Direct Access to Databases through WWW Browsers

Yannis Stavrakas
ys@dblab.ntua.gr

Vagelis Zorbas
vzorbas@dblab.ntua.gr

Kostas Chatzinas
kchatzin@dblab.ntua.gr

Knowledge and Database Systems Laboratory,
National Technical University of Athens, Greece,
http://www.dblab.ntua.gr/

Abstract: We present a model that allows Web users to directly access data at remote heterogeneous databases, in a way similar to accessing hyper-documents. The model deals with the expression of requests from web browsers to databases in order to retrieve stored information, and the transfer of results to web clients. In addition, we examine the possibility of presenting results from more than one databases in the same web page, and of selective incorporation of results into HTML documents. In order to support the desired functionality, the model introduces a new Web protocol, a new URL scheme, and a couple of HTML tags. We developed a prototype system that fully implements the features we describe in this paper.

1. Introduction

1.1. Databases in the Web

The Web is the main reason behind the wide adoption of Internet for communication, information publishing, and commercial purposes. The initial design specifications of the web became soon insufficient for the requirements of emerging web applications. The conventional usage of the web for publishing hyper-documents and navigating through the virtual network they create, has been complemented with many kinds of highly interactive applications where data processing plays an important role. In the frame of those applications, the utilization of databases has become essential. Let us quickly review some of the ways databases are used in the WWW today:

- Some times the term "web database" is used to refer to collections of encoded meta-information on web pages. This meta-information is used for answering queries about web pages with increased accuracy and efficiency. Search engines, using indexing mechanisms for retrieving pages according to their content, fall into this category.
- As web applications become more complex and handle large quantities of data, they often need the support of a database, especially designed for the application. In an virtual bookstore site, for example, the database may keep records about books and client orders.
- In some cases, there is need to publish information that is already stored in pre-existing databases. In contrast to the previous situation, those databases are not designed for a web application, but support organizational functions, and are probably used by legacy applications. The publication of this data through the Web comes as an additional requirement that emerges due to the advantages of the WWW technology.

Therefore, in the first case users access the Web through databases, while in the last two cases they access databases through the Web. The last case - accessing legacy databases through the Web - has the focus of the proposed model.

1.2. Motivation for the Proposed Model

There are various ways for web pages to communicate with databases [REN97]. Most of them extend the web pages by merging them with modules that handle the database connection. Those *extensions* could be thought as a series of executable files, as it is the case of CGI, or chunks of code embedded in HTML pages:

- Web server's CGI capability enables the web server to initialize, execute and get results from scripts, written in any language. The scripts are responsible for interacting with the database and generating or consuming web pages.
- Alternatively, HTML is extended with proprietary languages which are interpreted from a specific web server. Such languages are capable of communicating with databases.

Figure 1 illustrates the current architecture for accessing databases.

