

23 AUG 2004



166166



G00002

1

A METHOD AND APPARATUS DEFINING A COMPONENT
MODEL FOR CREATING DYNAMIC DOCUMENTS IN
A DISTRIBUTED DATA PROCESSING SYSTEM

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates generally to an improved data processing system and in particular to a method and apparatus for creating dynamic documents. Still more particularly, the present invention provides a method and apparatus for providing reusable functionality for dynamic XML documents in a distributed data processing system.

2. Description of Related Art:

The Internet, also referred to as an "internetwork", is a set of computer networks, possibly dissimilar, joined together by means of gateways that handle data transfer and the conversion of messages from the sending network over the protocols used by the receiving network (with packets if necessary). When capitalized, the term "Internet" refers to the collection of networks and gateways that use the TCP/IP suite of protocols.

The Internet has become a cultural fixture as a source of both information and entertainment. Many businesses are creating Internet sites as an integral part of their marketing efforts, informing consumers of the products or services offered by the business or providing

other information seeking to engender brand loyalty. Many federal, state, and local government agencies are also employing Internet sites for informational purposes. Further, the Internet is becoming increasingly popular as a medium for commercial transactions.

Currently, the most commonly employed method of transferring data over the Internet is to employ the World Wide Web environment, also called simply "the Web". In the Web environment, servers and clients effect data transactions using the Hypertext Transfer Protocol (HTTP), a known protocol for handling the transfer of various data entities or files (e.g., text, still graphic images, audio, motion video, etc.). The information in various data files is often formatted for presentation to a user by a standard page description language, the Hypertext Markup Language (HTML). In addition to basic presentation formatting, HTML allows developers to specify "links" to other Web resources identified by a uniform resource locator (URL). A URL is a special syntax identifier defining a communications path to specific information. Each logical block of information accessible to a client, called a "page" or a "Web page", is identified by a URL. The URL provides a universal, consistent method for finding and accessing this information using a Web browser. A browser is a program capable of submitting a request for information identified by a URL at the client machine. Retrieval of information on the Web is generally accomplished with an HTML-compatible browser. The Internet also is widely used to transfer applications to

users using browsers.

Applications are provided on the Web through the use of scripts and programs known as "applets", which may be embedded as objects in HTML documents on the Web.

Applets are Java programs that may be transparently downloaded into a browser supporting Java along with the HTML pages in which they appear. These Java programs are network and platform independent.

Through the use of documents, such as Web pages, and Web applications, businesses and other organizations have provided applications for use on the Web. Much of the business logic in today's Web applications is done by a combination of scripting languages embedded in HTML, and Web server extensions through Common Gateway Interface (CGI), Internet Server Application Programming Interface (ISAPI), Java servlets, Web server plug-ins, Distributed Component Object Model (DCOM), Enterprise JavaBeans (EJB), or Common Object Request Broker Architecture (CORBA) objects. This presents a complex solution to Web application developers because of the large number of technologies that the developer must integrate in each module of the application. Scripting in HTML and dynamic HTML (DHTML) is ad-hoc and is not based on any formal computing model. As a result, HTML documents that use scripting to provide business logic, view rendering, and dynamic user interfaces are complicated, difficult to develop, browser sensitive, difficult to maintain, and provide limited opportunity for reuse.

Extensible markup language (XML) is another markup

language used for interchanging documents on the Web. Currently there is no standard for creating dynamic XML documents or scripting support in XML. Extensible stylesheet language (XSL) can be used to transform an XML document into another XML document, but this is a one-shot process. Once the transformation is done, the result is a static document. XSL also does not provide an effective scripting solution.

Some proposed mechanisms for providing functionality in documents are DHTML, Active Server Pages (ASP), Java Server Pages (JSP), and Microsoft Internet Explorer 5 (IE5) behaviors. DHTML supports ad-hoc scripting, which can result in documents that have unpredictable structures and are difficult to maintain.

ASPs and JSPs address the same problem, but through the use of template languages. ASPs and JSPs require the document author to switch into the programming domain to invoke functions. HTML authoring tools help simplify the development of ASPs and JSPs, but these tools still mix very different domains.

Microsoft Corporation's IE5 behaviors provide a way to associate script functions with XML elements through Cascading Style Sheet (CSS) style sheets. Behaviors are fragments of scripting language declared as a CSS class. Invocation of IE5 behavior methods are implicit based on an HTML or XML element having the class associated with the method. Behaviors added to XML elements are not consistent with their schema. The same XML element might have completely different and inappropriate behavior

inherited from some other CSS stylesheet. Behaviors in IE5 do not support remote procedure calls. IE5 behaviors merely let stylesheet authors associate methods, computed properties, and/or events to any HTML or XML element, and define them in a separate document.

Therefore, it would be advantageous to have an improved method and apparatus for providing and creating documents on a distributed data processing system.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus in a data processing system for creating a dynamic electronic document in a markup language. Static content in the markup language is placed into the electronic document. A function invocation mechanism is placed in the electronic document using the markup language, wherein the function invocation mechanism includes a function name and parameters required to invoke a function.

