XVM: A Bridge between XML Data and Its Behavior

Quanzhong Li[*]
Dept. of Computer Science
University of Arizona
Tucson, AZ 85721
lqz@cs.arizona.edu

Michelle Y Kim
IBM T.J. Watson
Research Center
mykim@us.ibm.com

Edward So
IBM T.J. Watson
Research Center
edwardso@us.ibm.com

Steve Wood
IBM T.J. Watson
Research Center
woodsp@us.ibm.com

ABSTRACT

XML has become one of the core technologies for contemporary business applications, especially web-based applications. To facilitate processing of diverse XML data, we propose an extensible, integrated XML processing architecture, the XML Virtual Machine (XVM), which connects XML data with their behaviors. At the same time, the XVM is also a framework for developing and deploying XML-based applications. Using component-based techniques, the XVM supports arbitrary granularity and provides a high degree of modularity and reusability. XVM components are dynamically loaded and composed during XML data processing. Using the XVM, both client-side and server-side XML applications can be developed and deployed in an integrated way. We also present an XML application container built on top of the XVM along with several sample applications to demonstrate the applicability of the XVM framework.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features – *frameworks.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - *Graphical User Interface, prototyping.* I.7.m [**Document and Text Processing**]: Miscellaneous.

General Terms

Documentation, Design, Languages.

Keywords

Components, Web applications, XML, XML applications, XML processing, XVM

1. INTRODUCTION

The World Wide Web has been evolving from serving simple document requests to additionally supporting complex enterprise-level business applications. Today, web-based applications play a critical role in electronic commerce and in corporate information systems. At the same time, the Extensible Markup Language (XML) [3] is accepted as the standard for data representation and exchange, and it has been widely used in web-based applications. This creates a great demand for facilities that can incorporate XML support into development and deployment of web-based applications.

Much research work targeting web-based applications has been proposed in the literature. In the software engineering area, software development principles and guidelines [1][5][8][9] have been suggested. Despite the efforts that have been focused on the web, the difficulties and inefficiencies in development and maintenance of web-based applications are still prominent.

To give the client side the ability to process XML documents several XML based browsers are available [12][13][14][15]. However, they are not designed to support the processing of general XML data, nor do they have a flexible open architecture for incorporation into XML applications. As a result, many web applications are not built with a conceptually clean integrating model. Although many XML application development tools have been developed [16][18], they provide only the basic abilities for XML data processing, such as data loading, parsing, etc. There is a lack of a general model to facilitate applications that wish to implement business logic related with the XML data.

In this paper, we propose an extensible, integrated XML processing architecture, the XML Virtual Machine (XVM). The XVM is a component-based XML-centric architecture with dynamic binding, high modularity and reusability. It provides a framework for XML applications to implement application logic in a component-based approach. Components are naturally and dynamically composed according to the processed XML data. Application designers and developers have the freedom to choose the granularity of components. The XVM facilitates the development and deployment for both client-side and server-side XML applications. The contributions of this paper are:

- An integrated XML document processing framework is proposed.
- The XVM provides high-degree of extensibility and reusability by using component-based techniques.
- The component to XML element association connects static XML data with its behaviors.
- Deployment of XML applications is made easy through registry service and dynamic binding mechanisms.

The rest of the paper is organized as follows. Section 2 presents the motivations of this paper. The XVM framework is introduced in Section 3. Following this in Section 4, we describe the XVM prototype and sample applications. Section 5 discusses the related work. Finally, we summarize the contribution of this paper and give an outlook for future work in Section 6.

[*] Work done while the author was an intern at IBM T.J. Watson Research Center during summer 2003.

```
<employee>
  <name>Bill</name>
  <id>000001</id>
  <salary>
    <webservice name="PayrollService" />
  </salary>
</employee>
```

Figure 1: An employee element.

2. MOTIVATIONS

With the popularity of XML, it has become necessary for web-based applications to handle a variety of XML documents. To facilitate the development and the deployment of XML applications, an extensible, integrated XML processing model and platform is required. However, the emergence of various XML applications poses a great challenge to providing such an integrated processing model. XML documents provide data in different structures based on schemas and how the data is interpreted depends on the application.

For example, Figure 1 shows an employee data constructed according to a schema. In this example, the salary of the employee should be obtained from a web service. Upon processing such an employee element, different applications may handle it in different ways. For instance, one application may insert the information into a database, while another may display the information to the user. In other words, an XML application has the freedom to define the semantics of the data.