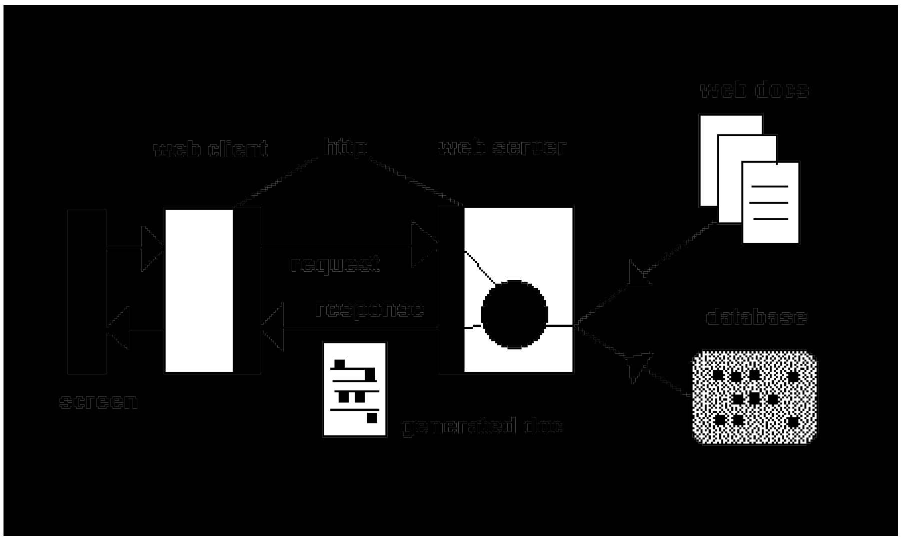

***Figure 1**. Current architecture for accessing databases through WWW*

For the co-operation between web pages and databases, the following points hold:

- It is the extension that has the knowledge that a database is being used. The web pages, the web server, and the web client do not "see" the database. Thus, databases in the Web are hidden in the backstage.
- The extension is an application written especially for the schema of a given database. For publishing information from a different database we need to develop another application.
- Limited interaction is allowed between users and databases, because all the possible actions are pre-defined and embedded in the application.
- Extensions are very probably dependent on the web server used, which results to low portability.

Evidently, the Web has been designed to work well with information in flat files. The way the Web merges with structured data, and more specifically with databases, seems to be the result of covering needs as they appear rather than that of a careful and comprehensive study.

Our proposal is motivated by the conviction that databases can play a more active role in the Web, increasing the availability of information, and taking advantage of the existence of the Internet. It seems to us that a prerequisite for such a development is that databases become first-class citizens and participate in the Web from a position similar to that of the hyper-documents. Instead of being hidden behind applications, they must be directly visible to Web users, in such a way that:

- Users should be able to specify through the web browser the databases they want to connect to, just like they can specify the web server hosting the required web pages.
- In addition, users should be able to express requests to the databases for performing operations, just like they demand to web servers that they provide specific web pages.
- Communication between databases and web clients should not require the development of especially-made applications. Every database, irrespectively of type and schema, should be supported.
- All of the above should be supported in a way that takes under consideration the web standards and that allows for maximum flexibility and control in merging structured information with hyper-documents.

1.3. Other Approaches

As we have already mentioned, connection between web pages and databases takes place at the server side through CGI scripts, or through proprietary languages of web servers. To the best of our knowledge there are only two approaches that deviate and follow to some extend the principles presented in the previous section:

- The Java [CHL96] programming language allows the development of applications that run on web clients and communicate directly with database servers through Internet. In this case, the client "sees" the database, however the conversation between them requires an especially built Java application, tailored to the occasional database schema.
- Microsoft has proposed Dynamic HTML (D-HTML) [DOB98]. D-HTML is based on Document Object Model (DOM) specifications [DOM97], published by the W3C, which defines a mechanism for attaching objects to web pages. Those objects could be executable programs with access to the structural elements of the web page. Thus, pages could change dynamically in response to events as DOM objects are being executed in the web client. In essence, D-HTML is a DOM-based implementation, which, among others, allows a web client to incorporate logic for communicating with databases.

In section 2 we continue with a detailed description of the proposed model. Section 3 gives some examples of the functionality. In section 4 we discuss briefly the implementation of the model. Section 5 closes with some conclusions and future work.

2. The Model

2.1. Architecture

The objective is to promote databases to first-class citizens in the Web. A basic concept seems to be that a web client must be aware of the existence of databases, and be able to express its demands directly to them. To make that possible, we must examine from this perspective the triplet HTML-URL-HTTP and come up with adjustments that will enable such operations.

- Extensions to HTTP led to defining *DBTP (Data Base Transfer Protocol)*, a new communication protocol that supports the requirements and functionality of the proposed mechanism.
- Extending URL semantics led us to the definition of *dbtpurl*, a new URL scheme, whose links define operations on databases.

- Finally, we augmented HTML with two new tags, in order to allow for flexible and controlled manipulation and presentation of results.

