Browsing Object Databases Through the Web[§]

Joachim Hammer, Rohan Aranha, and Kelly Ireland

Computer Science Department
Stanford University
Stanford, CA 94305
{joachim, aranha, kelly}@cs.stanford.edu

As a result of the explosive growth of the Internet, more and more sites have connected their databases to the WWW, allowing users to retrieve information dynamically and on-demand. However, most dynamically created HTML pages currently available on the WWW, provide the user with only a flat view of the contents giving little visual help as to how the information is organized. As part of the TSIMMIS project at Stanford, we have built a system for formatting and displaying the information that is retrieved from an object-based database as a "web" of hypertext documents using HTML and HTTP. The contents of these "documents" can then be explored using the built-in functionality of a WWW browser much in the same way one would explore the contents of a book. As part of our system, users can customize the formatting of their results by choosing different layouts, or controlling how much of the object structure is visible in the browser, for example. In this paper, we illustrate the details of the approach and describe our prototype implementation called MOBIE that is installed and operational in the TSIMMIS testbed.

1. Introduction

Due to the explosive growth of the Internet, the WWW has become an open, distributed forum in which people can publish and exchange information freely and without any restrictions in terms of what hardware or software they use. In order to organize the ever increasing amounts of information, more and more data is gathered and served dynamically from databases rather than from static HTML files. This approach combines the advantages of a database management system (e.g., efficient search and retrieval capabilities, concurrency, recovery, etc.) with the advantages of the WWW (e.g., ubiquitous, standard interface, etc.). However, most current approaches to dynamically "publishing" the contents of databases are not fully exploring the fundamental

§ Research sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Material Command, USAF, under Grant Number F33615-93-1-1339.The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of Wright Laboratory or the US Government. This manuscript is submitted for publication with the understanding that the US Government is authorized to reproduce and distribute reprints for Government purposes.

paradigm of the WWW, namely being able to link together information units that are related but exist in separate documents. Rather, most WWW-based databases provide the user with just a "flat view" of the entire result, giving little visual help as to how the information is organized. This is acceptable as long as the information to be displayed is short or has little structure that requires special formatting. However, information sets that are large or very structured (with many levels of nesting) cannot efficiently be displayed in a flat structure and will result in a lot of scrolling when viewed on the screen. For this reason, we have built a system for dynamically formatting and browsing structured information as a "web" of hyperlinked HTML [Berners-Lee and Connolly 1992,Connolly]documents allowing users to explore the structure and content of an information source much like readers explore the contents of a book: by using the table of contents to find the chapters and pages that contain the information they are interested in. The only difference is that in our scenario, page numbers are replaced by hyperlinks and the displayed content is dynamic, i.e., it depends on the query that is producing the result as well as the underlying information source.

In this paper we illustrate our approach to formatting and browsing structured information and describe the implementation of a fully functional prototype called MOBIE (MOsaic Based Information Explorer[1]) that we have built as part of the TSIMMIS (The Stanford IBM Manager of Multiple Information Sources) project [Chawathe et al. 1994,Hammer et al. 1997,Hammer et al. 1995] at Stanford. We argue that this new way of exploring structured information is superior to existing approaches (focusing mainly on providing WWW connections to commercial database servers with little or no result buffering and formatting) and can help databases regain some of the importance they have lost since the advent of the "information superhighway" [DeWitt 1995]. Of course, the advantages of using the WWW as a front-end to databases also bring some challenges along with them. In this paper, we describe in detail how we have solved these challenges and contrast our current solution to how it might be accomplished more elegantly and efficiently in the future using new technologies such as the next generation webservers, browsers capable of executing mobile code, etc.

The remainder of this paper is structured as follows. In Sec. 2 we describe our approach using screen snapshots from a sample database interaction to illustrate the functionalities of MOBIE. Sec. 3 describes in detail how we implemented our prototype. We conclude the paper in Sec. 4 with an evaluation of our current system and some suggestions for how MOBIE could be improved using the latest WWW technologies.

[1] The name MOBIE was coined when Mosaic was the only browser available; it is misleading in that our prototype is not dependent on any one browser.

2. Object Formatting and Browsing in MOBIE

The main idea behind the work described in this paper centers around the need for displaying nested objects in a way that makes it easy for the user to grasp their structure and explore their contents, for example, when viewing the result of a database query. By nested objects we mean objects that contain a top-level (root) object and zero or more subobjects (sometimes referred to as children). Each subobject may itself be a nested object. In general, nested objects are structured like trees (or graphs if we allow cycles). In the TSIMMIS project, we are using an object-based model, called Object Exchange Model (OEM) [Papakonstantinou et al. 1995] for representing nested data. Besides an Object Identifier (OID), type (either atom or complex), and value field, OEM objects also contain a label that describes the contents of an object. For example, a *book* that has a *title*, a *publication date*, and *authors* can be represented as a nested object labeled **book** with subobjects labeled **title**, **publication date**, and **authors**. Furthermore, the subobject `authors` can itself be a nested object that contains the names of the participating authors as subobjects, for example. In MOBIE, labels such as **book**, **title**, **publication date**, etc. are used for identifying and referring to objects that are displayed in the browser window.

Anybody who has worked with nested objects before can attest to the fact that it becomes increasingly difficult to understand the contents of a nested object the more its structure increases in complexity (i.e., the larger the number of sub-objects and the deeper the level of nesting). For this reason, we have built a system that transforms structured data into a "web" of hyperlinked documents that can be viewed using any WWW browser. An object that is selected for viewing is formatted as an HTML document. If the object is a complex object, the document also includes hyperlinks pointing to some or all of the object's substructure depending on the user's preferences. If the object is atomic, it will be displayed by itself. Each document always contains a link to the parent object, unless the selected object is the root of the entire structure. The main contribution of our system is that it gives the user the option to decide which information is to be displayed, how much of the chosen information he[2] wants to see, and when. Information is presented one screen at a time, allowing the user to browse complex objects, which may be too large to view all at once, in a "cafeteria-style" (pick-and-choose) fashion. This approach to browsing nested objects is analogous to how one uses the table of contents to explore the individual chapters of a book.

An important part of the functionality of our system focuses on the layout of information on individual pages. Since this is a process that depends heavily on each user's individual preferences as well as the data that is being displayed, we have paid careful attention to design a system that is flexible enough so that it can be tailored to satisfy many different needs. Our goal was to provide

[2] Because of the lack of neuter personal pronouns in English, the terms "he", "his", etc. are used throughout this paper to refer to an individual who may be either male or female.

users with choices as to how information is to be displayed: from the overall layout of a screen down to the format of an individual object. Initially, the system uses default settings that maximize the amount of information that can be displayed within the given real-estate of the window. The result can then be improved upon by changing the values of *session variables*, which control the document layout, the level of nesting per screen, the number of subobjects per level, etc. By default, these variables control the formatting for the complete object hierarchy. However, by using the label names that refer to a particular object in the hierarchy, the scope of session variables can be limited: from the entire hierarchy, to a specific substructure, to one object. Although customization of the object display can be tedious and time consuming in certain cases, the state of the session variables can be saved on a per-user basis and re-used again during subsequent sessions.