In response to activating the function invocation mechanism, the markup language (e.g. XML) requesting the function execution is replaced by markup language representing the result. For example, a document in a markup language, such as HTML, may be created and later displayed on request that introspects the function definition and displays a form to prompt for the input parameters for the function. When these parameters are received, the function is invoked by posting a request to another XML document that invokes the function. The function in this example is invoked by retrieving a document containing markup language requesting the function invocation, placing the parameters into the function, and executing the function. The result from the execution of the function replaces the request to invoke the function with the markup language result representing the returned value(s). The document may then be formatted and returned for presentation. This

23 AUG 2004

7

formatting may be accomplished by applying a stylesheet
to the document.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a pictorial representation of a distributed data processing system in which the present invention may be implemented;

Figure 2 is a block diagram of a data processing system that may be implemented as a server in accordance with a preferred embodiment of the present invention;

Figure 3 is a diagram illustrating components used in dynamic extensible markup language (DXML) documents in accordance with a preferred embodiment of the present invention;

Figure 4 is a diagram of a function definition in accordance with a preferred embodiment of the present invention;

Figure 5 is a diagram illustrating a document containing a function definition in accordance with a preferred embodiment of the present invention;

Figure 6 is a diagram illustrating code for invoking a function in accordance with a preferred embodiment of the present invention;

Figure 7 is an example of code for executing native

functions, whose implementation is not included with the function definition, through an execute element in accordance with a preferred embodiment of the present invention;

Figure 8 is an example of syntax for returning an attribute value in accordance with a preferred embodiment of the present invention;

Figure 9 is a flowchart illustrating processes used to execute a function as shown in Figure 3 in accordance with a preferred embodiment of the present invention;

Figure 10 is a diagram illustrating data flow between various components in accordance with a preferred embodiment of the present invention;

Figure 11 is an example XML document that imports a function definition that when used with an XSL stylesheet may be used to display an HTML form in accordance with a preferred embodiment of the present invention;

Figures 12A and 12B are an example of an XSL stylesheet used for rendering the XML document in Figure 11 in accordance with a preferred embodiment of the present invention;

Figures 13A and 13B are an illustration of an XML document containing an execute element in accordance with a preferred embodiment of the present invention; and

Figures 14A and 14B are a diagram illustrating an XSL stylesheet for displaying the XML document in Figures 13A and 13B in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

I. Environment

With reference now to the figures, **Figure 1** depicts a pictorial representation of a distributed data processing system in which the present invention may be implemented. Distributed data processing system 100 is a network of computers in which the present invention may be implemented. Distributed data processing system 100 contains a network 102, which is the medium used to provide communications links between various devices and computers connected together within distributed data processing system 100. Network 102 may include permanent connections, such as wire or fiber optic cables, or temporary connections made through telephone connections.

In the depicted example, a server 104 is connected to network 102 along with storage unit 106. In addition, clients 108, 110, and 112 also are connected to network 102. These clients 108, 110, and 112 may be, for example, personal computers or network computers. For purposes of this application, a network computer is any computer, coupled to a network, which receives a program or other application from another computer coupled to the network. In the depicted example, server 104 provides data, such as boot files, operating system images, and applications to clients 108-112. Clients 108, 110, and 112 are clients to server 104. Distributed data processing system 100 may include additional servers, clients, and other devices not

shown. In the depicted example, distributed data processing system 100 is the Internet with network 102 representing a worldwide collection of networks and gateways that use the TCP/IP suite of protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, government, educational and other computer systems that route data and messages. Of course, distributed data processing system 100 also may be implemented as a number of different types of networks, such as for example, an intranet, a local area network (LAN), or a wide area network (WAN). Figure 1 is intended as an example, and not as an architectural limitation for the present invention.

The present invention may be implemented within distributed data processing system 100 to provide a component model to distribute documents and provide dynamic logic or functions in the documents using reusable components. In these examples, the model provided by the present invention is centered around documents and based on XML. Additionally, the model is language neutral, location independent, and platform independent. Documents may be distributed to various users at clients, such as clients 108-112. The functions may be located on these clients or on other systems, such as server 104. Network 102 also may contain additional servers. DXML documents may be communicated from server-to-server for different applications.

Referring to **Figure 2**, a block diagram of a data processing system that may be implemented as a server, such as server 104 in **Figure 1**, is depicted in accordance with a preferred embodiment of the present invention. Data processing system 200 may be a symmetric multiprocessor (SMP) system including a plurality of processors 202 and 204 connected to system bus 206. Alternatively, a single processor system may be employed. Also connected to system bus 206 is memory controller/cache 208, which provides an interface to local memory 209. I/O bus bridge 210 is connected to system bus 206 and provides an interface to I/O bus 212. Memory controller/cache 208 and I/O bus bridge 210 may be integrated as depicted.

Peripheral component interconnect (PCI) bus bridge 214 connected to I/O bus 212 provides an interface to PCI local bus 216. A number of modems may be connected to PCI bus 216. Typical PCI bus implementations will support four PCI expansion slots or add-in connectors. Communications links to network computers 108-112 in **Figure 1** may be provided through modem 218 and network adapter 220 connected to PCI local bus 216 through add-in boards.

Additional PCI bus bridges 222 and 224 provide interfaces for additional PCI buses 226 and 228, from which additional modems or network adapters may be supported. In this manner, data processing system 200 allows connections to multiple network computers. A memory-mapped graphics adapter 230 and hard disk 232 may

also be connected to I/O bus 212 as depicted, either directly or indirectly.

Those of ordinary skill in the art will appreciate that the hardware depicted in Figure 2 may vary. For example, other peripheral devices, such as optical disk drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is not meant to imply architectural limitations with respect to the present invention.