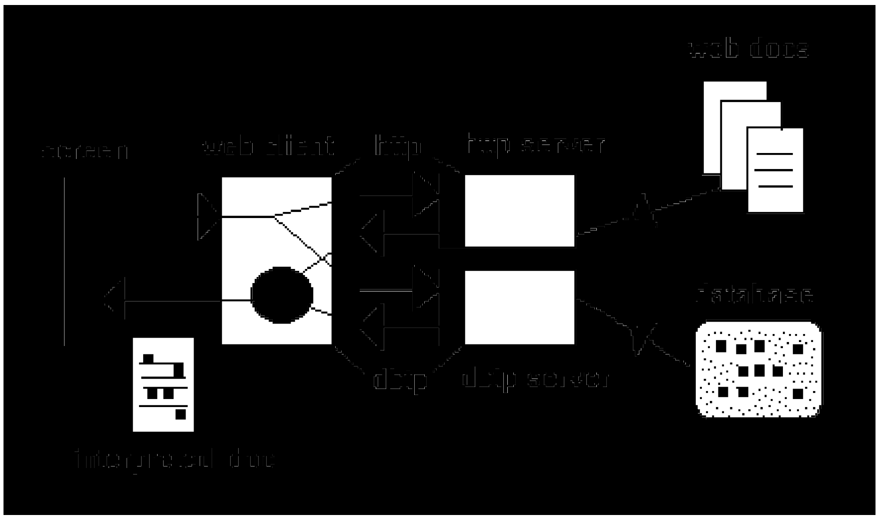

Figure 2. *Proposed architecture for accessing databases through WWW*

The architecture for the proposed system is illustrated in *Figure 2*. The components in *Figure 2* are the web client, which is an extended browser also called *dbtp client*, and an especially built server named *dbtp server*. Those components implement the features presented in the following paragraphs.

2.2. URL Scheme

URL (Uniform Resource Locator) [RFC1738] is the standard way for specifying a resource available on the Internet. Following the general URL syntax rules, we defined a new URL scheme, tailored to database resources. The *dbtpurl* allows web users to specify in a generic way the database to connect to, and the operations to perform.

```
dbtpurl = "dbtp://" login ["/"[ ["metadata" | Range] "/"] [ dbName [":" query ]]]
            login = [user[":"password] "@"] hostport
            hostport = host [":"port]
            Range = "Range:" *DIGIT "-" *DIGIT
            DbName = *unreserved
            Query = *xchar
```

Table 1. *Formal definition of the dbtpurl*

The formal definition of the new URL scheme is presented in Table 1 in augmented BNF syntax [RFC822]. Parts of this definition are the server IP address, the database name, the database query, and additional information regarding the request. The tokens "*user*", "*password*", "*host*", "*port*", "*DIGIT*", "*unreserved*" and "*xchar*" are further defined in [RFC 1738].

Additional request information supported by *dbtpurl* include:

- Username and password for user authorization against the database.
- Indication that the user would like to receive information about which databases are available on a specific dtpt server.

- Indication that the user would like to receive meta-information on the schema of a particular database.
- Indication that the user can receive only a maximum number of results.
- Indication that only a specified part of the database results should be returned to the user.

The validity of the *query* part of the URL is ultimately determined by the database which handles it. It can be any valid expression of the database own query language. For SQL queries we provide the alternative to apply a simple encoding, mainly for the sake of brevity, where SQL keywords are replaced by special characters.

Examples of *dbtp* URLs, accompanied with a short explanation, follow (for readability, the spaces have not been replaced with "%20" as they normally are).

1. *dbtp://www.dbtpServer.com:4444/BooksDB:# * ? Book*
   Apart from the protocol to be used (dbtp), the host, and the port the dbtp server is listening to, this URL indicates:
   - The database to which the query must be sent: "BooksDB"
   - The encoded query to the database: "# * ? Book", which corresponds to the SQL query: "SELECT * FROM Book".

2. *dbtp://name:pass@www.dbtpServer.com/range:0-20/BooksDB:# * ? Book*
   Further indicates:
   - The username and password for authenticating the connection to the database.
   - That the user wishes to receive only the first 20 lines of the results.