XML Schema [24] was approved as a W3C Recommendation in May 2001 and is now being widely used for structuring XML documents in Web based applications. The design of XML Schema incorporates object-oriented analysis and design principles. Object-oriented technologies have been proven to be an effective methodology to analysis and design applications. However, currently most Web applications are implemented based on low-level technologies directly. It is beneficial to provide a framework to process XML data in an object-oriented approach following the XML schema design. For example, in Figure 1 the `employee` element can be considered as an object and its "behavior" can be realized through the application development framework.

With the evolution of an application development, the schemas defined for the application may be changed in accordance with new requirements. The best way for applications to keep up with the evolution of schemas is to maximally reuse previously developed software. If a component-based model can be utilized for XML application development, then the required updates can be limited to a small set of components. Components can also be dynamically updated and new features can be added without difficulties. For example, let us assume the schema of the employee data in Figure 1 is changed for some reason. An element `phonenumber` is added as a sub-element of `employee`. In the model proposed in this paper, the changes to the application are limited to only two components, as we will see in Section 3.2.

Another motivation of the XVM is related to the processing of *compound XML documents* – documents contain vocabularies from a set of different namespaces. Using this feature, XML content from one specification can be used in other XML specifications without having to define new specifications each time. For example, SVG, MathML fragments, and XForms [24] can be embedded inside HTML documents. Obviously, it is inefficient and costly to develop separate processing applications for the variety of different types of compound documents that may ensue. A better approach is to use a component-based framework to construct the processing application reusing existing components that have been implemented in individual namespace applications. In this way, little or no extra work is needed to build these new applications. Section 4.3.2 will show how the reusability of existing components is supported in our architecture in a compound XML scenario.

In the web-based application framework, web browsers are used to display the content of markup languages, like HTML, on the client side. Since a web browser initiates web applications and supports user interactions, it can be viewed as an application container, and this emphasizes the importance of web browsers in web-based applications. With the advent and extensive use of XML, one limitation of current web browsers is becoming apparent. The commonly used Web browsers, such as Internet Explorer and Netscape, were originally designed to display HTML content, which makes it hard to extend them to support general XML applications. An XML application has the freedom to define its own vocabulary and specify the semantics of the vocabulary. However, current web browsers cannot cope with the semantics of arbitrary XML applications.

XML can also be used to facilitate the distribution of processing load from a web server to a web client. In order to do that, application specific XML data can be sent and processed on the client side. With the current web browsers, this feature is not fully explored. Although, the Extensible Stylesheet Language Transformation (XSLT) [10] can convert XML data into HTML or XHTML format for a web browser, XSLT cannot solve everything. That is because the target language of transformation is HTML or XHTML and the limitation of using HTML remains, as it is not suitable for specifying semantics for all applications. For example, the salary element in Figure 1 needs to access a remote service to get the actual data. No matter how you translate the data, you cannot express the remote service access semantic in HTML. Without knowing how to access the remote service, it is not possible for web browsers to display the data as desired.

Using client-side JavaScript enables more interactivity, and gives web applications a certain amount of freedom to implement dynamic semantics. However, JavaScript is not a panacea. JavaScript does not have a built-in component model, nor can it be reasonably expected to be suitable for implementing large and complex applications. Such applications normally require component-based facilities for extensibility and reusability.

Motivated by the above observations, we propose a component-based XML processing framework — XML Virtual Machine (XVM). In the XVM, the association between elements and components connects the XML data to its behavior. XML applications can implement their application logic in an extensible and reusable way.

3. XML VIRTUAL MACHINE (XVM)

In this section, we will show the proposed application framework and how the XVM can be used in web based applications. Then, we introduce the key ideas behind the XVM framework and present more details about the XVM architecture. In the rest of the