The data processing system depicted in Figure 2 may be, for example, an IBM RISC/System 6000 system, a product of International Business Machines Corporation in Armonk, New York, running the Advanced Interactive Executive (AIX) operating system. Further, the data processing system illustrated in Figure 2 also may be used to implement a client. A client may take various forms, such as, for example, a notebook computer, a personal digital assistant (PDA), a hand held computer, a kiosk, or a Web appliance.

II. DXML

The present invention recognizes that in order to support applications, such as e-business on the Web, a component model encapsulating the state and behavior of reusable components is needed. Without a component model, client and business applications are difficult and expensive to write, as each application must mine the data and recreate its meaning and behavior.

Interoperability is compromised by applications that reinvent the meaning of data producing silos of automation that are difficult to integrate. Data integrity is compromised because each application accesses raw data and may interpret it differently. Finally, reuse is severely limited.

The component model of the present invention adds capability and functionality in document creation in a manner that is familiar to developers and consistent with common practice. This mechanism makes the component model accessible to a broad authoring community. The mechanism of the present invention keeps the roles of the document author, document schema author, and document layout author separate to maximize the use of role-specific skills. As used herein, "schema" is the definition of a document's structure.

The present invention focuses, in these examples, on a document centric component model based on XML. No standard is currently present for creating dynamic XML documents, or scripting support in XML. XSL can be used to transform an XML document into another XML document, but this is a one-shot process. Once the transformation is complete, the result is a static document. XSL also does not provide an effective scripting solution.

The present invention provides a method, apparatus, and computer implemented instructions for a component model for the Web. In the depicted examples, the Web component model is formed from an XML scheme and function definitions. The component model, in these examples, is

dynamic XML (DXML). In these examples, the DXML is a combination of XML and scripting. XML elements have a tag name, a set of attributes, and associations with other elements including containment through their content model and general associations through XLink and XPointer. DXML extends XML by allowing elements to also have methods that return either other XML elements or character data. DXML combines extensions to an XML schema for defining methods on XML elements, common scripting languages for specifying the implementation logic of the element methods, simple object access protocol (SOAP) for function invocation, and XML to define a document centered component model for the Web. In these examples, a new XML mechanism is used to invoke functions. Of course, SOAP and other known function invocation mechanisms may be used for function invocation in accordance with a preferred embodiment of the present invention.

One advantage of the present invention is that it focuses on enabling XML document authors to use behavior defined by scripting functions, as well as Java and other component models without requiring them to use programming languages.

By combining standard scripting languages in a formal way with XML, DXML meets the requirements for a Web component model as described above. Because the Web component model of the present invention is based on XML and implements functions using common scripting languages, this model will be familiar to current HTML

authors who use tag languages and scripting. Web application complexity is reduced, and maintainability is increased by separating schema and logic development, business data, and rendering instructions while integrating them through a formalized programming model. DXML allows functions to be implemented either using scripting languages or as native methods of some distributed service. DXML uses XML to invoke either scripts or the services of distributed components in a manner that is independent of the component model used to implement the services. In essence, DXML creates a component model that is ideal for Web applications by combining the notion of methods on XML elements with common scripting languages and remote procedure calls invoked using XML.

III. Data Flow

With reference now to Figure 3, a diagram illustrating data flow used in processing dynamic extensible markup language (DXML) documents is depicted in accordance with a preferred embodiment of the present invention. In this example, client 300 accesses and requests information from server 302 through the use of DXML documents. This information may be, for example, a request to identify overpaid employees. The communication between client 300 and server 302 occurs using HTTP.

A browser on client 300 is used to generate a request for execution of a function. In particular, client 300 generates a DXML document URL (step m1), which

is sent to get function 306 in server 302. This DXML function document URL may be sent to get function 306 through various mechanisms. For example, a user at client 300 may select a link or hyperlink in a Web page displayed on Web browser 304 within client 300. The request link may be one that requests a function, such as a function identifying overpaid employees in a company. In response to selecting such a link, the URL associated with this particular function is returned to get function 306.

In response, get function 306 obtains the XML document used to implement the DXML function corresponding to the URL from Web server repository 308 within server 302 (step m2). In this case, server 302 is configured to translate all XML documents (documents having MIME type text/xml or application/xml) to HTML using translate function 310 in order to be displayed in browsers that are not XML/XSL enabled. All documents are available from the repository. The particular configuration used depends on the Web server used. For example, IBM WebSphere AppServer and servlet filtering may be used to configure the translation.

If a browser capable of rendering XML using XSL, such as IE5, is employed, then this translation step on the server is not required. In this case, the result of get function 306 is chained to translate function 310 (step m3). Translate function 310 converts the DXML document into an HTML document based on the instructions in the referenced XSL stylesheet (referenced in the DXML document). Then, this HTML document is returned to Web browser 304 as the result of get function 306 (step m4).

In response to obtaining the DXML function document, this DXML document is sent to translate function 310,

which translates or transforms the DXML document into an HTML document (step m5). This is accomplished by obtaining an XSL stylesheet from Web server repository 308 (step m3). The stylesheet is applied to the DXML document to generate an HTML document. This HTML document is returned by translate function 310 to get function 306 (step m6), which returns the HTML document to the client in response to the original request.