3. *dbtp://www.dbtpServer.com/*
Alternatively: *dbtp://www.dbtpServer.com/metadata/*
   As a result to the above requests, metadata on the services offered by the server are formulated by the dbtp server and sent back to the dbtp client. Metadata may include the names and schemas of the databases exposed by the dbtp server, and information on the server capabilities.

2.3. Communication Protocol

The principal question is whether HTTP can accommodate the *dbtpurl* features. The answer is an initial *yes*; HTTP can be used for serving database communication needs, either by extending or not:

- Leaving HTTP unmodified means that *dbtpurl*-related information must be added at low level inside the body of the existing HTTP messages, since the higher level HTTP methods refer to documents and are not suitable for databases. This means that if HTTP is used as is, the semantics of a new protocol are "burried" inside the HTTP infrastructure. This solution is not satisfactory, as the HTTP methods will be overridden by semantically irrelevant information hidden inside the HTTP messages body.
- Extending HTTP to incorporate new functionality for serving *dbtpurl* requests, is the second solution we considered. It seems that an extended HTTP protocol could support *dbtpurl* needs. However, this approach may present a few drawbacks: (1) HTTP operations could prove inadequate for the required functionality. HTTP security and caching would probably need modifications to stand for databases as well. (2) The new protocol would become heavy and complicated, trying to serve both hyper-documents and databases.

Our approach was to build a new protocol from scratch, because of the following advantages this solution has.

```
DBTP-message = DBTP-Request | DBTP-Response

DBTP-request = Request-Line          DBTP-response = Status-Line
               |*(general-header                     |*(general-header
               | request-header                      | response-header
               | entity-header )                 | entity-header )
               CRLF                                  CRLF
               [ message-body ]                      [ message-body ]

Request-Line = Method SP dbName SP DBTP-Version CRLF
Status-Line = DBTP-Version SP Status-Code SP Reason-Phrase CRLF
DBTP-Version = "DBTP" "/" 1 *DIGIT "." 1 *DIGIT
Request-header = Authorization          Entity-header = Content-Length
                 | Maxlines                             | Content-Type
                 | Range
                 | Query-Encoding
Method = "query"                          Status Code = "200" ; OK
         | "metadata"                                   "206" ; Partial Content
dbName = "*"                                            "400" ; Bad Request
         | token                                        "401" ; Unauthorized
                                                        "420" ; Bad Query
                                                        "500" ; Server Error
                                                       "505" ; Version not supported
Authorization = "Authorization" ":" userid ":" password
Maxlines = "Maxlines" ":" *DIGIT
Range = "Range" ":" *DIGIT "-" *DIGIT
Query-Encoding = token
Content-Length = "Content-Length" ":" 1*DIGIT
Content-Type = "Content-Type" ":" type "/" subtype

(SP = Space, CRLF = Carriage Return, Line Feed)
```

Table 2. *Formal definition of the DBTP message*

- A new protocol can be fully adapted to the *dbtpurl* needs. Operations like connection duration, message encoding, caching and security, can be designed with database communication in mind.
- The new protocol, being independent from HTTP, could be selectively deployed from web browsers and specialized dbtp servers.

DBTP, a new Internet protocol, is formally defined in terms of BNF (*Table 2*). It is an application level protocol, designed to work over a reliable transport protocol (like TCP), with the following features:

- Its operation is based on the request/response paradigm. A dbtp client establishes a connection with a dbtp server, sends a Request message to the server, receives a Response message from the server and finally the server closes the connection.
- It is an object oriented, human readable and stateless protocol.
- The communication unit of the protocol is the message. Request and Response messages carry the client's database request to the server, and the database response back to the client.
- Its operation and the form of messages are similar to those used in HTTP.

The formal definition in augmented BNF of the dbtp messages is given in Table 2. The tokens "*token*", "*type*", "*subtype*", "*DIGIT*", "*userid*", "*password*", "*SP*", "*CRLF*" are further defined in [RFC2068].