We have implemented a fully functional prototype system called MOBIE which currently provides the graphical interface to TSIMMIS data sources. However, MOBIE is not limited to browsing only data from TSIMMIS but can easily be modified for displaying and formatting structured information from any object-based database[3]. We now describe the functionality of our system in greater detail using screen snapshots from a sample interaction with a TSIMMIS source. The source provides bibliographic information that is represented in the OEM data model. Since we are mainly concerned here with the formatting and browsing capabilities of MOBIE, we start our description when the result is returned from the database, omitting such details as how to connect to the database server, transmission of the query and its results, etc.

2.1 Conventions

Before we go into the details of our sample session, we first describe the general layout of our user interface. Every screen is organized into two basic areas: a navigational bar with buttons that show the allowable options at any given point in the session, and a result screen where the data is displayed. We will explain the functionality of most buttons as we go along. However, it is important to note that the need for these navigational aids arose from the fact that WWW browsers are stateless. Thus MOBIE maintains its own knowledge about the state of a particular session by trapping all MOBIE-related actions in the browser. Using the browser-supplied *Back* and *Forward* buttons bypasses MOBIE's control over the session and causes inconsistencies between the session state inside MOBIE and the actual state. We have much more to say on this in Sec. 3.3.1. When displaying data, we use the following conventions. Object labels are displayed in **bold**, object values are *italicized*. Underlining indicates the existence of a hyperlink.

[3] One can either use a translator for converting data into OEM or modify our algorithms to work with other object-based data models.

2.2 A Sample Session with MOBIE

Let us assume that we have submitted a query asking for all database publications written by author "Widom". Let us also assume that the answer to this query consists of 47 publications that are grouped together under one root object, labeled **answer**. Figure 1 shows a subset of the **answer** object as it is displayed in MOBIE. Each object labeled **document** is a subobject of **answer** as well as a complex object exhibiting additional substructure underneath: the objects labeled **author**, **info**, and **title**. Both the **author** and **title** subobjects are atomic meaning they contain no further substructure. In those cases, the value of the object is displayed with no additional links (except for the parent link). The **info** subobject on the other hand, is a complex object that contains three additional subobjects: **abstract**, **publication**, and **keywords**. Labels belonging to complex objects are underlined meaning that a hyperlink exists that will take the user to another screen containing the subojects.

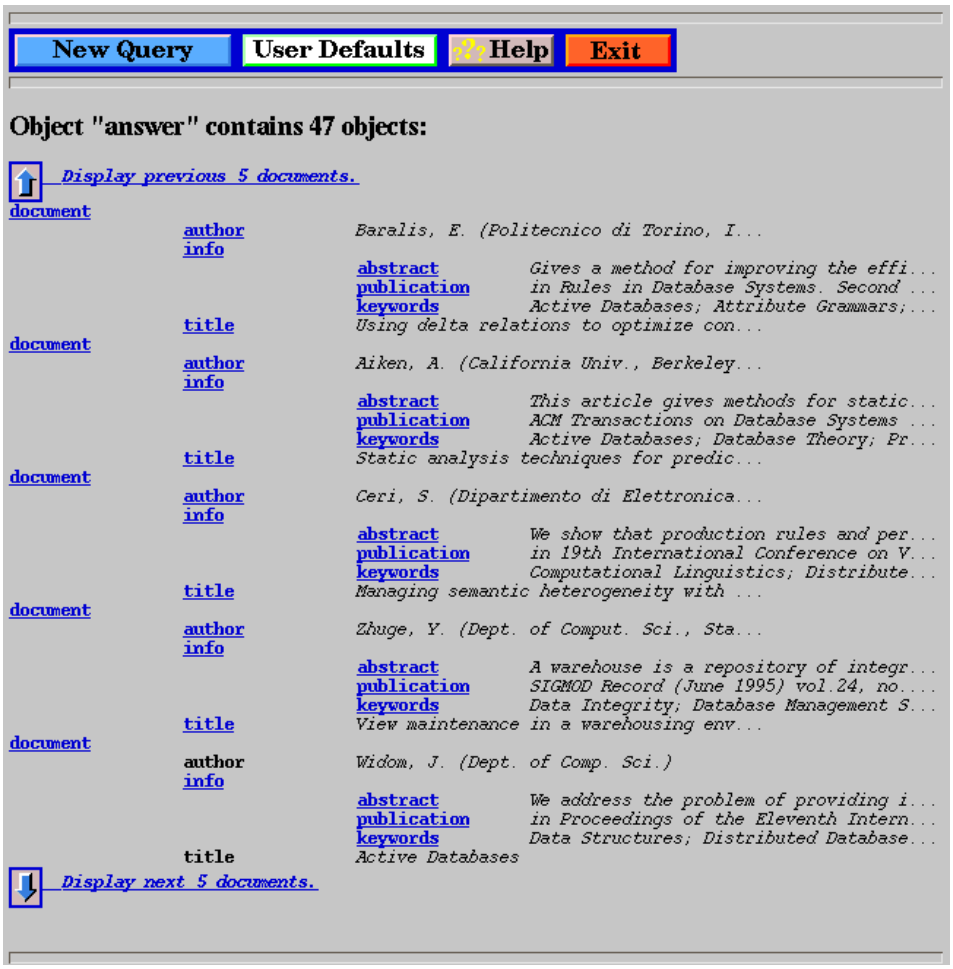

Figure 1: Query result

There are several important details to note on this first screen. Rather then looking at all 47 **document** subobjects in their entirety, only a subset is currently displayed at once. This number (five in this case) can be controlled through a session parameter accessible from a separate screen which is shown later. In order to facilitate navigation, a previous and next button (the up and down arrows on the left side in the figure) are provided to jump to the set containing the previous or subsequent five **documents** respectively. By default, immediate suobjects are displayed in sets of 50. However, in order to improve the readability of our screen snapshots, we have set the value to five. In addition, three levels of nesting are displayed by default, revealing further substructure. This means that for each **document** subobject, its subobjects as well as its sub-subobjects are also displayed. This so-called fan-out level can be controlled through a session parameter as well.

Formatting also affects labels and values. By default, values longer than 40 characters[4] and labels longer than 16 characters are truncated. Truncation is indicated by the appearance of an ellipsis (…) at the end of the string. For truncated values, a hyperlink points to where the complete value can be found. This is similar to a hyperlink that connects a complex object to its subobjects. The display of values and labels can also be controlled through session parameters. Without any user interaction, MOBIE will attempt to display the maximum length up to the default limit. In the example in Figure 1, the values for the objects labeled **author** and **title** belonging to the fifth **document** subobject are displayed in full (hence no hyperlink) since their values are less than 40 characters long. MOBIE will truncate values that would otherwise extend beyond the edge of the window in order to avoid horizontal scrolling.

