a well-known research issue in the field of keyword-based selection of WWW pages. Such spiders usually rely upon the indexing of WWW documents and subsequent searching of indices for user-provided keywords. For a given document, if keyword matching is succeeded then the document's URL is returned, possibly with some "weight" to denote the relevance with the given keyword set. Unfortunately, at least three drawbacks can be thought of:

- Intentional keyword repetition by document's author might bias indexing.
- A simple list of URLs does not provide enough information on which URL to select and on what is expected to be found when traversing the link.
- Many index bases or, in general, heterogeneous data sources might be available for posting a single request. Techniques like user feedback for a single index base [8] or multiple data sources [9], phrase indexing and documents clustering/abstraction have been proposed to overcome the first two issues. Although multi-agent approaches and 3-tier frameworks, like [10], have managed to alleviate the third drawback, the core of the problem is undervalued: how the user will perceive the structure and the type of the available data before posting some query, in a coherent and unified manner.

As mentioned, in the majority of Web-based information systems, based on HTML, the inner-structure of the retrieved information was undervalued while the scope of relevant semantics was restricted to the server-side. Such limitations can be surpassed with the XML standard, which caters for a more structured handling of the exchanged information. We present a relevant, 3-tier solution based on XML, Java, HTTP and Microsoft's OLE DB/ODBC. Specifically, on the server-side we have developed a hierarchical data retrieval mechanism, which comprises static Agents and coordinating entities (also termed Integrators). Static Agents handle individual information sources like RDBMS or Web indices. The schema/data model of each individual source is mapped to a generic representation known to both Agents and Integrators. Integrators operate behind classical Web servers using interfaces like CGI. A series of protocols were designed and implemented to handle information exchange between the various entities. Integrators map meta-information as well as the actual data to a proposed XML DTD. XML streams are transmitted to clients using regular HTTP dialogs, where they are parsed by a specially designed Java applet.

The rest of the paper is structured as follows. In Section 2, we discuss the overall architecture of the system. In Section 3, we elaborate on the various components located at the middle-tier of the architecture. Section 4 focuses on
the implementation details of the client module. We conclude this paper in Section 5.

Figure 1. Overall System Architecture

2. System Architecture

In Figure 1 we provide an illustration of the architecture of the proposed system. The main components are:

a) **Data Sources**: legacy information systems such as conventional RDBMSs (e.g., Oracle) or Digital Libraries (e.g., Dienst).

b) **Agents**: static (pre-spawned) processes that control the interaction with specific Data Sources. Access to Data Sources is accomplished through well-established specifications like Microsoft’s ODBC or OLE DB. Agents may also operate like typical Web spiders. As shown in Figure 1 the operation of each agent is supported by a number of meta-schemata, which will be discussed in subsequent paragraphs.

c) **Integrators**: information brokers which receive specific requests from Web clients, determine the proper Agent for their execution, relay requests, combine results coming from more than one Agents (much alike a horizontal fragmentation scheme) and, lastly, provide for session management.

d) **Gateway instances**: could be either light-weight CGI processes spawned by the Web server software or threads of the Web server enriched with code built upon the ISAPI, NSAPI or Servlet API specifications.

e) **XML-aware Java applets**.

3. Middle-tier components

In the following paragraphs we elaborate on the various components of the middle-tier (i.e., Agents, Integrators and Gateway Instances).

3.1. Agents

The role of the agent component is to establish a two-way connection with a specified Data Source, as well as to serve requests arriving from one or more Integrators. The connection with the Data Source is performed, mostly, via ODBC. In the prototype implementation, connections against various RDBMS (e.g., MS-SQL Server) have been successfully tested. Besides these types of connections, the Agent can perform searches in an index-base (formed after Agent’s operation as a Web spider) or forward search requests to an external application. In the latter case, a simple protocol has been defined, named OpenConnectivity (not discussed in this paper). Table 1 summarizes the possible types of data sources and the relevant protocols.

<table>
<thead>
<tr>
<th>Data Source type</th>
<th>Connection type</th>
<th>Protocol used</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>ODBC</td>
<td>ODBC calls</td>
</tr>
<tr>
<td>Local Index Base</td>
<td>ODBC</td>
<td>ODBC calls</td>
</tr>
<tr>
<td>Remote Application</td>
<td>TCP Sockets</td>
<td>OpenConnectivity</td>
</tr>
</tbody>
</table>

Table 1. Protocols between Data Sources and Agents

To ensure the generality of the Agent, the problem of different data types and different data type inner (binary) representations (usually encountered in commercial RDBMSs) had to be dealt with. The adopted solution involved the introduction of a global data type definition table, and the mapping of proprietary data types to the contents of this table. Agents’ communication with Integrators is compliant with this definition table thus, “hiding” data type diversity from the rest of the system (data type transparency). Agents are responsible for performing data type conversions in both directions (convert from proprietary data types to global and vice versa). A second issue was how to render the representation of schemata, exposed to the Integrator, independently of ad-hoc realizations in the various Data Sources. To tackle this, a Schema Description Language, known both to the Agent and the Integrator, was specified. The creation of a Schema for a given Data Source is semi-automatic. The Agent is capable of retrieving table definitions and processing them in order to create a preliminary Schema. Administrators may then alter such a Schema by:

- adding aliases for table and fields, to be shown to the client browser e.g., for multilingual interfaces,
- declaring that sample data should be provided to the client along with the Schema,
- removing table or field definitions to “hide” parts of the database that should not be exposed to the WWW,
- adding relationships between tables, as many DBMSs do not provide for their inner representation (e.g., DBASE files).