A mapping of dbtpurl to DBTP Request messages has been defined in detail. The information is carried into headers alike to HTTP messages.

A sample of dbtp messages follows, based on the 2nd example of the dbtp URLs given in a previous section:

1. The DBTP Request message issued by the dbtp client:

" *query BooksDB DBTP/1.0 \r\n*
Authorization: name : pass\r\n
Range: 0-20\r\n
\r\n
*# * ? Book* "

2. A possible answer from the Server could be the following dbtp Response message:

" *DBTP/1.0 200 Query Executed\r\n*
\r\n
<…Results From Database…> "

2.4. HTML Extensions

The dbtp client can display the database results directly, formatted as HTML table. However, it should be possible for the dbtp client to select data from the results, and insert them at specific positions, in a web page that will be the actual response. In order to enable this selective merging of data and hypertext at the client side, we extended HTML with the following tags:

1. The <fetch> tag specifies a URL, dbtpurl or conventional, whose contents are accessed and cached locally at the client, but not immediately displayed.
2. The <insert> tag refers by name to a <fetch> tag, and specifies a subset or all of the data that correspond to the <fetch> URL. Those data substitute <insert> at its location, and are subsequently displayed as part of the web page.

```
<!ELEMENT FETCH - O EMPTY>
<!ATTLIST   FETCH
        name   NAME    #REQUIRED
        href    %URL     #REQUIRED
        >

<!ELEMENT INSERT - O EMPTY>
<!ATTLIST   FETCH
        src      NAME       #REQUIRED
        row     NUMBER   #IMPLIED
        col      NUMBER   #IMPLIED
        >
```

Table 3. *Formal definition of the FETCH and INSERT tags*

Multiple <fetch> and <insert> tags can be used in the same HTML page. In case the <fetch> URLs are conventional URLs referring to HTML documents, <insert> tags can be used to dynamically interpose HTML documents in the current page. This is advantageous because documents will automatically reflect changes made at their site of origin. In case the <fetch>

URLs are dbtpurl however, those tags become even more useful as they offer a way to selectively incorporate in a web page data returned from databases. This mechanism can be thought of as a way to *define views* over hyperdocuments and structured data.

Formally, in terms of SGML [SGML], the two tags are defined in Table 3. The tag FETCH contains a URL whose contents are downloaded by the dbtp client. The URL can be a dbtpurl, or other URL type. In any case, the resource is retrieved and embedded in the HTML document, substituting the INSERT tag whose *SRC* name is the same as that of the FETCH tag. The optional attributes ROW and COL can be used in the case of referring to database results, to specify what data to insert at given locations of the web page.

3. Operation

3.1. System Design

Using the appropriate protocols, the dbtp client can receive information both from a Web server and a dbtp server. The data is combined, using the HTML extensions, to formulate an HTML page which is subsequently displayed. *Figure 3* illustrates the system.