[4] All values in MOBIE are converted into strings in order to keep the logic of our display routine simple.

Looking at the navigation bar on top of the screen in Figure 1, we see the following four buttons: ***New Query***, which exits the current browse mode and returns to the query input screen (not shown). ***User Defaults***, which provides the user with a fill-ou-form for altering the values of session parameters. We will discuss this option in more detail below. The ***Help*** button displays additional semantic information for the current object (in our case the object labeled **answer**) in case the information is available. In TSIMMIS for example, help information is provided by the data server that is exporting the objects. Finally, the ***Exit*** button terminates a session and initiates miscellaneous clean-up operations within the MOBIE server. In addition, the exit command will guide the user through a sequence of screens for saving the current state of the session parameters in case he hasn't done so on the ***User Defaults*** screen. Session values are currently saved in a user specified file that resides in the file system where MOBIE is running.

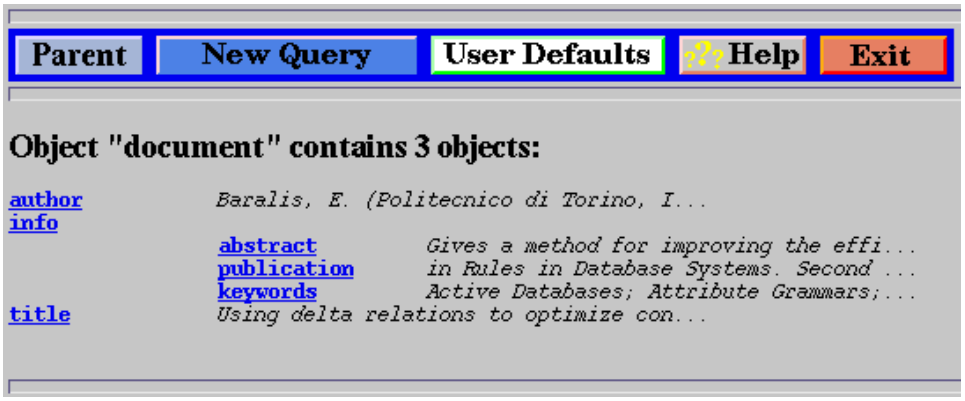

Figure 2: Query result - subobjects

Except for very small answers, i.e., objects with little or no substructure, to see the complete result, the user will move through the structure of the answer object using the activated hyperlinks which take the user to subsequent result pages. The result of clicking on the first **document** subobject in Figure 1 is shown in Figure 2. This screen displays one **document** object by itself but reveals no additional substructure beyond what we have already seen in Figure 1. Everything below the level of **abstract**, **publication**, and **keywords** is an atomic object and their values are too long to be displayed in full. Clicking on the label **abstract,** for example**,** will display the complete value string (see Figure 3). Clicking on the label **info** will zoom in on the info object and display its substructure. Notice that a new button called ***Parent*** has been added to the navigation bar in both Figure 2 and Figure 3. This button allows the user to return to the parent object which is either the **document** object or the **answer** object depending on where we are in the hierarchy.

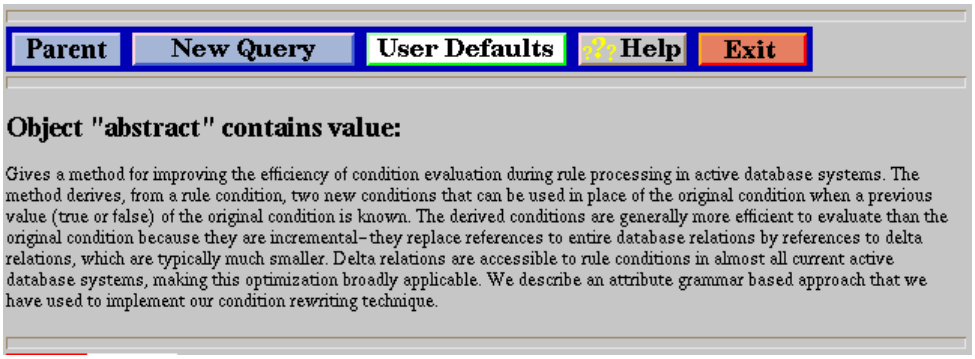

Figure 3: Query result - atomic value

2.3 Formatting Options

As mentioned before, the user can control the formatting of objects through various control parameters. These parameters are called session variables and can be accessed from the ***User Defaults*** screen shown in Figure 4. Our formatting options fall into two categories: "Global Settings", which apply to the whole object structure, and "Label-Based Settings" for which the scope can be specified based on object labels. By default, "Label-Based Settings" affect the complete object structure unless otherwise specified.

Starting from the top, the following global options are currently available (all options are shown with their default values). `Maximum levels of sub-objects` controls the number of visible levels of subobjects for each object that is displayed. The default value is three, meaning that the object as well as its subobjects and sub-subobjects are displayed (if they exist). `Sub-object indentation` controls the amount of indentation used for subobjects The default value is five (characters).

The label-based settings offer the following options. `Default layout` controls the overall "look-and-feel" of the output when it gets displayed in the browser window. As mentioned earlier, we have designed several layout options for how objects can be organized. So far we have shown the "list" layout, where objects are organized in lists, using indentations to indicate nesting. This layout is well suited for displaying any kind of hierarchical data, such as the publication data shown in our screen snapshots, or the contents of a file system, for example. In addition to the list template we have also implemented a "table" layout that displays information in the familiar row and column format.