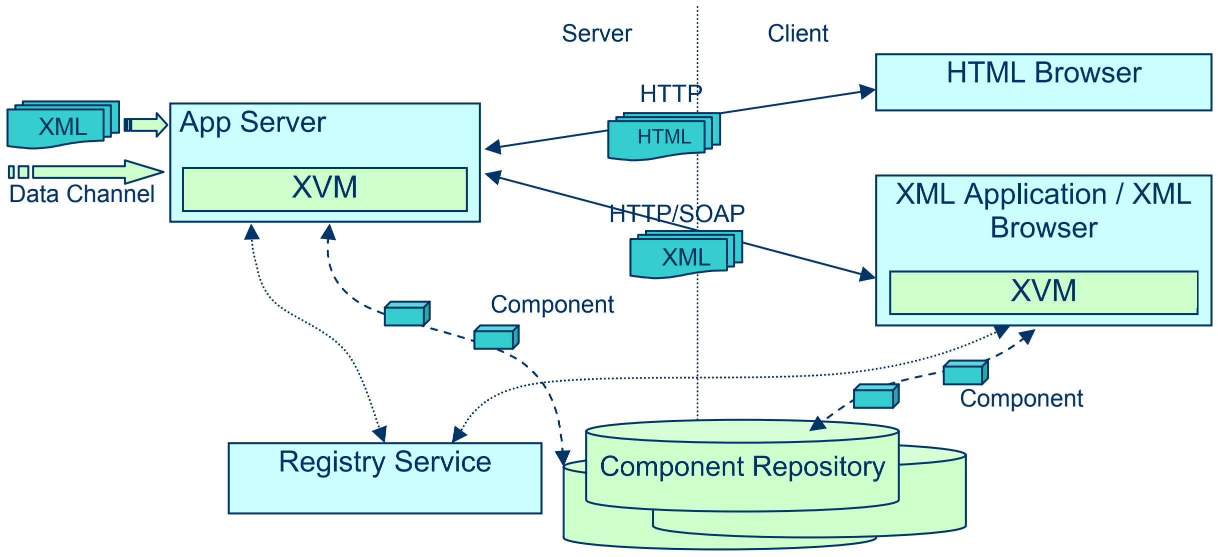


Figure 2: The XVM Application Framework.

paper, we will use the Document Object Model (DOM) interface to introduce the XVM. However, this framework can also be built up on Simple API for XML (SAX) interface only with some changes to the implementation.

3.1 The Application Framework

The XVM is a component-based technology. Instead of deploying a monolithic package that includes every feature, a component-based system can add new features or upgrade existing features using components [1]. Figure 2 depicts the framework of how the XVM can be used for web-based applications. On the server side, an application server built on the XVM can process XML documents and/or XML data from data channel connections. To support legacy HTML browsers, HTML formatted data is transferred to the client. For XML browsers or applications, built on the XVM, XML data can be directly sent to the client. XML browsers can realize the semantics of the XML data through the related XML application. Each XML application consists of a set of components, whose structure will be introduced in the following sections.

For both server- and client-side XVM, the application components can be downloaded dynamically from component repositories. The dynamic downloading feature makes it unnecessary to install everything of an application before running it. The XVM-based applications can adapt to computing environments with limited resources such as handheld devices. Another benefit is that an application has the freedom to choose what and when to download components even during runtime. Also, components are cacheable inside the XVM with the most frequently used components cached locally for better performance and rarely used ones being downloaded on demand. In most cases, components are downloaded only once during the first run. After that an application will run without additional cost and delay of downloading components.

Inside the XVM, a registry service is used to locate components, which will be detailed in Section 3.5. The whole XVM application architecture is not only a development framework; it is also a deployment framework. When an application needs to be upgraded, only the registry needs to be

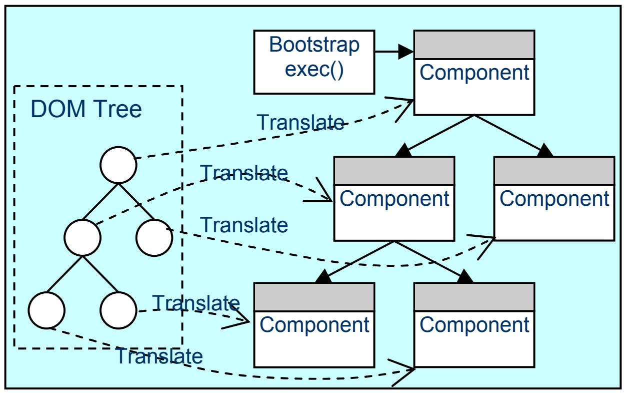


Figure 3: Mapping between Components and DOM Element Nodes.

changed to replace existing components or add new components. The updated/new components will be automatically downloaded for the application. In this way application maintenance is made much simpler.

3.2 Components and XML Element Association

In the XVM architecture, the key idea is a mapping between XML elements and software components, which associates XML elements with software components. The component is responsible for realizing the application logic related to the associated element and providing services to other components.

Components are building blocks for constructing XML applications. They are reusable software units that can be dynamically composed into larger components through the XVM component composition and mapping mechanism. The composition is based on XML document information being processed. The element to component mapping is mediated by the XVM registry service.