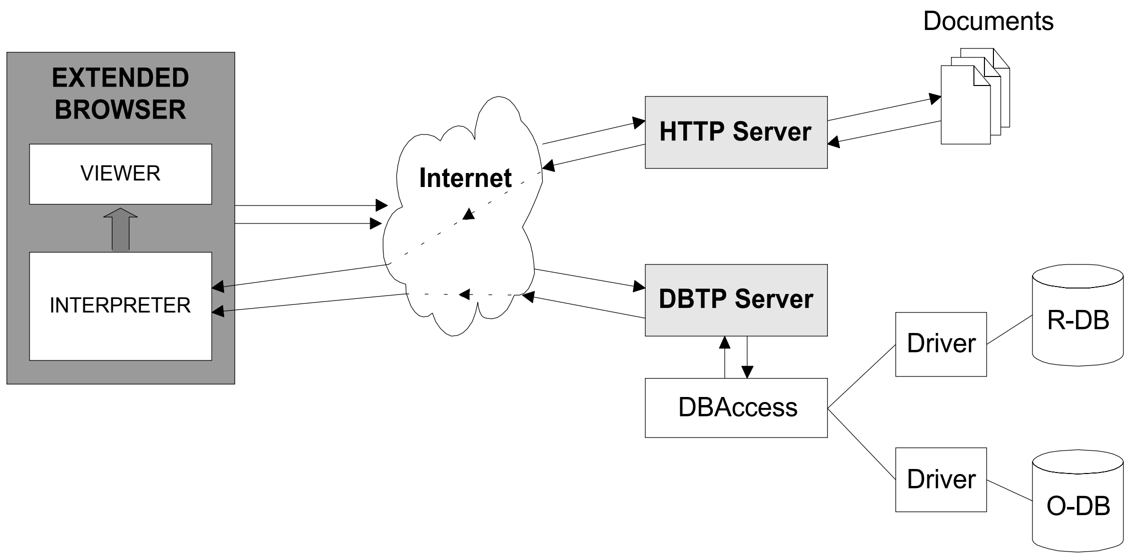

Figure 3. *HTTP and DBTP operation*

In the case where the results of a database request are immediately presented without further processing, the following steps describe the transaction.

1. The user inputs a dbtpurl, or clicks on a dbtpurl link.
2. The dbtp client forms a DBTP Request message and sends it to the dbtp server.
3. The dbtp server receives the DBTP Request message, extracts the query and the target database, and passes the information to the DBAccess module, which is responsible for communicating with the database.
4. The DBAccess translates the query if necessary and submits it to the database.
5. The query response is sent back to the DBAccess module.
6. DBAccess translates the results to a suitable format and passes them to the dbtp server.
7. The dbtp server sends a DBTP Response message to the dbtp client.
8. The dbtp client displays the results, formatted as HTML table.

In the case where data is merged with hypertext, the process is as follows:

1. The user inputs a conventional URL that refers to a page containing HTML extensions.
2. The page is downloaded and parsed to identify HTML extensions. Each time a <fetch> tag is encountered, the dbtp client downloads and stores temporarily the resource *href* points to. The resource can be database results from a dbtp server or documents from a http or ftp server (e.g. http, ftp server).
3. The interpreter substitutes <insert> tags with the data they refer to. Results from databases and html documents can be inserted in the place of <insert> tags.
4. The viewer displays the outcome, a standard HTML document.

3.2. Example

For our example we will consider the following hypothetical servers: (a) the dbtp server "gorgon.com", that publishes a database with information about books, (b) the dbtp server "latest.resources.com", that holds a database with product sales information, and (c) the web server "latest.resources.com", where a test page (/XML/main.html) is residing.

The test page is an extended HTML page, which dynamically provides the user with updated information on XML resources, gathered from dbtp and web servers. We will use the W3C site as a web server data source, to incorporate XML specification information in the final result.

The user requests the "http://latest.resources.com/XML/main.html" and the test page is downloaded from the web server. The contents of the page appear in Table 4.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
        <title>Latest XML Resources</title>
</head>

<body>
<FETCH NAME="W3C" HREF="http://www.w3c.org/xml/">
<FETCH NAME="gorgon" HREF="dbtp://www.gorgon.com/BooksDB:
                                    ? * Books % Category='XML'">
<FETCH SRC="best"
  HREF="dbtp://products.eval.com/Products:# Name,URL ? Sales % Category='XMLTools'
                      AND SalesNo in (# max(SalesNo) ? Sales % Category='XMLTools')">

<h3>The latest XML specification from W3C</h3>
  <p>
<INSERT SRC=W3C>
  </p>
<hr>

<h3>Books on XML available from the bookstore gorgon.com</h3>
  <p>
<INSERT SRC="gorgon">
  </p>
<hr>
```