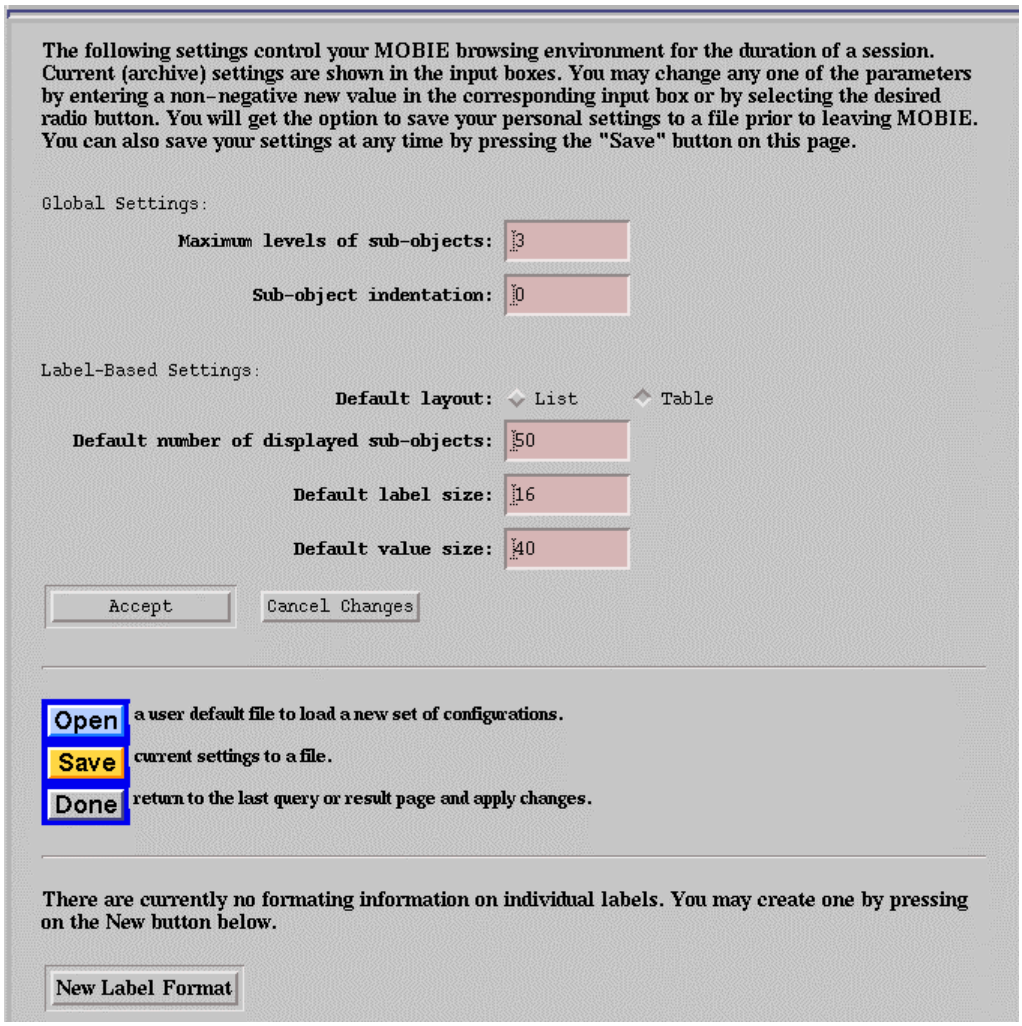

Figure 4: User defaults

This format is best suited for displaying arithmetic data such as financial information or statistical data, for example. Figure 5 shows the **document** object from Figure 2 formatted as a table. Since **document** is a complex object, each row represents one of its subobjects. Labels are shown on the left side. If the subobject is an atomic object, e.g., the subobjects labeled **author** and **title**, the first column starting form the left will contain the subobject's value. If the subobject is a complex object, e.g., the subobject labeled **info**, the first column will be empty and subsequent columns will contain the values of its lower-level subobjects. In case of a complex subobject, the column headings are the labels of the lower-level subobjects. Note, if there are several complex subobjects with different substructure, the table will have different headings for each of the complex subobjects. Also not that it makes no sense to format atomic objects as a table and we do not offer this as an option..

Going back to the user default screen in Figure 4, `Default number of displayed sub-objects` controls the number of first-level subobjects that are displayed on a screen. `Default label size` and `Default value size` control the length of labels and values respectively.

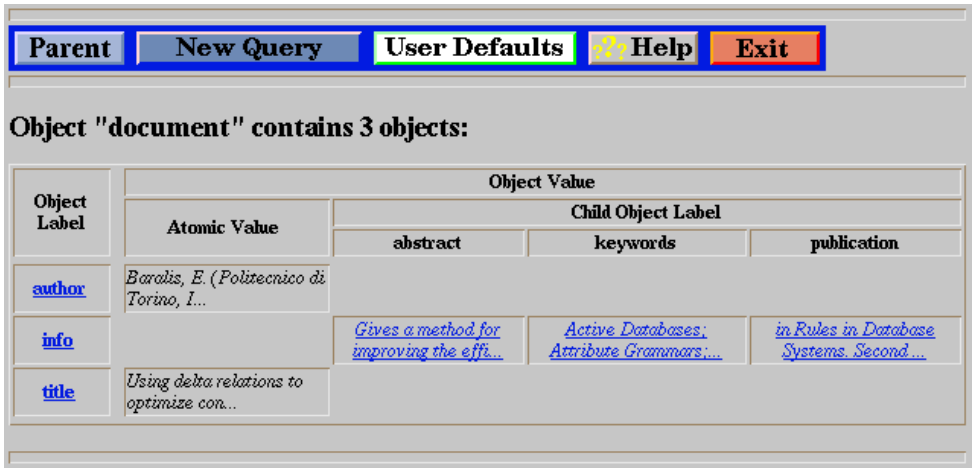

Figure 5: Query result - table layout

As mentioned before, label-based settings either apply to the complete structure or to objects which are identified through their labels. In order to format an object, a formatting choice associated with the label that refers to the desired substructure must be defined. This can be done through the ***New Label Format*** command shown on the bottom of Figure 4. The options that can be specified for the selected label are exactly the same as shown before. Using the ***New Label Format*** option, it is possible, for example, to display three or more levels of nesting for the root object, and then reduce the number of visible levels to just one when viewing its subobjects. As another example, one can display the part of a result that contains numerical values as a table but leave the part that is mostly textual in list format. Finally, the user has the option to save the state of the session parameters using the ***Save*** option near the bottom of the screen so that their values can be re-use in future sessions.

There are many more possibilities for formatting and displaying objects in MOBIE, many of which cannot be adequately described in this limited amount of space. The interested reader is invited to demonstrate our prototype implementation which can be accessed via the TSIMMIS home page at URL http://www-db.stanford.edu/tsimmis/tsimmis.html.

3. Implementation

We now describe the implementation of our system. Specifically, we focus on the challenges that we had to overcome and how we solved them given the technology that was available when we began this project almost two years ago. At that time, we had two options for implementing a WWW-based system that can provide the interactive capabilities illustrated in Sec. 2: either to implement our own WWW server with all the MOBIE-added functionality built-in, or use the existing server technology and design a separately running MOBIE server. Given our limited resources and experience, we opted for the second approach. Since then, many improvements to existing technologies (more extensible WWW servers) as well as new inventions (e.g., Java [Sun Microsystems 1995] and Java Script) have opened up additional possibilities that would allow us to redesign the existing system more efficiently and elegantly if we began today. We have summarized some of our ideas for improving MOBIE in Sec. 4.