This HTML form provides an interface through an HTML form for use in prompting a user to input values and provides an example indicating how a DXML document may be interpreted or introspected for other uses. At client 300, this HTML form is displayed in Web browser 304 and is used to request parameters required by the DXML function. At Web browser 304, the HTML document is displayed with and includes fields and/or prompts for values, such as job type and the maximum salary. These fields and prompts may have been derived from introspecting the function definition and generated by the XSL stylesheet. In response to a user input to this HTML document, a request for a DXML document and a query string are sent to post function 312 (step m7). Post function 312 is used to access documents in Web server repository 308. Get function 306 and post function 312 are illustrated as separate functions for the purposes of clearly explaining the data flow used in a preferred embodiment of the present invention. These components may indeed be the same component depending on the implementation or specific request. This request is passed on to execute function 314 (step m8).

Execute function 314 is used to execute a DXML document based on the request and the query parameters received from post function 312. Post functions are used

to provide a mechanism to allow access to processes within an HTTP Web server from clients. In these examples, the get or post functions will communicate with the client using HTTP. HTTP requests received from a client are translated into the appropriate form for processing within server 302. Responses to these requests are received by the get or post functions and translated back into an HTTP format to transfer back to client 300. In response to receiving the request and the query parameters, execute function 314 retrieves the appropriate DXML document from the Web server repository 308 (step m9).

Additionally, an XML schema (if any) and DXML function definition also are retrieved from Web server repository 308 (step m10). Execute function 314 will read the DXML document. In response to reading the DXML document, requests to invoke functions are identified in the document markup language. Parameters from the query are substituted into the function invocation markup language. After that, the function is executed and the result is placed in the DXML document with markup language representing the result of the function. This replacement turns the DXML document into an XML document.

Next, the final form of the XML document is sent to translate function 316 (step m11). Translate function 316 obtains the appropriate XSL stylesheet from Web server repository 308 (step m12). This stylesheet is applied to the XML document to generate an HTML document. The HTML document is sent to post function 312 (step m13). This HTML document is transferred to client 300 (step m14). At client 300, the HTML document containing the results is displayed within Web browser 304. Additionally, the XML to HTML translation step is not

required if the browser is capable of processing XML with XSL stylesheets.

The DXML model of the present invention defines both document (or "free") functions and element member functions. Document functions are functions that do not belong to any element. These functions stand alone and have a context defined by the document in which they are invoked and their actual parameters. In these examples, document functions can be defined either in the XML document in which they are used, included in an XML document using an XML entity declaration, or defined in an XML schema, but not associated with any particular element in that schema. Therefore, document functions do not require an XML schema, but only require an XML document to be well formed.

The document functions, in these examples, correspond to free functions in C++ or functions of procedural languages like C, BASIC, JavaScript, and Ada. The document functions are also similar to class functions of C++, Java, and Smalltalk where the class corresponds to the document, which represents a shared context in which the functions can execute.

Element member functions are defined using an extension to XML schema. Elements in XML can have a name, a number of attributes, and associations with other elements including a containment association with the elements in their content model and more general associations through XLink and XPointer. DXML extends the XML schema to include behavior by allowing the schema author to include member functions in the element

definition. DXML extends XML so that elements can now correspond to UML, C++, or Java classes, or COM and CORBA objects. DXML functions encapsulate behavior with the XML element whose state transitions (i.e., the element's life-cycle) the function effects. This allows XML to be used to describe services on the Web and to use XML documents to script together the invocation of these services by document authors to solve some business problem and communicate the result to some other program or display it in a Web browser.

In these examples, functions are described using XML as much as possible in order to make them easy to introspect, simple to use by XML document authors, and language neutral. Only the function body is language specific.

IV. Function Definition

Turning next to Figure 4, an overall schematic of DXML and a diagram of a function definition are depicted in accordance with a preferred embodiment of the present invention. DXML consists of the combination of standard XML and common scripting languages, plus extensions to XML schema to include function definitions to form a component model for the Web. Function definition 400 includes a function name 402, a function description 404, formal parameters 406, type of return value 408, and function body 410. Formal parameters 406 includes, in these examples, the names of the parameters, the types,

the descriptions, and the default values. Type of return value 408, in these examples, may be either an XML element or character data (CDATA). Function names may be scoped by an XML namespace, while function parameter names are scoped by the function with which the names are associated. Function body 410 may include script instructions if the function is implemented using a scripting language. Alternatively, function body 410 may be empty if the function is implemented elsewhere. In this case, the function may be a document function in which case its name is sufficient to identify its implementation to a server. In other cases, an href attribute of the execute element, such as that used in HTML, may be used to identify a remote object that implements the function.

With reference next to Figure 5, a diagram illustrating a document function is depicted in accordance with a preferred embodiment of the present invention. In this example, document function 500 is entitled "overpaidEmployees". This document function corresponds to DXML function XML document in Figure 3. Document 500 contains an example of a document function that could appear in any XML document or an XML schema. Document function 500, in this example, has six formal parameters in section 502 including four that are required for database access. The return value in section 504 is an XML element with the tag name "overpaidEmployees". The function body in section 506 is written using dynamic structured query language (SQL) as

indicated by the language attribute of the body element.

If the body contained a large number of XML characters that must be escaped, it may be convenient to enclose the complete function body in a CDATA section. If the function is implemented as an EJB, CORBA, or DCOM object, then the language attribute would have the value "native", and the body would be empty as the implementation is done elsewhere. The meaning of the formal parameter types and their permissible types are defined by the function body language but described using XML. In this example, the parameter types are taken from SQL. The function execution engine has the responsibility to determine if the types are valid for the function body language.