```
<h3>The best selling XML tool from products.eval.com</h3>
  <p>
Until today the best selling XML product is
<FONT color="Blue">
  <INSERT SRC="best" ROW="2" COL="1">. <br>
</FONT>
More information is available at
<FONT color="Blue">
  <INSERT SRC="best" ROW="2" COL="2">
</FONT>
  </p>
<hr>

</body>
</html>
```

***Table 4.** An example of Extended HTML page*

The extended browser will parse the document and locate the three <fetch> tags.

- For the first FETCH tag “W3C”, the html page corresponding to the URL “http://www.w3c.org/xml/” will be downloaded and inserted into the document, replacing the INSERT tag having SRC attribute “W3C”.
- For the second FETCH tag, named “gorgon”, the extended browser will open a dbtp connection with the dbtp server *www.gorgon.com*, and send a query with the SQL equivalent:
```
SELECT *
FROM Books
WHERE Category='XML'
```
- The server will return the results from the database to the browser. The results will be inserted in the document as an HTML table, replacing the INSERT tag having SRC attribute ”gorgon”.
- For the third FETCH tag, named “best”, the extended browser will send to the dbtp server a query with the SQL equivalent:
```
SELECT Name,URL
FROM Sales
WHERE Category='XMLTools' AND SalesNo   in
        (SELECT max(SalesNo)
        FROM Sales
        WHERE Category='XMLTools')
```
- This query will retrieve from the Products database the best selling XMLTool (the tool with the maximum number of SalesNo). The result, formatted as HTML table is shown in Table 5. The two INSERT tags with SRC attributes “best” are replaced in the final document with “Xmagic” and “http://xmagic.tool/main.html” respectively.

Name	URL
Xmagic	http://xmagic.tool/main.html

***Table 5.** Results of <FETCH SRC="best" ...>*

The result is a conventional HTML page, without <fetch> and <insert> tags. The page that will be finally displayed will look like the one illustrated in Appendix I (the W3C part is cropped and only a couple imaginary XML book titles are included).

4. Prototype Implementation

A prototype system was designed and implemented at the Knowledge and Database Systems Laboratory of the National Technical University of Athens. The system consists of a web client and a dbtp server; they were both developed in Java (JDK 1.1).

The web client offers basic HTTP and FTP functionality, and implements the new DBTP protocol over TCP/IP connections. The main functionality and features of the web client are given below:

- It parses the dbtp URL and forms the corresponding DBTP Request message.
- It communicates with a dbtp server, sends the Request and receives the Response.
- It decodes the Response message and it either presents the results or a message indicating error.
- It identifies the HTML extensions (FETCH and INSERT) in downloaded documents and performs a merging of downloaded resources to formulate the final HTML page. The document is then temporarily stored on disk, and a standard web browser is called to display it.
- It can open many connections to http or dbtp servers simultaneously on separate threads.

The dbtp server is mainly responsible for the communication with databases through the JDBC API, and the formatting of the response. The main functionality and features of the dbtp server are the following:

- It receives a DBTP request message and decodes it, extracting the user's query.
- It sends the query to the database and gets the results.
- It formats the results as HTML table.
- It forms the appropriate DBTP response message containing the database answer.
- It can serve many databases and many clients simultaneously in separate threads.
- It is able of sending to the dbtp client database schemas and other metadata.
- It can listen to user-defined ports for incoming requests.

Both the dbtp client and server are portable (the system has been tested on Windows and UNIX platforms) and have graphical interfaces. Informix and MS Access were used in testing as database systems. The prototype system is available for downloading at http://www.dbnet.ece.ntua.gr/~ys/dbtp/bin/. Installation instructions and a brief tutorial are available at the same address in the file "manuals.doc".

5. Conclusions & Future Work