Figure 3 illustrates the mapping between components and element nodes in the document. When an XML document is loaded into the XVM, a DOM tree is built. Corresponding to the

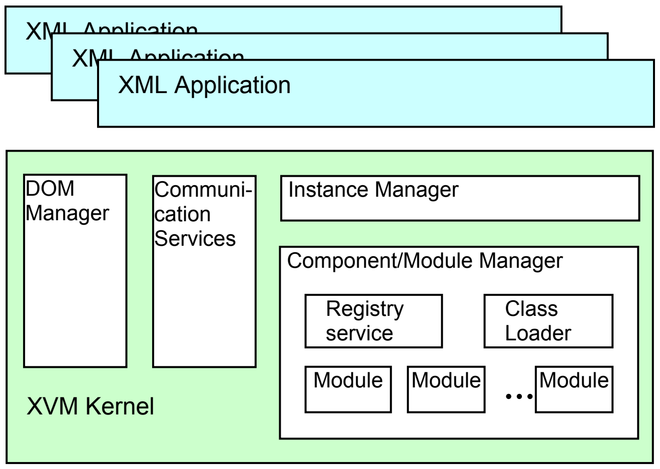


Figure 4: The XVM Kernel Illustration.

root element node of the DOM tree, the root component is first created and executed. During the execution of the root component, if it needs the services of sub-components corresponding to the child nodes in the DOM tree, these sub-components will be created. The actual translation from DOM nodes to components is performed by the XVM automatically through the registry service. In this way, a tree structure of components is dynamically composed. The execution of these components then realizes the semantics of the XML document. Every component implements one or more interfaces. Through the interfaces, the component can be invoked not only by the components in the same namespace but also the components in other namespaces. In this way, a compound document can be processed in the same framework without extra effort.

We shall use the employee data in Figure 1 as an example to explain the association. From the viewpoint outside of the employee element, the employee component is responsible for processing the data contained inside the employee element. Since `name` and `id` elements are trivial data, it is not necessary to assign components to them (although an application designer could choose to do so). The employee component can directly access the data of these child elements through the DOM interface.[1] As for the salary element, given that it needs web service access, we design a component to be associated with it. The task of the salary component is to use the employee id information, which is passed from the employee component, to obtain the salary information from the web service described within the salary element. When the employee component encounters the salary element, it asks the XVM to create a salary component and uses the interface provided by the salary component to get the salary information. With all the employee data in hand, the employee component can then process the information according to the semantics of the employee element.

We should emphasize that not every element is necessarily associated with a component. The designer has the freedom to choose the granularity of components. For example, one can use one component to process one type of XML data, e.g., the whole segment of SVG or MathML data. On the other hand, one can associate components to meaningful elements or functionally independent elements, e.g., the basic shape elements in SVG. In this way, compound XML documents can be processed by invoking correspondent components, without the need to write an application for each combination.

When a schema is changed, only the related components need to be updated. Back to the example in Section 2, if `phonenumber` is added to `employee`, a new component corresponding to `phonenumber` may or may not be needed (depending on the application). Among the existing components, only the component corresponding to its parent element (`employee`) needs to be updated. We shall see in Section 3.5, the component update is a simple job with the introduction of the component registry service.

3.3 XVM Kernel

As illustrated in Figure 4, the XVM kernel manages DOM information, components, and component instances. It also contains common libraries, e.g., communication services that are needed by XML applications. The DOM manager provides the implementation of the DOM Recommendations. The Instance Manager relies on the Component Manager to create instances of components. It tells the Component Manager the *component key*, which contains the XML namespace and tag name information, of a component. The Module Manager looks up the registry and finds the location of the corresponding component. If the component is not already downloaded and cached locally, the Component Manager retrieves the component, and uses the component class loader to load the component classes. Finally, the Instance Manager uses the returned component class to create instances for applications.

The DOM manager loads XML data and builds an XML DOM tree. In this way, components have the ability to dynamically change the document structure. This is more useful in client-side applications, where dynamic data manipulations according to user activities are necessary. For server-side applications, where performance is more important, SAX interface can be used. This can be viewed as a progressive application loading process. A component can be loaded and started to run without the need to wait for the rest data to be fully loaded.

3.4 Component Structure

In the XVM a component implements one or more interfaces. An interface is a group of services (operations or methods) that a component exposes. Different interfaces correspond to different aspects of an application. Interfaces can be extended and combined forming an inheritance structure as in Figure 5 above. Several basic interfaces have been predefined in XVM. For example, the *Viewer* interface provides the simple rendering interface, and the *Doer* interface provides start and stop methods for controlling the execution of a component. Component developers can extend or combine existing interfaces and they can also add their own interfaces.

As illustrated in Figure 5, inside a component, there is a proxy object and one or more module objects. A proxy is the portal of a component, through which other components can access the services. One proxy may support multiple interfaces, which are implemented by modules. Proxies provide a dynamic binding mechanism between interfaces and modules. Each module

[1] Trivial XML data can be converted to "simple data" components, thereby completing XML-to-component transformation. But we will not consider it here for performance reasons.