In this manner, DXML can support many other common scripting languages, such as, for example, JavaScript, VBScript, Java, REXX, and Perl. Element member functions are similarly defined in an XML schema. The difference between element member functions and document functions is that by associating the function with an element, the context of the function includes its containing element, including its attributes and content model, and the invocation of the function is required to appear as a child of the element in which the function is defined. Since DXML uses XML to define functions, DXML is extensible. Implementations may include additional meta-data as attribute or content model extensions to DXML. Applications that are not aware of these extensions would just ignore the additional elements, as

is the convention in XML.

The invocation of functions is described in the following section. All functions are invoked in DXML documents by using XML, not the language used to implement the function. The XML document author never has to know anything about the implementation of the function, the language it was written in, or whether its written in a scripting language or implemented as a distributed service. When a DXML document is accessed, the function invocations in the document are executed and replaced with their result. All functions return either an XML element or CDATA. Functions that return XML elements must be invoked in the context of a parent element that has the returned XML element as a child in its content model in order to produce a valid XML document as the result. If the DXML document has no document type definition (DTD) or XML schema, then the returned XML element may appear as the child of any XML element, as the document must only be well formed. If the function returns CDATA, then the data may appear in any XML element that allows parsed character data (PCDATA) in the content model of the element. PCDATA can be the content of the element in XML. Alternatively, the data may be used to provide the value of an attribute.

The actual XML syntax for invoking DXML functions should be consistent with mechanisms for invoking remote procedure calls from client to server applications using XML to marshal the method, its parameters, and the return

value. The invocation used in the examples is not meant to specify a new syntax for distributed function invocation, but rather an example of how it could be done in the context of dynamic documents. The exact format of the function invocation syntax, in these examples, corresponds as much as possible to existing XML remote procedure call marshaling languages like SOAP, WebBroker, or Web Interface Description Language (WIDL). In other words, the function invocation is in XML and includes at least (1) the name of the function to execute (including any necessary namespace information); (2) a reference to an object implementing the function if the function is implemented as a distributed service; and (3) values for the actual parameters.

V. Function Invocation

With reference now to Figure 6, a diagram illustrating code for invoking a function is depicted in accordance with a preferred embodiment of the present invention. Function invocation 600 invokes function 500 in Figure 5. This example invokes the function 500 in the context of a practice in a department. DXML allows the query parameters of the request universal resource identifier (URI) to be used as part of the context of the DXML method. In the example, the value of the job parameter is provided by the job parameter of the request URI. For example,
`http://someDomain/companyStatistics?job=MANAGER&salary=10`

0000. The salary parameter value is a constant in this case. On execution of the DXML document, the execute element in section 602 is replaced by the results of the function, in this case an "overpaidEmployees" element containing information about managers who make over \$100,000.

Native functions are supported by using an execute element having, if necessary, an additional href attribute whose value is the URI of an RMI, CORBA, or DCOM object, and invoking the method on the native object.

Turning next to **Figure 7**, an example of code for executing native functions through an execute element is depicted in accordance with a preferred embodiment of the present invention. Execute element 700, in this example, provides for invocation of native functions. In this example, `iiop://someDomain/aCompany` is the URI referring to an instance of a class that has the method `overpaidEmployees`. This type of invocation mechanism makes invoking scripting functions and functions on other objects identical other than where the function is implemented. Document authors can now define simple functions of their own using SQL, JavaScript, Java, or any other scripting language supported by a DXML enabled Web server. Additionally, document authors also can access functions, such as those of Enterprise JavaBeans and COM objects. This access may be accomplished using exactly the same simple mechanisms. The function interface of the present invention is still defined as a

DXML function. The DXML function is invoked using XML. The only differences are that the body is empty, has language="native", and the document author provides the name of an appropriate service provider.

DXML functions that return CDATA also may be used to provide values for attributes. With reference next to **Figure 8**, an example of syntax for returning CDATA is depicted in accordance with a preferred embodiment of the present invention.

A number of ways exist in which DXML functions may be executed. For example, these DXML functions may be executed on the client using a Web browser plug-in or applet, on the server using CGI or a servlet or any other Web server extension mechanism, or invoked through XML document object model (DOM) in an application program. A servlet is a small application written in Java that runs on an Internet or intranet HTTP Web server.

A client Web browser may be extended using a plug-in or Java applet to execute a DXML document reducing the load in the server. The semantics of execution are the same regardless of the location at which the DXML document is executed. The difference in the location of DXML document execution is in performance. If the processing in the DXML functions is simple scripting, or access to distributed services, the DXML document can be executed on the client with no loss in performance. However, if the DXML functions access data that is already present on the server, it may be better to execute the DXML documents closer to the data to improve

performance. An "at" attribute of the executed element can be used by the document author to specify where the function should be executed. The default is to execute on the server.

DXML documents can be executed on the server using any of the mechanisms for extending Web browsers including CGI programs, servlets, or Web server plug-ins. A client program may also directly execute DXML functions. DXML extends the XML DOM interface by including document and element methods that return DOM for XML elements or CDATA representing the function return value. The client application can then execute these methods to obtain their results. This mechanism allows programmers to reuse the business logic contained in DXML documents written in various scripting languages within their programs.