For every Data Source more than one Schemata can be defined and be locally stored and manipulated by some Agent. For each Agent-Integrator association, one of these Schemata is set as “active”. Different “views” of the same Data Source may, simultaneously, be shown to one Integrator by more than one Agents (Figure 2.a). Moreover, the same Agent may show different “views” of the same Data Source to different Integrators (Figure 2.b).
Agent-Integrator communication is based on BSD Sockets. The involved protocol adopts a simple request/reply scheme (the Agent dispatches queries in a stateless manner). Three types of requests are envisaged. The first request, GET_LIST_OF_SCHEMATA returns to the Integrator descriptive information for each “active” Schema. The second request type, GET_SCHEMA, returns a selected Schema (from the ones included in the GET_LIST_OF_SCHEMATA response). The administrator of the system drives the decision for Schema selection. Henceforth, all communication between a specific pair of Agents and Integrators refers to this chosen Schema. The last request, GET_DATA, followed by an ANSI SQL statement, invokes an ODBC request for data retrieval. Three scenarios may occur for GET_DATA: a set of records has been fetched, no records have been matched against the query, or the query was incorrect (in terms of syntax or references). In any case, a reply message is transmitted back to the Integrator.

In order to provide access to HTML pages, a Web spider mechanism has been incorporated into Agents. Such spider uses a local database for the persistent storage of the meta-information that has been produced during the indexing stage. The Agent handles this local database, in a way similar to regular Data Sources. Hence, no special manipulation is required by the other entities of the architecture (e.g., Integrators) to access the indexing information. The Schema associated with an HTML Data Source, practically is a relational representation of HTML documents. Such a Schema doesn’t reflect the actual database structure in support of the HTML Data Source, as this would add extra complexity in query formulation and impose a certain dependency of the architecture to specific indexing schemes. Instead, some additional processing is undertaken by the Agent to ensure this type of abstraction.

Given a set of starting URLs, the spider fetches the relevant pages and performs selective indexing for parts of each page (e.g., Title, Headings). The procedure of indexing has a pre-indexing phase where all stop-words (taken from a predefined list) are excluded from further processing. The remaining words are then parsed and indexed both as single terms and as phrases. Queries for HTML Data Sources are formulated using an SQL-like syntax. Indexed parts of the HTML documents form the results of query execution. To initiate indexing, the administrator sets the values of 5 parameters and, possibly, defines a location filter. These parameters are:

- Depth: the number of subsequent link traversals that will be performed before indexing concludes.
- Width: the number or the percentage of embedded links that will be traversed by the spider.
- Minimum/Maximum size of the document. If a document has lesser/larger size, it is excluded from further processing.
- Connection time-out: a waiting time-out for response from a certain host.
- Location filter: a list of terms that will be traversed by the spider.

The purpose of the location filter is to exclude/include an HTML document from indexing if its URL matches a filter's predicate; wildcards are also allowed. In order to expedite lengthy indexing, we enhanced our spider with the option for incremental processing (i.e., given the list of URLs to be indexed, those that have not yet been processed are marked for future processing).

In conclusion, the Agent component ensures access to RDBMSs supporting ODBC. Moreover it can serve as a WWW spider and search-engine. It handles many simultaneous connections to Integrators, providing different “views” of the available Data Sources, in terms of what information will be exported (tables and fields) and how such information will be described on the client side (aliases). It accomplishes both data-type and schema abstraction thus, enabling a unified and coherent representation of various Data Sources.

3.2. Integrator

The Integrator component is responsible for relaying requests from Gateway Instances (CGI, Servlets, etc.) to Agents and controlling the flow of results in the reverse direction. Integrators have a multi-threaded architecture, which allows simultaneous communication with multiple Agents. Specifically, Integrator is responsible for:

- receiving requests from Gateway Instances,
- authenticating users and performing session management,
- detecting the appropriate Agent(s) that must be invoked to supply the requested data,
- translating incoming requests to local schemata and forwarding them to Agent(s),
- collecting results from the Agent(s) and, possibly, merging such results, and
- converting the received results in XML format and returning them to the Gateway Instance.
The Integrator performs all these steps in the context of individual user sessions. State information is generated and kept within the Integrator to ensure the logical correlation of distinct requests coming from the same client (WWW session management has been addressed in [11]). In the discussed architecture, state is preserved by means of unique connection identifiers that the Integrator generates.