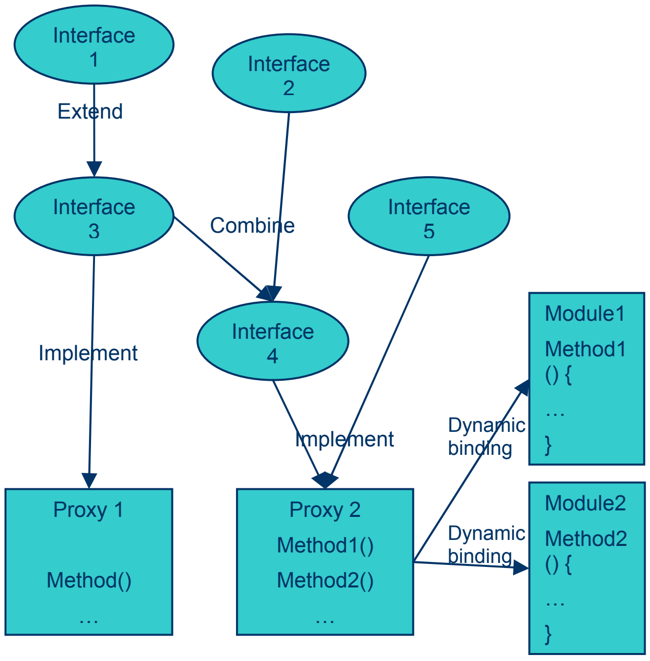


Figure 5: Component Structure Illustration

implements one or more interfaces of the proxy. During a method invocation, if the implementation module object is not loaded, the proxy asks the XVM kernel to load the module into memory. Then, the proxy passes the invocation to the module object.

The introduction of proxy and modules in the XVM gives more freedom for component developers to choose the suitable granularity of modules. With dynamic binding, components can be dynamically downloaded, and the ongoing application development and maintenance can be facilitated without affecting the running system. The initial deployment of an application also becomes simple. The only work needed is to correctly input the information in the XVM registry service. Components of the application will be downloaded on demand and frequently used ones will be cached for better performance.

3.5 Component Key and Registry

In XVM, a component is dynamically constructed according to the XML DOM information. Each component is associated with an element in the DOM tree. This association is done through a registry service, which maps a component key to the location of the associated proxy object and program modules of the component. A component key consists of a namespace and a tag name, which uniquely identifies the component and hence the proxy of the component. In order to distinguish different modules of a component, a module index field is added to the key. Please refer to Figure 6 for the structure of a component key.

An application may have different ways to process the same XML document in different processing contexts. For example, with the employee data in Figure 1, an application may insert the data into a database, or it may display the data to a user. For different client-side consumer devices different components suitable for different devices may be needed. In order to support different processing modes, we added the "processing mode" field in the component key. During a registry lookup, the application provides the value of the processing mode field to locate the

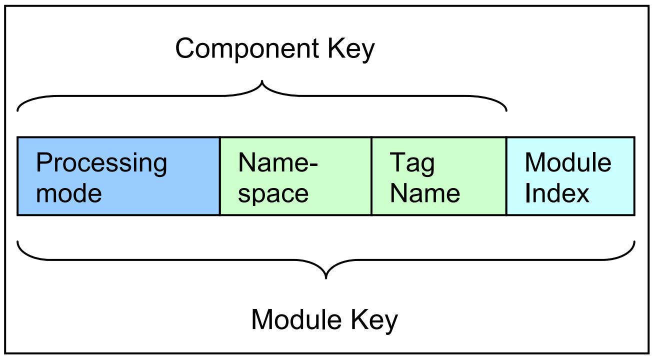


Figure 6: Component Key Structure

appropriate component. If there is no processing mode registered or specified, the default processing mode will be used.

For XML applications without a namespace, we provide an ad hoc mapping mechanism. An element can specify the location of the associated component through an attribute "`component`". For example, the `foo` element

```
<foo component = "http://example.com/foo"/>
```

specifies `"http://example.com/foo"` as the location of the component.

Before an XML application can be run in the XVM, the (key, location) pairs of the components should be registered with the registry service. Once that is done, the Component Manager, using the registry service, can then access and load the application's registered components as needed. The registry service enables the XVM to deploy or upgrade an application by merely registering the keys for new components. The actual component retrieval and loading are automatically done by XVM.