When a DXML function executes, the function executes in a context that provides additional information to the function. The function context includes: (1) the document query parameters, its document context; (2) the parent element of the execute element, its dynamic context; and (3) the element this function is a member of its lexical context. The values of the query parameters are available in the DXML function execution language using the convention of preceding the query parameter name with a colon as used by host variables in SQL. If the function is implemented as a script, then the function has access to the DOM of the parent element in which the function is executed. If the function is an

element member function, the function has an additional context including the DOM for the containing element, the element's attributes, and content model. In this case, the DXML function is used to compute dynamic content for the containing XML element, and any invocation of this function appears within an instance of the containing XML element.

Another way to use DXML is to add a plug-in to XML/XSL browsers that is used as an XML "Virtual Machine" (VM) to execute dynamic XML documents on the client. This mechanism supports live, active documents as a snapshot of the persistent state of a collection of collaborating elements. Accessing the document "wakes up" the document by invoking the contained DXML function invocations to restart the collaborations and possibly changing the state of the document.

With reference now to **Figure 9**, a flowchart illustrating processes used in an execute function, such as execute function 314 in **Figure 3**, is depicted in accordance with a preferred embodiment of the present invention. The process begins by receiving a query (step 900). A DXML document is obtained from a source and read based on the query (step 902). The source may be, for example, a Web server repository containing documents. The functions in the DXML document are found (step 904). Parameters from a query are substituted into the functions (step 906). The functions containing the substituted parameters are executed (step 908). The results from the execution of the functions replace the

invocations of the functions in the DXML document (step 910). The DXML document becomes an XML document after execution of the function. An XML document is returned (step 912) with the process terminating thereafter.

With reference now to **Figure 10**, a diagram illustrating data flow between various components is depicted in accordance with a preferred embodiment of the present invention. In this example, user agent 1000 initiates a request for service (step p1). User agent 1000 is typically a process located on client 1002. The request formulated by user agent 1000 represents a desired service provided by a Web server. User agent 1000 may take different forms, such as, for example, a user or another application.

This request is processed by client 1002 and typically sent to server 1004 through communications channel 1006 (step p2). Communications channel 1006 may take various forms, such as the media found in network 102 in **Figure 1**. The request is sent over communications channel 1006 using a protocol understood by both client 1002 and server 1004. Server 1004 may use backend services 1008 to fully respond to the request (step p3). Backend services 1008 will return a result to server 1004 (step p4). Backend services 1008, in this example, may include a legacy process or system. Access to this legacy system may be used to satisfy part or all of the request depending on the implementation. This legacy system may be located on server 1004 or on another server. The request and the business logic on server

1004 is used to specify how such a delegation may occur in satisfying or responding to the request.

Server 1004 will create a response and transmit it back to client 1002 using communications channel 1006 (step p5). Client 1002 will then process the request for user agent 1000 (step p6). This processing may include displaying the data returned to the user. This display may be as simple as interpreting HTML statements or as complex as performing calculations, sorting operations, or other manipulations of the data. Alternatively, the data returned in the response may be used by another application located on client 1002.

For example, with a simple Web application, a user agent 1000 in the form of a user enters a URL for an HTML page using client 1002, which is a Web browser. The Web browser uses HTTP to communicate the request to server 1004, which is a HTTP Web server over TCP/IP. The request is processed by an HTTP Web server, which translates the URL to a pathname of a file on the server. No additional processing using backend services 1008 is required. The Web server sends the response back to the client again using HTTP. The requesting Web browser displays the HTML page to the user.

For a slightly more complex example, the user, using a Web browser, provides query parameters using an HTML form that is sent to the Web server using an HTTP POST over an SSL connection. The Web server processes the POST request by invoking a CGI script, which connects to backend services 1008 in the form of a backend database

server to execute an SQL query. The CGI script receives results from the backend database server and formulates the results in an HTML table, which is sent back to the client through the Web server, again using HTTP over SSL where it is rendered by the requesting Web browser.

In a more complex example, the user, using a Web browser running on a hand-held PDA, provides a query through an HTML form. This query is sent as a request over Internet Inter-Object Request Broker Protocol (IIOP) to an EJB enabled Web server. The EJB on the Web server represents a distributed business object that provides the requested service by invoking a command bean that communicates to a Customer Information Control System (CICS) application running on an multiple virtual storage (MVS) system using Message Queue Series (MQSeries). MQSeries is messaging middleware from International Business Machines Corporation (IBM) that allows programs to communicate with each other across all IBM platforms, Windows, VMS, and a variety of UNIX platforms. This middleware provides a common interface, such as an API, that programs are written to. The command bean returns the result as an XML document, which is sent back to the Java applet running in the client Web browser. The client Web browser uses an XSL stylesheet, XML parser, and XSL processor to render the document in a manner appropriate for the client device.

DXML introduces the opportunity for additional patterns that both enhance the power of Web applications while simplifying their implementation.

User preferences cover a broad spectrum of subject areas including, for example, device display characteristics, domain interest, user interface (UI) look-and-feel, and level of detail. Many of these preferences can be simply handled by different XSL style sheets when accessing XML documents. In other situations it is necessary to calculate the contents of the requested XML document based on user preferences or to calculate the right stylesheet to use. A DXML document on the server can be used to perform these calculations. A typical pattern might be a user, using a Web browser running on a hand-held PDA submitting a request using an HTML form by sending the request over HTTP to a DXML enabled Web server. The Web server accesses the requested DXML document and executes the document, which uses information about the user agent to access user preferences to calculate the response XML document contents. The response document is sent back to the client Web browser, which uses an XSL stylesheet, XML parser, and XSL processor to render the document in the manner specified by the user preferences.