The purpose of this work is to show that structured data can become first-class citizens in the web, a status that flat files already enjoy. To achieve that, more complex mechanisms are needed for manipulating, transferring, and incorporating structured data in web documents. We have proposed adjustments to the current URL, HTML, and HTTP standards, in order to promote the WWW connection to databases to the same level as that of flat files.

The main advantages of our approach are that client access to structured data does not depend on server-resident custom applications, and that it becomes possible to incorporate in the same web page data coming from more than one web servers.

As possible directions for future work we consider:

- *XML use:*

Using XML for formatting database results would have significant advantages over HTML. XML is rich enough to represent complex data structures. This means that XML documents can represent data results from all kinds of databases, Relational or Object. In this case the need to define extensions to HTML would be eliminated. The merging of hypertext and database results could be designed in a more efficient way, taking advantage of the structured, self-describing XML documents. In this context, joining database results from many <fetch> tags at the client side could also be considered.

- *Security:*

One of the most important features that have to be added to the system is security. Database information can be very critical. Authentication, protected access level and encryption methods and protocols must be enforced to ensure confidentiality of information.

- *Applet Version:*

The dbtp client is developed as a Java program. This means that a user must first install the program in order to enable DBTP access. An applet version of the program would give the advantage of publishing the program to every web browser supporting Java.

- *Dynamic Insertion:*

It would be desirable to add dynamic features to web pages, by enabling a co-operation between DBTP features and JavaScript. Such a co-operation would allow to create web pages acting as database front-ends at the client side.

- *Statefull DBTP:*

Database transactions involve consequent queries posed to a database. Following the HTTP/1.1 standard, a persistent connection between the dbtp client and the dbtp server would have many advantages in that case.

- *Caching*

Caching mechanisms similar to those used in HTTP, could be designed and employed to enhance database access efficiency.

References

[CHL96] "The Java Class Libraries", Patrick Chan, Rosanna Lee, Addison-Wesley, 1996.

[CW] "The Java Tutorial", M. Campione, K. Walrath.

[DOB98] "Data Binding in Dynamic HTML", Rick Dobson, DBMS, March 1998.

[DOM97] "Document Object Model", http://www.w3.org/DOM/, September 1997.

[JLW] "Web Programming", Kris Jamsa, Suleiman Lalani, Steve Weakly, Jamsa Press.

[REN97] "Basic Web Architectures", Martin Rennhackkamp, DBMS, May 1997.

[RFC1738] "Uniform Resource Locators", T. Berners-Lee, L. Masinter, and M. McCahill, December 1994. Available at http://ds.internic.net/rfc/rfc1738.txt.

[RFC1808] "Relative Uniform Resource Locators", R. Fielding. June 1995. Available at http://ds.internic.net/rfc/rfc1808.txt.

[RFC1945] “HTTP Version 1.0", R. Fielding, H. Frystyk, and T. Berners-Lee, May 1996. Available at http://ds.internic.net/rfc/rfc1945.txt.

[RFC2068] "HTTP Version 1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, and T. Berners-Lee, January 1997. Available at http://ds.internic.net/rfc/rfc2068.txt.

[RFC822] In chapter 2 of RFC 822: “NOTATIONAL CONVENTIONS”, the principles of the augmented BNF (Backus-Naur Form) grammar are described.

[RHJ97] Dave Raggett, Arnaud Le Hors, Ian Jacobs, "*HTML 4.0 Specification*", W3C Proposed Recommendation 7-Nov-1997, PR-HTML40-971107, available at http://www.w3.org/TR/PR-html40/cover.html

[SGML] “Standard Generalized Markup Language”: An official ISO standard (ISO 8879)

[SSJS] “Writting Server-side JavaScript Applications”, http://developer.netscape.com/library/documentation/enterprise/wrijsap/index.htm

[XML] “Extended Markup Language”. http://w3c.org/xml/

Appendix I: Example page after HTML extensions have been replaced

Note that the W3C page is cropped.

The latest XML resources from W3C

Books on XML available from the bookstore gorgon.com

Title	**Author**	**Publisher**	**Price**
XML developers Library	Megane	Book Hall	20$
XML and Internet	Laverel	Kaktos	25$

The best selling XML tool from products.eval.com

Until today the best selling XML product is Xmagic.
More information is available at http://xmagic.tool/main.html

---