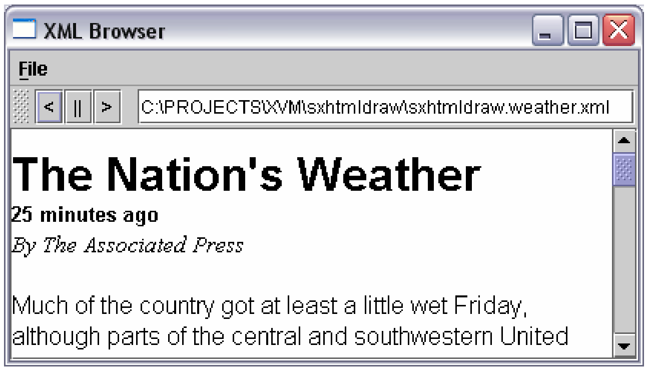
3.6 Why Is It Called XVM?

Readers may be wondering why this architecture is called XML Virtual Machine. We shall make it clear that we are not introducing a new XML programming language. Our goal is to provide a platform for design and construction of XML-based applications to process different XML documents. If we consider the XML data as "machine instructions" (a special type of tree structured instructions) and components as "instruction interpreters", then we can easily understand why it is called XML Virtual Machine. The XVM dynamically loads "interpreters" and uses them to "interpret" XML "instructions". An XML document is like an executable "program" that can be executed by the XVM to realize the semantics of an application.

4. XVM IMPLEMENTATION

We have built a prototype implementation of an XVM using the Java programming language. The dynamic class loading mechanism of Java makes it easy to implement a dynamic component-based programming architecture. We also used the Apache Xerces2 DOM implementation [16] in the DOM Manager.

Two different XML applications could use the same Java class name in their components for different purposes. When they run at the same time in the XVM, a name clash may occur. To solve this problem, we used different class loaders for different XML namespaces. Thus components in different namespaces are loaded using different class loaders. Since classes loaded by different class loaders are considered different classes even if they



The Nation's Weather

25 minutes ago

By The Associated Press

Much of the country got at least a little wet Friday, although parts of the central and southwestern United

Figure 7: Sample Snapshot of SXHTML Application.

have the same class name, the namespace class loader technique solved any potential class name clash problem. There is also one global class hash table shared by all namespace class loaders to avoid loading the same class multiple times. Each entry in the table consists of three fields, namespace, class name, and the loaded class object. Before a namespace class loader tries to load a class, it checks this global class table first to see if such a class is loaded already.

In this prototype implementation, the registry service is simplified. The mapping from a component key to the component proxy and modules is specified in a configuration file. During runtime, the XVM kernel consults the configuration file to retrieve component information.

4.1 An XML Application Container

In order to facilitate the development of XML applications with GUI, we developed a GUI application container called *XML Browser*. The XML Browser is a client runtime platform for XVM applications. It provides a graphical user interface for loading and executing XML files.

During startup, the XML Browser creates an XVM instance and initializes it. The XVM instance will be used to run XML applications. The XML Browser has a file menu, in which there is a menu item, "open", to let users choose the XML file to be opened via a file open dialog. Users can also directly input the file name in a text edit field.

When an XML file is loaded into the XML Browser, the browser invokes the `exec` method of XVM to start the application. A graphics container is also supplied to the application if the application is *renderable*. The renderable interface means the application can display GUI content in the graphics container. With the container in hand, the application may create its own content and add it to the container. If an application is not renderable, it does not output any content in the container. In this case, the browser will show nothing in the GUI. However, the application may still do some computation and print out information in the command line terminal.

The XML browser also remembers the history of visited XML files. Users can use the backward and forward (either by using menu items or toolbar buttons) actions to navigate through the history. We shall note that, even without XML browser, an XVM application can also be run by a default XVM runtime. The purpose of the XML browser is to provide additional functionalities to facilitate the implementation of GUI-oriented XVM applications.

4.2 Sample Applications

Based on the prototype implementation of the XVM, we implemented several sample XML applications to demonstrate the broad applicability of the XVM framework.

4.2.1 SXHTML

For this application, we implemented a simple markup language called SXHTML. It resembles HTML, but supports only a small set of tags: "sxhtml", "font", "p", "b", and "i". We shall note that for this simple application there could be other ways to implement besides the XVM. For example, we can simply transform SXHTML to HTML and use a standard web browser to display the content. However, the purpose of this simple sample is to demonstrate the potential ability of the XVM, especially the extensibility of the XVM. That is, new elements can be added easily.

In the implementation, we used two methods to display the content of an SXHTML file. In the first method, each element tries to display its content using the Java Swing container manager, with text being displayed as labels. An element container could be contained in another element container if the corresponding element of the former is a sub-element of the corresponding element of the latter. This method provides an example to build applications using Java Swing containers to display contents. The other method displays elements and text inside a single Java Swing container. In both methods, component contexts are used to pass information from parent element components to child element components. A sample snapshot is shown in Figure 7.