Another common use of DXML is to access or query a database. The user, using a Web browser, submits a request using an HTML form by sending the request over HTTP to a DXML enabled Web server. The Web server accesses the requested DXML document and executes the document, which executes DXML functions that access a backend database system to obtain the requested data. The response document is sent back to the client Web

browser, which uses an XSL stylesheet, XML parser, and XSL processor or Java applet to render the document in some appropriate manner.

As behavior becomes more prevalent on the Web, Web applications may be dynamically bound to published services obtained by an intelligent search engine. DXML can be used to invoke these services and capture the results into a composite XML document. The search engine would locate DXML documents containing functions that provide advertised services. The function definition would be extended to provide additional information like keywords that could be used to facilitate the search. Once the client finds a collection of desired services, a DXML document can be used to invoke the services and collect the results into a target XML or DXML document. This document can either be processed by an XSL stylesheet to transform the document into the desired form, or can be recursively executed as a DXML document for further refinement.

VI. Examples for DXML Implementation

With reference now to Figures 11-14B, code illustrating an example DXML implementation is depicted in accordance with a preferred embodiment of the present invention. These examples involve the use of a document function, such as the document function found in document 500 in Figure 5. This document function is a dynamic XML (DXML or .dxml) document that contains a document

function definition for querying an SQL database looking for employees that are overpaid with respect to their job responsibilities. The function is defined in an XML function element. The function definition includes its formal parameters, their types, descriptions, and default values. The function body in this case is written in SQL.

With reference now to Figure 11, an example XML document used to display an HTML form is depicted in accordance with a preferred embodiment of the present invention. The XML document in Figure 11 includes the function defined in Figure 5 as a convenience for function reuse in the example. XML document 1100 includes XML document 500 in Figure 5 as an external entity, and specifies an XSL stylesheet 1200 in Figures 12A and 12B to render the function as an HTML form for requesting values for the formal parameters. The reason for these three documents is to allow the function to be specified in a separate document so that it can be easily reused and to separate what the function is from how it is viewed. This document also is an example of a technique for rendering the same document with different XSL style sheets.

With reference now to Figures 12A and 12B, an example of an XSL stylesheet used for rendering the XML document in Figure 11 is depicted in accordance with a preferred embodiment of the present invention. XSL stylesheet 1200 is a more detailed example of an XSL stylesheet used in Figure 3. XSL stylesheet 1200 is the

XSL stylesheet used for rendering XML document 1100, overpaidEmployees.xml, to obtain values for the input parameters. In this example, XSL stylesheet 1200 does not display the url, driver, user, and password parameters and just uses their default values. Other style sheets may be used to display the input parameters for the function in some other way, or the function could be included in an XML document with other functions, including functions that validate the inputs.

Turning next to **Figures 13A and 13B**, an illustration of an XML document containing an execute element is depicted in accordance with a preferred embodiment of the present invention. XML document 1300 is an example of a dynamic XML document used in Figure 3. In this example, XML document 1300 contains company information including departments and practices. XML document 1300 contains an execute element that executes the overpaidEmployees function for a practice. An XML execute element (defined by DXML) is embedded in the XML document in section 1302.

When the document is processed, this element will be replaced by its results which will be an overpaidEmployees element containing the results of the query translated to XML. All DXML functions return XML elements, although the elements could be as simple as PCDATA (a simple character string).

With reference now to **Figures 14A and 14B**, a diagram illustrating an XSL stylesheet for displaying the XML document in **Figures 13A and 13B** is depicted in accordance with a preferred embodiment of the present invention.

XSL stylesheet 1400 is a more detailed example of the XSL stylesheet used in **Figure 3**. XSL stylesheet 1400 is an XSL stylesheet for displaying XML document 1300 in **Figures 13A and 13B**, companyStatistics.dxml, after XML document 1300 has been processed. XSL stylesheet 1400 displays a simple report of the companies, their names and descriptions, and for one practice, gives a table of overpaid employees. In these examples, servlets may be used to provide various functions. For example, a servlet for translating XML to HTML using an XSL stylesheet is employed to render the XML as HTML for Web browsers that are not capable of handling XML and XSL directly.

Further, a DXML servlet processes dynamic XML documents by removing the function elements, and replacing the execute elements with their result XML element or PCDATA. The DXML servlet may be used to support many scripting languages. The servlet outputs the original document after replacing execute elements with their results. In the DXML servlet, the value of an actual parameter in an execute element may contain ":variableName" in order to indicate that the value should be obtained from the similarly named parameter from the document query string.

Thus, the present invention provides a method, apparatus, and computer implemented instructions for reusable components for use in a document. The present invention includes a document centric architecture in which logic or functions used to build applications or

dynamic documents are provided in a manner in which the logic or functions are reusable in many different documents. Through the use of DXML in the depicted examples, Web application complexity is reduced by separating schema, logic development, data, and rendering instructions. Functions are implemented using scripting languages or native languages in which XML is used to invoke the functions. The invocation of these functions is independent of the mechanism used to implement the function.