3.1 Initial Problems

In general, MOBIE was built to perform the following tasks: (1) buffer and format the results of a query so that the information can be displayed in a WWW browser, and (2), give the user the option to decide which part of the result is to be displayed, how much of the chosen information he wants to see and when. Since the WWW was not intended to support user interactions that require the preservation of the browser's state from request to request, there was no notion of memory in a WWW browser[5] other than the ability to return to a certain number of previously displayed pages that have been cached by the browser. However, most of the browsing functionality in MOBIE depends on information from previous screens (i.e., the current viewing location in the object hierarchy, etc.). Generating and preserving this state was our first main problem. The second problem that we faced had to do with displaying dynamically generated pages (as opposed to static HTML files): each screen has to be generated on-the-fly by MOBIE since its contents are not known until the user takes an action. A brief review of the way the WWW works will further illustrate the problems.

WWW clients and servers communicate using the so-called Hypertext Transfer Protocol (HTTP) [Gettys and Nielson]. The basic procedure for communication between a client and a WWW server is: a client connects to a server and sends a request[6] for a resource; the server sends a response back to the client, supplying both status code as well as the requested resource (if available); the client disconnects. With respect to our first problem, HTTP has the following important characteristics: HTTP is *stateless*, meaning that in between two subsequent requests, the server does not maintain information about the resource being requested, the client who is requesting the resource, the status

[5] One can argue that nowadays, browsers do have a limited amount of memory through the use of "cookies."

[6] A request contains the name of the resource that the client wants to retrieve, the method of retrieval, the name and version of the communication protocol, as well as other various items describing data types, encoding method, etc.

of server's response etc. HTTP is also *connectionless*, allowing each client one request per connection. After a client receives the resource, the server disconnects. Subsequent client requests require a new connection. Given these two characteristics, we therefore need a mechanism that can continuously monitor a session and save its state from one connection to the next. In a sense, we need a transaction capable client rather than just a "request-response" oriented interface.

With respect to the second problem, to support the creation of dynamic documents (created by an external program) HTTP uses CGI [McCool ,NCSA Development Team] (Common Gateway Interface) as a standard that defines the input and output interfaces through which an HTTP server and programs interact. The drawback of CGI is that the HTTP server will not return the output from a program to the browser, until the program has exited and closed all of its standard file descriptors (*stdin*, *stdout*, and *stderr*). Otherwise, the server will assume that more data will be arriving and waits indefinitely. Thus the process in our system that is responsible for generating the output must terminate every time it wants to display a new screen in a client's browser (which happens many times during a typical session).

Based on these requirements, namely to save the state of a session and to terminate the process after the output is ready, we arrived at the following (incorrect) approach: transfer all the data that makes up the state of the current session to a newly started process whenever output needs to be sent to the user; terminate the parent process and continue the session with the child process. However, this solution did not work since is not possible to communicate between an HTTP server and a running process using CGI. Therefore, we needed to split up our design into two components: one continuously running process that monitors and saves the state of a session, and a second process that serves as a "communicator" between the HTTP server and the running process. This communicator process is started every time a new request is generated by the user. Its only purpose is to transfer user requests to the running process and relay the results back the HTTP server. We are aware of the fact that the overhead of starting a new process is quite high compared to swapping a running process in and out of memory, and we have several alternative solutions that we offer to the reader in Sec. 4. There was still one additional hurdle to overcome: the communicator process needs to know on which port to contact the running process. Since the communicator process is started by the HTTP server, the address of the running process must be preserved at the client side in between invocations of the communicator process. Since the client has no memory, the only way to achieve this is to store the port number as a *hidden value* on each generated HTML page. Hidden values can be included in any HTML document using the hidden value tag. We now describe the complete process structure of MOBIE in more detail.

3.2 MOBIE Process Structure

Figure 6 shows the process structure of MOBIE as well as the interactions with our HTTP server, called `www-mobie.stanford.edu`, and with a hypothetical client in form of a WWW browser. A MOBIE session is started by requesting the execution of a CGI executable called MOBIE_SERVER from our HTTP server. MOBIE_SERVER does not take any input parameters and simply initializes the state holder variables for the upcoming session. We discuss the state of a session in more detail in the next section. The output of the MOBIE_SERVER process is always an HTML document. MOBIE_SERVER also initializes the connection to the database server. This connection information is read from a special configuration file. We will not discuss the issues related to submitting and receiving data from a database server in this paper.

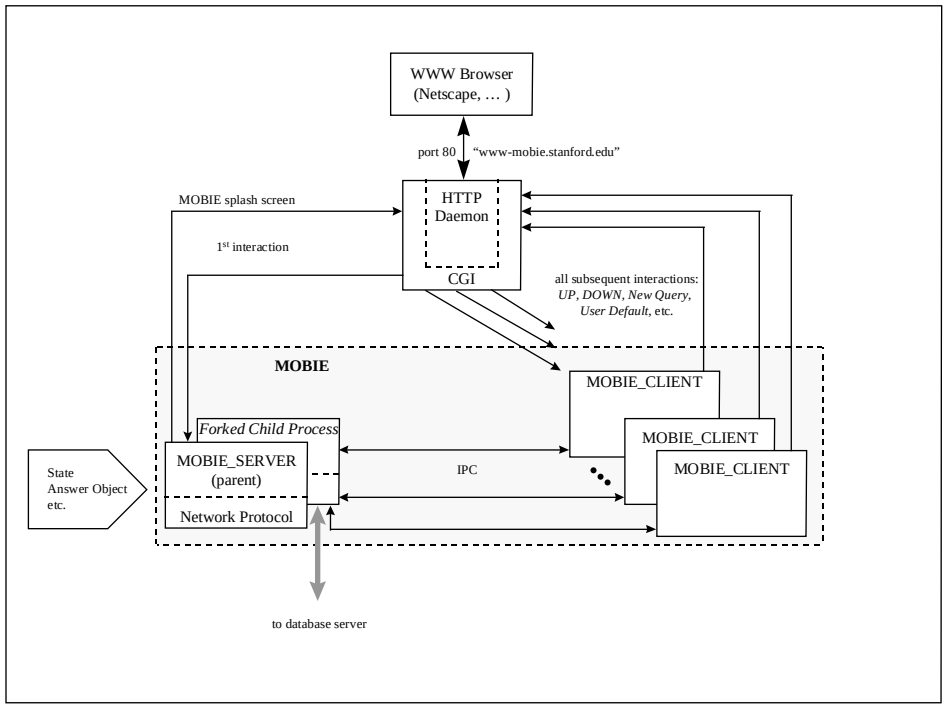

Figure 6: MOBIE process structure

After the initialization phase, MOBIE_SERVER generates a special MOBIE splash screen as its output and forks into two processes, thereby duplicating the state that has been set up so far. The parent process then terminates which causes the HTTP server to display the splash screen in the browser window. The forked child process continues to run in the background and becomes the session monitor for the current session. The port number of the forked MOBIE_SERVER process is attached to the document that makes up the splash screen. This first interaction is depicted in the left side of Figure 6. From this point on, all further interactions with the running MOBIE_SERVER have to be handled through an additional "communicator" process labeled MOBIE_CLIENT in our figure. Every request by the user will start up a new MOBIE_CLIENT

which acts as a relay between the HTTP server and MOBIE_SERVER. MOBIE_CLIENT receives its input as an encoded string which is appended to the URL that is used to make the request. When the HTTP server receives the request, it copies the encoded string into an environment variable called QUERY_STRING. When MOBIE_CLIENT starts, it knows to obtain its input from this variable. Requests are submitted using the GET method which allows the transmission of up to 255 characters as part of the URL. For a technical overview of HTML forms and CGI scripts refer to [December and Ginsburg 1995,Grobe].