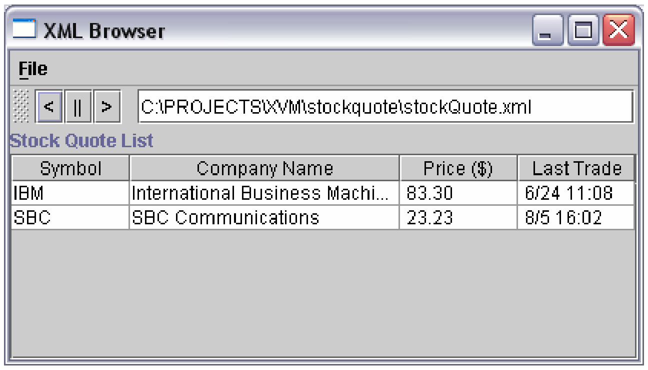
4.2.2 Stock Quote

There are two elements, stockQuote and stock, in this sample application. The stockQuote element module creates a table object, and adds the table to the container to display. The table content is obtained from each stock element. The stock component obtains the stock information either from a local data file or from a web service according to the information in the attributes of the element.

In this application, the stock component only performs computation and provides information to its parent component. The `stockquote` element component collects the stock information from its child components and displays the results in a table. This demonstrates that, in an application using graphics, some elements may only provide computation and information; they may not participate in rendering. Figure 9 shows an execution snapshot of the sample XML file in Figure 8.

```
<?xml version="1.0"?>
<sq:stockQuote xmlns:sq="urn:stockquote">
  <sq:stock symbol="IBM" source="stockquote
                          /stockPrices.txt"/>
  <sq:stock symbol="SBC" source = "http://
              stockquote.com/StockQuoteWAR
              /servlet/rpcrouter">
    <sq:stockQuoteWebService/>
  </sq:stock>
</sq:stockQuote>
```

Figure 8: Sample File for Stock Quote Application.



Stock Quote List

Symbol	Company Name	Price ($)	Last Trade
IBM	International Business Machi...	83.30	6/24 11:08
SBC	SBC Communications	23.23	8/5 16:02

Figure 9: Snapshot of Stock Quote Application.

4.2.3 *Compound XML Application*

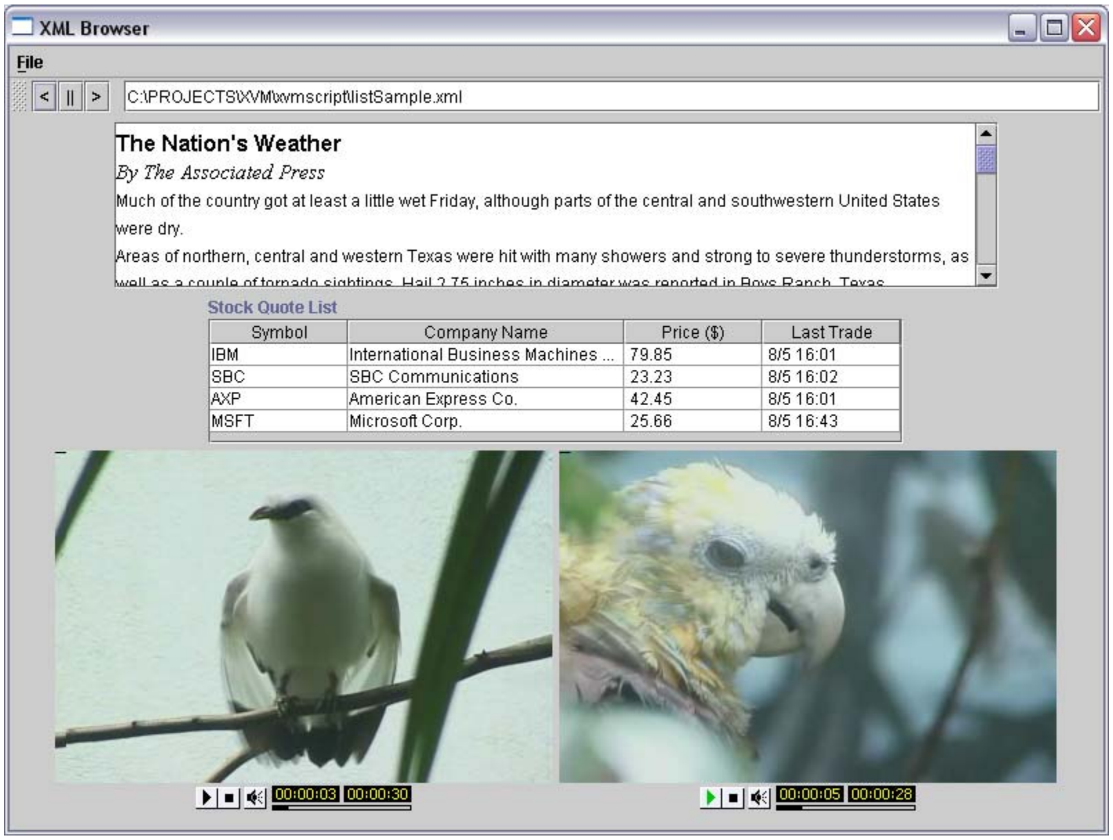
One important benefit of using XVM and the component-based methodology is that the developed components can be easily reused. We show this by using a compound XML document, which contains elements from four different namespaces. Two of the namespaces correspond to the two previous XML applications, whose components are reused. The third namespace defines a new element with a corresponding component for a Java applet-like element. The last namespace introduces a new element called *list*, which can contain and graphically lay out other components, e.g., *Doer, Viewer*, and it is the root element of this application.