As shown above, the Web application development of the present application allows for adding behavior to markup languages. The mechanism of the present invention avoids the use of template languages, as in ASP and JSP, by providing active documents based on a component model as described above. In this manner, a document author does not have to switch into a programming domain to invoke functions. With DXML, the document author only uses XML. The interfaces for the methods and the invocation of the methods are both defined using XML. Only the function body, or its implementation, is language dependent. Further, the DXML functions of the present inventions can be defined in an XML schema, rather than being included in the document itself. Further, implicit invocation of methods, as in IE5 behaviors, is avoided through the use of direct function invocation. These advantages are provided through the formal component model described above for DXML in which DXML defines and executes functions using XML. This

mechanism is in contrast to having a scripting language execute the function directly. In this manner, function declaration and invocation are language independent.

Functions can be methods of XML elements that contributed to their scope. Using a formal component model enhances the use of scripting languages, simplifies document development, and keeps a clear separation between the document schema author, document author, and document layout roles. The model of the present invention unifies existing distributed object component models like CORBA, EJB, and DCOM, and makes the services of these distributed component models available to XML document authors through XML.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in a form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media such as digital and analog communications links, wired or wireless communications links using transmission forms such as, for example, radio frequency and light wave transmissions. The

computer readable media may take the form coded formats that are decoded for actual use in a particular data processing system.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. For example, although the depicted examples are illustrated using XML, the present invention may be implemented using other markup languages. As another example, the invocation of functions may be accomplished through the XML execute element. Another approach would be to use an XML document language like SOAP to execute the functions. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated. The particular document language used in the examples above was chosen because of its simplicity to explain the concepts of DXML. In addition, many other related technologies that might be employed in DXML like SOAP and XML Schemas.

WE CLAIM:

1. A method in a data processing system for creating a dynamic electronic document in a markup language, the method comprising the data processing system implemented steps of:

placing static content in the markup language into the dynamic electronic document;

placing a function invocation mechanism in the dynamic electronic document using the markup language;

in response to activation of the function invocation mechanism in the dynamic electronic document, invoking a function; and

replacing the function invocation mechanism in the dynamic electronic document with a result from the function.

2. The method of claim 1 further comprising:

defining the function using a function definition in which the function definition includes a function name, formal parameters, and a function body.

3. The method of claim 2, wherein the function is located in a second dynamic electronic document.

4. The method of claim 2, wherein the function body comprises a language attribute, and wherein the language attribute identifies a language in which the function is written.

5. The method of claim 4, wherein the language attribute is script and the function body further includes instructions in script.
6. The method of claim 4, wherein the language attribute is native and the function body is empty and wherein the function is implemented outside of the dynamic electronic document.
7. The method of claim 1, wherein the function invocation mechanism includes a reference to an object implementing the function.
8. The method of claim 1, wherein the markup language is extensible markup language.
9. The method of claim 1, wherein the function includes instructions written in script.
10. The method of claim 1, wherein the function is written in a language other than a scripting language embedded in the markup language.
11. The method of claim 1, wherein the function is implemented in one of a plug-in, an applet, a servlet, and an application.
12. An electronic document system for use in a data processing system, the electronic document system comprising:
 - a function; and

an electronic document, wherein the electronic document includes:

content in a markup language; and

a function invocation mechanism written in the markup language,

wherein, in response to an activation of the function invocation mechanism, the function invocation mechanism calls the function and a result from the function replaces the function invocation mechanism in the electronic document.

13. The electronic document system of claim 12, wherein the electronic document is located on a first data processing system and wherein the function is a remote function located on a second data processing system remote to the first data processing system.

14. The electronic document system of claim 12, wherein the electronic document is a first electronic document and further comprising:

a second electronic document written in the markup language, wherein the function is located in the electronic document.

15. The electronic document system of claim 12 further comprising:

a stylesheet used to present the electronic document.

16. A data processing system for creating a dynamic electronic document in a markup language, the data processing system comprising:

first placing means for placing static content in the markup language into the electronic document; and

second placing means for placing a function invocation mechanism in the electronic document using the markup language, wherein the function invocation

mechanism is configured to invoke a function and a result from the function replaces the function invocation mechanism in the electronic document.

17. The data processing system of claim 16 further comprising:

defining means for defining the function in which the function includes a function name, formal parameters, and a function body.

18. The data processing system of claim 17, wherein the function is located in a second dynamic electronic document.

19. The data processing system of claim 17, wherein the function body comprises a language attribute, and wherein the language attribute identifies a language in which the function is written.

20. The data processing system of claim 19, wherein the language attribute is script and the function body further includes instructions in script.

21. The data processing system of claim 19, wherein the language attribute is native and the function body is empty and wherein the function is implemented outside of the dynamic electronic document.

22. The data processing system of claim 16, wherein the function invocation mechanism includes a reference to an object implementing the function.

23. The data processing system of claim 16, wherein the markup language is extensible markup language.
24. The data processing system of claim 16, wherein the function includes instructions written in script.
25. The data processing system of claim 16, wherein the function is written in a language other than the markup language.
26. The data processing system of claim 16, wherein the function is implemented in one of a plug-in, an applet, a servlet, and an application.
27. A computer program product in a computer readable medium for creating an electronic document in a markup language, the computer program product comprising:
- first instructions for placing static content in the markup language into the electronic document; and
 - second instructions for placing a function invocation mechanism in the electronic document using the markup language, wherein the function invocation mechanism is configured to invoke a function and a result of the function replaces the function invocation mechanism in the electronic document.