Clients submit their authentication information (username, password) upon connection request. The Integrator tries to match such information within internal lookup tables, updated upon Agents’ start-up and connection to the specific Integrator. Following successful authentication, all the relevant Schemata are transmitted back to the user. Subsequent requests by the interacting user may refer to one or more of the returned Schemata. Therefore, certain scenarios can be identified:

1. The incoming request designates one target Agent along with a SQL Select statement. The request is translated to the local Schema and then forwarded to the appropriate Agent. Results are converted to XML and forwarded to the appropriate Gateway Instance.

2. More than one Agents are identified in the request (along with a SQL Select statement). The Integrator detects those Agents and forwards the request to each one individually. Similarly to Case (1), results received by the Integrator are converted to XML and relayed to a Gateway Instance with a header indicating which part has been provided by which Agent.

3. An incoming request has no Agent binding information. A “best-match” approach is used to handle the request. Given the set of “valid” Agents, the Integrator attempts to match the query against the relevant Schemata. A “best-match” occurs when all parts of the query (i.e., SELECT and WHERE clauses) can be mapped to the Schema elements. A “partial-match” occurs when some parts of the query can not be mapped to a Schema (e.g., a field in the “SELECT” clause does not exist in the considered Schema). A transformation of the query is then performed, leading to a “reduced” query which is valid for the considered Schema. Such “reduction” is attempted for all valid agents, excluding those that “achieved” a best-match. Queries’ reduction may be successful or not (more detailed discussion will follow). The relevant Agents then execute successfully reduced queries. All the result sets returned to the Integrator are combined as in Case (2).

As mentioned, query reduction may either be successful or not. The decision is based on the next algorithm.

Upon request for a Schema, the Integrator locates and parses the relevant Schema, adding record data according to Option values. For an Option value equal to 1, no extra data are added. If the Option has value 2, then a set of sample records, equal to the next parameter found, are added (e.g., “2:5” causes the addition of 5 sample records). Finally, a value equal to 3 denotes that the whole set of table records should be added to the meta-schema. The sample data mainly serve two purposes on the client-side:

- To help the user reach a conclusion about how to set values for fields in terms of semantics (data type checking is performed by the client applet).
- To enable the creation of lookup tables. For an Option value equal to 3, a lookup table is displayed on the client interface instead for an input box. For an Option value equal to 3 and a mandatory, 1-to-many relationship, the client interface displays a lookup combo-box.

### 3.3. Gateway instances

Gateway instances are invoked upon receipt of an HTTP request posted by the client-side. They decode query information, distinguish information elements and pass those to the Integrator. On the reverse direction, they receive XML code and pass it to the client-side. Integrator-Gateway communication is Socket-based. Our architecture is open to all known gateway specifications.

### 4. XML-aware client-side components

The suggested architecture demonstrates the suitability of the eXtensible Markup Language (XML) [12] in the area of Web-powered database applications. XML is used as a low-level protocol for passing information to the Web client. Such information may encompass the database schema, the actual data retrieved by a DBMS as well as information pertaining to the particular session the user has established (subsequent interactions between the client and the server are assumed to follow the session model instead of the stateless scenario). Below, we provide the XML DTD used for the communication between the client applet and Integrator (note that the gateway instance simply relays data to both directions). Firstly, we show the response to a request for a Schema:

```xml
<?xml version="1.0"?>
<!DOCTYPE schema_response [
<!ELEMENT schema_response [transaction, database]>
<!ELEMENT transaction (#PCDATA)>
<!ELEMENT database (table+, relationship*)>
```
Secondly, we present the response to a request for data. The use of XML for publishing database information on the Web is considered extremely advantageous [13]. One of the most important benefits stemming from the adoption of database-specific XML code is the redistribution of processing load from servers to Web clients. Additionally, due to its intelligence and processing capability, the Web client (a Java applet in our system) may present different views of the same data to different users (in contrast to typical HTML applications where the same view is always accessible). The combination of XML and Java for the manipulation of database information at the client-side allows the realization of complicated processing. Indicative examples are different kinds of sorts, the synthesis of queries by graphical means or conventional QBE, the validation of user input, etc. XML manages to overcome basic deficiencies of HTML like:

a) the inability to represent different structures and entity attributes within the delivered content, and,
b) the absence of structural validity checking mechanisms.

Due to point (a), XML has been considered the more suitable technology for bringing together the world of database systems and the Internet world [14].

5. Conclusions

We have presented a distributed architecture for the efficient retrieval of information residing in various types of databases (e.g., RDBMSs or HTML document collections). Such architecture is based on the emerging XML W3C standard as well as ODBC, Java and HTTP. In the future, we plan to extend the proposed architecture and associated protocols in directions such as the handling of all kinds of database operations (e.g., updates).

6. References