The contents of the QUERY_STRING depend on which action the user wants to perform. Several scenarios are possible and we illustrate two with examples.

1. The user fills out an HTML form (e.g., for changing session parameters as shown in Figure 4). In this case, the browser collects the entered data together with our own internal command (`change_defaults`) and appends both to the URL that invokes MOBIE_CLIENT.
2. The user clicks on an object label while browsing. In this case, the URL which has been generated by the previous interaction already contains the internal command (e.g., `fetch_child`) as well as the data that is needed to retrieve the desired object: the current location in the object hierarchy as well as a reference to the object that is to be fetched from MOBIE_SERVER.

Other MOBIE-specific commands include moving up and down in the object hierarchy, requesting help on an object, and terminating a session. In addition, the QUERY_STRING always contains the port number on which MOBIE_CLIENT can communicate with the running MOBIE_SERVER.

After reading the input from QUERY_STRING, MOBIE_CLIENT passes the contents to MOBIE_SERVER by opening an IPC connection using the transmitted port number. MOBIE_SERVER processes the request and returns the output back to MOBIE_CLIENT. After MOBIE_CLIENT terminates, the HTTP server returns the result in the form of an HTML document to the browser. We now describe what happens inside MOBIE_SERVER.

3.3 MOBIE_SERVER

The MOBIE_SERVER process provides the main functionality of our system: a means for storing the state of a session as well as the buffering and formatting of query results.

3.3.1 Session State

In order to provide a context-sensitive WWW interface, our system must be able to keep track of user activities as well as the contents that are displayed in the browser window. Furthermore, there must be a notion of a session with a clearly defined start and end, so that we know when to start monitoring, when to stop, when to flush the object cache, when to clear the saved state, etc. In a

sense, having a session is essential for providing the user with the right context for making navigational requests.

A session starts when MOBIE_SERVER is invoked by the user. In the beginning, the state of a session consists of the user id of the person who has invoked MOBIE, as well as the contents of the session parameters that are used for formatting the results. The session parameters are initialized with their default values. The other state variables which hold the database query, the current pointer into the result set where the user is browsing, and the MOBIE-specific commands are still empty. In addition, the result of a query is also part of the state since it may change after every query. After the MOBIE_SERVER forks, the parent process obtains a port number from the operating system that the child process will use for communication with future MOBIE_CLIENTS. The port number is the only state value that is continuously passed around: from MOBIE_SERVER via MOBIE_CLIENT to the result page inside the browser and back to MOBIE_CLIENT. This is the only way we could achieve connectivity between the short-lived MOBIE_CLIENTS and the continuously running MOBIE_SERVER.

When the session is in progress, the state of the system constantly changes which is reflected in the values of the state variables. Specifically, the state changes when the user submits a query, changes formatting options, navigates through the object hierarchy, or exits. Each of these events needs to be "trapped" and an alert must be sent to MOBIE_SERVER so that it can update the state. State-changing events occur when the user (1) traverses a hyperlink in the object structure, e.g., for "zooming" in on certain subobjects, (2) submits an HTML form, e.g., in order to change session parameters, or (3), clicks on one of the MOBIE-supplied buttons, e.g., the EXIT button in the navigation bar or the UP and DOWN buttons in the result area.

However, there is an implicit danger in the way we monitor the state of a session. Since we are not disabling the native functionality of the browser, we have no real control over every single activity that may occur during a session. For example, if a user decides to use the browser's own *Back* or *Forward* buttons or leaves the session completely by jumping to a new site, the state inside MOBIE_SERVER becomes inconsistent with the real state on the client side. Furthermore, since we have no control over the contents of the cache of the browser, it may be possible for the user to return to pages that contain query results that are no longer buffered inside MOBIE_SERVER.

In order to distinguish between requests for valid data (i.e., the objects that are currently buffered by MOBIE_SERVER) and stale data (objects from the browser's cache), some of the session state is also stored on each result page as hidden values (similar to the port number). By comparing the state in the hidden values with the actual state in MOBIE[7], our system can determine if it can service

[7] The actual comparison is based on a checksum value that is the result of OR'ing the values of the state variables with a number that is unique to the current session.

the request or not. In the latter case, an error is generated. Note that using the browser's native buttons to move back or forward in a session does not necessarily cause MOBIE_SERVER to raise an error. If the cached screen refers to objects that are still in the buffer, MOBIE_SERVER will use the current pointer (which is stored as a hidden value inside the cached HTML document) to determine the reference location for the request. In the case when the user unexpectedly leaves the session, bypassing the ***Exit*** command, MOBIE_SERVER will continue running until it is eventually terminated by an internal time-out.

A session ends properly when the user clicks on the ***Exit*** button in the navigation bar. This will cause MOBIE_SERVER to clear its buffer and release all the memory that was claimed during the session. After the clean-up is performed, MOBIE_SERVER terminates.

3.3.2 Result Formatting and Browsing

The second service provided by MOBIE_SERVER is the buffering and formatting of query results to support object browsing. As mentioned before, MOBIE_SERVER stores the complete result for the current query in memory, giving users instant access to any part of the object hierarchy. However, we realize that fetching only complete answers may sometimes result in suboptimal performance, namely when the result is large and/or the connection to the database server is slow. For these cases, it may make sense to implement some sort of partial fetch scheme that only returns a certain number of objects (if the result is suspected to be large) or those components of the answer that are available within a certain amount of time. Missing objects are replaced by stubs that indicate to the browser that it has to go to the source rather than its memory for retrieving the answer. We have implemented partial-fetch in the TSIMMIS protocol that is used for communication between MOBIE and TSIMMIS sources.

Object browsing in MOBIE is implemented as follows: MOBIE_SERVER keeps the complete object structure in its memory. As an initial result, it will return to the client a top-level view of the root object together with a default number of subobjects. Additional information can be requested by the user and is then fetched from MOBIE_SERVER. In a sense, MOBIE provides the user with a "window" through which he can see a pre-determined subset of the result. Each navigational request contains the location of where the user has positioned his window in the object hierarchy (i.e., the current pointer), the size of the window (i.e., the values of the session parameters that control number of visible objects), as well as the desired navigational action. A navigation action is one of `fetch_parent`, `fetch_child`, `up`, and `down`.

There is an implicit ordering associated with each level in the object hierarchy of an OEM object. This ordering allows the client to request the n^{th} subobject for a given parent, for example. When MOBIE_SERVER receives a request involving navigation, it will use the current pointer as well as the "window size" to determine at which part of the object hierarchy the user is currently looking.

Then MOBIE_SERVER executes the action given the current location as a reference. The result is a new location in the object hierarchy which is used as the root for calculating the new contents of the browser window. This is done by retrieving the values of the session parameters that determine the "window size". For example, the `fetch_parent` action, moves the "window" up one level in the hierarchy. The `fetch_child` action, which must always include the number of the child to be fetched, repositions the "window" using the new child as the current root object. The `up` action, repositions the window on the current level and returns the set of subobjects that occur before the currently displayed set. For example, if currently subobjects 11-20 are visible and the user clicks on the up button, MOBIE_SERVER repositions the window so that it will display subobjects 1-10 (using the session variable `number_of_subobjects` to determine the size of the set). The `down` action positions the window in the other direction. It is important to note that this navigation only works in a system that maintains session state. As a result, a user can browse the results of his query without ever having to resubmit the query, assuming unlimited memory at the server side. This is a major improvement over most WWW-based interfaces to database servers which are still operating under the stateless request-response paradigm.

Once MOBIE_SERVER has determined what the new contents of the browser window should be, it formats the selected objects using standard HTML commands. However, the look-and-feel of the result depends on the state of the session variables, e.g., list or table layout, length of values and labels, etc. In order to generate output, MOBIE_SERVER initializes a temporary result buffer for holding the formatted result, complete with HTML tags and URLs to the rest of the structure. To anybody who was inspecting the contents of this buffer, it would look like the source file for a static HTML page. When the formatting is complete, MOBIE_SERVER returns the contents of its result buffer to MOBIE_CLIENT. MOBIE_CLIENT immediately passes the contents to *stdout* which is where the HTTP server is waiting to receive output. When MOBIE_CLIENT exits, the HTTP server knows the result is complete and sends the formatted HTML document to the client for display.

4. Evaluation and Future Improvements

To summarize our work, MOBIE is a user-friendly, WWW-based interface to databases that supports browsing of nested objects. Specifically, MOBIE buffers the result of a database query and gives the user the ability to explore the entire object hierarchy, which is formatted as an interconnected "web" of HTML documents, by clicking on the appropriate hyperlinks in a browser. Users can customize the formatting of their query results by choosing from a menu of formatting options that control such characteristics as layout of objects on the screen, number of objects that are visible at

the same time, level of nesting, etc. By using MOBIE, the user can decide which information is to displayed, how much of the chosen information he wants to see, and when.

4.1 Related Work

Despite the relatively young age of the World Wide Web (approx. six years) there has already been extensive research on bringing together the field of database systems with the Web. This work can be categorized into several areas depending on the intended usage as follows:

- Web organization: Text-based indexes for keyword searches such as WAIS [Kahle and Medlar 1991], for example, W3Objects [Ingham et al. 1996], etc. Here, the focus is not so much on dynamically translating and formatting data into HTML documents but rather on using database technologies for organizing site addresses and their contents and for providing an efficient directory service that can keep up with the seemingly boundless growth of the Web.
- There are numerous projects that attempt to connect a variety of information sources (including legacy sources, relational and object-oriented database systems, etc.) to the Web in order to allow wide-spread and easy access to large amounts of queriable, electronic information (e.g., [Varela et al. 1995], [Dossick and Kaiser 1996], [Hadjiefthymiades and Martakos 1996]). Here the focus is on using the Web as an ubiquitous front-end to databases and on dynamically formatting and displaying database objects as HTML pages. Specifically, Card et al. [Card et al. 1996] have developed two new ways of viewing and organizing WWW data based on information foraging theory from ecological biology.
- In addition, some vendors, such as Microsoft, for example, provide proprietary solutions for dynamically publishing COM/DCOM objects stored in Microsoft Access or Excel on the Web (see Microsoft DBWeb, for example).
- An important area of research is that of keeping stateful connections with the information source. Since HTTP is a stateless and connectionless protocol, there is a need for providing stateful information services [Perrochon and Fisher 1995, Putz 1994].

4.2 Contributions

The main contribution of our work is twofold: First and foremost, MOBIE provides a new way of viewing and exploring the contents of a database, allowing users to customize their views depending on the contents and structure of the data. Our model is particularly well suited for object-based systems with deeply nested objects allowing users to zoom in and out of a particular substructure as necessary. Current database front-ends typically attempt to display nested objects in a flat view, making it difficult for users to grasp the contents and structure of their result.

Secondly, MOBIE narrows the gap between the database camp and the WWW, which up until now has ignored database technologies and built an information space that still relies mainly on the file system for persistent storage. Experts from both camps agree that database technology and the WWW should go together, but so far, most approaches have stopped short of providing a real solution. Although there are numerous approaches to connecting databases to the WWW, we are not aware of any system that provides the same *generic* solution to formatting and displaying persistent data as HTML documents in the same way MOBIE does.

4.3 Future Work

MOBIE is a fully functional prototype that we are using for browsing OEM objects in the TSIMMIS project. It has met all of our initial expectations in terms of performance and usability. For example, browsing object structures that contain several hundreds of subobjects with many levels of nesting is as easy as looking at a small object with little substructure. However, given the state-of-the-art in WWW technology, it is now possible to enhance MOBIE in the areas of performance, scalability, as well as usability, something we were unable to do just a short year ago. In the next sections, we briefly outline some of the upgrades that we are proposing using criteria such as usability, performance, and scalability to evaluate each proposal. For a comprehensive collections of technical reports and specifications describing the latest WWW technologies, refer to the W3 Consortium's publication's page [The World Wide Web Consortium (W3C)].

1. We consider improvements to the usability of our system the most important kind of enhancement. For example, we are experimenting with additional layouts, such as a tree-structured layout, for example, that would allow users to view some hierarchical data in a more natural top-to-bottom way. In addition, we have noticed that users tend to spend a lot of time trying to customize the formatting of their result sets. This process could be automated by implementing so-called "wizards" which would generate HTML templates in way that is similar to visual programming: by assembling templates for what a screen should look like using the available building blocks from a menu, for example.

Items 2-8 focus mostly on improvements to the efficiency of our implementation.

2. Manage user sessions in one single MOBIE_SERVER process rather than starting a new one per user. MOBIE_SERVER would probably be multi-threaded containing the state for each session in an efficient data structure for fast access (e.g., a hashed heap). Controlling multiple sessions with one MOBIE_SERVER reduces the number of processes that are running which is one of the factors that limits the scalability of our current approach. In addition, by managing multiple users from within one process, the query results can be shared (see option 2 below).
3. Buffer more than one query result in MOBIE_SERVER. Currently, a user can only navigate through the result of the most recently asked query. However, saving the results of the last *n* queries and allowing the user to select which query to browse will enhance performance (reduced number of roundtrips to the source) as well as the usability (less wait time for source data to arrive in the cache) of our system. The latter is especially true if one combines proposal 1, multiple user sessions per process, with this one. As a result, users could choose from a set of available answers which makes good sense if there is a set of frequently asked queries for a group of users. The drawback is that this approach will not work well for highly dynamic sources, i.e., where the contents change quickly. In those cases, the buffer contents have to be refreshed too often, erasing the time that was initially saved.
4. Add secure connections, for example, for browsing sensitive data. Security can be achieved in many ways and at many levels. The most apparent security loop hole involves the HTTP server. Security can be improved. by using a secure server and certificates, for example. Providing security in mobie would enhance the usability for a certain group of users.
5. Remove the MOBIE_CLIENT process. The use of a separate "relay" process for connecting the HTTP server with an already running process, is very inefficient, especially since MOBIE_CLIENT is invoked many times during a session. New commercial HTTP servers from Microsoft or Netscape, for example, provide libraries that allow for a direct IPC connection between the server and running processes resulting in higher efficiency.
6. Add client-side query parsing and input field validation using Java Script. Currently, the client browser just gathers user input and transmits the unfiltered data back to MOBIE_SERVER for processing. So, when the user enters an invalid input (e.g., a negative number for a session parameter or a query with syntax errors), the mistake will not be detected until MOBIE_SERVER processes the input. Using Java Script, however, some of the intelligence that currently resides exclusively on the server side could be relocated to the client. In this case, the client could filter out some of the simple mistakes reducing the number of connections that have to be made to MOBIE_SERVER.
7. Re-implement MOBIE as a Java applet. This is the most drastic proposal but also promises the highest degree of improvements. The new Java-based MOBIE would be downloaded to a WWW

client, where it executes in the user's process space. From this point on, there is no further connection to the HTTP server needed when interacting with MOBIE. MOBIE will connect directly to the database and download the result into the users's filesystem where it resides for the duration of the session. All formatting and browsing occurs locally and no state holder process is necessary. This is the most efficient of all implementations and the options for enhancing usability are so far only limited by the capabilities of the Java language.

Acknowledgments

We are grateful to Hector Garcia-Molina, Jennifer Widom, Kai Hwang, Jiang Wu, and entire TSIMMIS team for numerous fruitful discussions and comments.

Bibliography

[Berners-Lee and Connolly 1992] T.J. Berners-Lee, and D.W. Connolly. "Hypertext Markup Language - 2.0." , HTML Working Group of the Internet Engineering Task Force, 1992.

[Card et al. 1996] S.K. Card, G.G. Robertson, and W. York. "The WebBook and the We Forager: An Information Workspace for the Wold-Wide Web." In *Proceedings of the Conference on Human Factors in Computing Systems*, 1996.

[Chawathe et al. 1994] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. "The TSIMMIS Project: Integration of Heterogeneous Information Sources." In *Proceedings of the Tenth Anniversary Meeting of the Information Processing Society of Japan*, Tokyo, Japan, 7-18, 1994.

[Connolly] D.W. Connolly. "Hypertext Markup Language." URL, http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[December and Ginsburg 1995] J. December, and M. Ginsburg.*HTML and CGI Unleashed*, Sams.net Publishing Company, 1995.

[DeWitt 1995] D.J. DeWitt. "DBMS—Roadkill on the Information Superhighway?" In *Proceedings of the International Conference on Very Large Databases*, Zürich, Switzerland, 1995.

[Dossick and Kaiser 1996] S.E. Dossick, and G.E. Kaiser. "WWW Access to Legacy Client/Server Applications." In *Proceedings of the Fifth International World-Wide Web Conference*, Paris, France, 1996.

[Gettys and Nielson] J. Gettys, and H.F. Nielson. "HTTP—Hypertext Transfer Protocol." URL, http://www.w3.org/pub/WWW/Protocols/.

[Grobe] M. Grobe. "An Instantaneous Introduction to CGI Scripts and HTML Forms." URL, http://kuhttp.cc.ukans.edu/info/forms/forms-intro.html.

[Hadjiefthymiades and Martakos 1996] S.P. Hadjiefthymiades, and D.I. Martakos. "A generic framework for the deployment of structured databases on the World Wide Web." In *Proceedings of the Fifth International World-Wide Web Conference*, Paris, France, 1996.

[Hammer et al. 1997] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. "Extracting Semistructured Information from the Web." In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, 1997.

[Hammer et al. 1995] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. "Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System." In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, California, 483, 1995.

[Ingham et al. 1996] D. Ingham, S. Caughey, and M. Little. "Fixing the "Broken-Link" Problem: The W3Objects Approach." In *Proceedings of the Fifth International World-Wide Web Conference*, Paris, France, 1996.

[Kahle and Medlar 1991] B. Kahle, and A. Medlar. "An Information System for Corporate Users: Wide Area Information Servers." *Connexions—The Interoperability Report*, 5(11), 2-9, 1991.

[McCool] R. McCool. "The Common Gateway Interface." URL, http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[NCSA Development Team] NCSA Development Team. "The Common Gateway Interface." URL, http:/hoohoo.ncsa.uiuc.edu/cgi/intercace.html.

[Papakonstantinou et al. 1995] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. "Object Exchange Across Heterogeneous Information Sources." In *Proceedings of the Eleventh International Conference on Data Engineering*, Taipei, Taiwan, 251-260, 1995.

[Perrochon and Fisher 1995] L. Perrochon, and R. Fisher. "IDLE: Unified W3-Access to Interactive Information servers." In *Proceedings of the Third International Conference on the World-Wide Web*, Darmstadt, Germany, 1995.

[Putz 1994] S. Putz. "Interactive information services using World-Wide Web hypertext." In *Proceedings of the First International Conference on the World-Wide Web*, CERN, Geneva, Switzerland, 1994.

[Sun Microsystems 1995] Sun Microsystems. "The Java Language." Technical Report, Sun Microsystems, 1995.

[The World Wide Web Consortium (W3C)] The World Wide Web Consortium (W3C). "The World Wide Web Consortium: Technical Reports and Publications." URL, http://www/w3/org/pub/WWW/TR/.

[Varela et al. 1995] C. Varela, D. Nekhayev, P. Chandrasekharan, C. Krishnan, V. Govindan, D. Modgil, S. Siddiqui, O. Nickolayev, D. Lebedenko, and M. Winslett. "DB: Browsing Object-Oriented Databases over the Web." In *Proceedings of the Fourth International World Wide Web Conference: "The Web Revolution"*, Boston, Massachusetts, 1995.