In Figure 10, the XML Browser displays the content of the compound XML document using the described components to realize its semantics. In the figure, the top is the SXHTML application and the middle is the stock quote application. On the bottom, there are two applet elements playing MPEG-4 video clips, which is a new feature introduced by this application. The content of the XML file is a simple composition of previous two applications with their own namespaces, and the new applet-like elements. The actual Java applet code, for the applet component, is obtained from the IBM Toolkit for MPEG-4 [19].

While this is a simple sample, more functionality could easily be added, in the form of components, to build a real world application. Consider the employee example. If the employee information (parts or all) is combined with some other elements, i.e., job information, and the components for both the two elements exist, there is no need to write a new processing program for the compound document. This sample demonstrates how an application can be constructed by assembling existing components in the XVM framework.

5. RELATED WORK

XML is an emerging standard for data representation and data exchange on the Internet. XML-based web applications have been widely used in e-commerce and enterprise information management. How to effectively design, build, and deploy web-based applications has been an active field of research. In the software engineering area, software development principles and guidelines [1][5][9] have been suggested. The Web-Composition model [8][9] defines an object-oriented model for constructing



The Nation's Weather

By The Associated Press

Much of the country got at least a little wet Friday, although parts of the central and southwestern United States were dry.

Areas of northern, central and western Texas were hit with many showers and strong to severe thunderstorms, as well as a couple of tornado sightings. Hail 2.75 inches in diameter was reported in Boys Ranch, Texas.

Stock Quote List

Symbol	Company Name	Price ($)	Last Trade
IBM	International Business Machines ...	79.85	8/5 16:01
SBC	SBC Communications	23.23	8/5 16:02
AXP	American Express Co.	42.45	8/5 16:01
MSFT	Microsoft Corp.	25.66	8/5 16:43

Figure 10: Snapshot of the Compound Application.

Web applications. The components in this model are described using WCML [7], an application of XML. WCML defines specialized XML vocabulary that provides a simple notation for objects and their relationships. A compiler translates a WCML document into a target form. BML [26] is an XML-based component configuration or wiring language customized for the JavaBean component model. Unlike WCML and BML, XVM introduces no new XML vocabulary. The XVM is a runtime middleware and, at the same time, it can utilize the Object-Oriented concepts in XML Schema [24] during development. XML Schema provides Object-Oriented design principles. Normally, an XML schema can be designed following Object-Oriented methodology. Consequently the XML elements usually represent real world objects and the associated components can be viewed as their behaviors.

J2EE [20], a Java-based technology, provides a high-level component-based approach to the design, development, assembly, and deployment of applications. The component of the XVM is a lower-level software unit with clearly defined interfaces. Applications are dynamically composed of these components at runtime based on the XML document being processed. One important goal of the XVM is to provide an architecture that enables the dynamic composition of components.

Many efforts have been made to extend the functionality of current web browsers, e.g., the browser plug-in [21] and Java applet. Unlike the XVM, Java applet is not specially designed for XML data processing. The Browser plug-ins can be used to process different types of documents. However, since it is a coarse granularity software unit, it is hard to deploy or upgrade. There has been some research work in the literature to let client-side browsers directly support XML processing [12][13][14][15].

Several XML application development tools have been developed [16][18][20][22]. They provide the basic functionalities for XML data processing, such as data loading, parsing, etc. The XVM utilizes these tools and provides a more general framework for XML applications. Cocoon [22] is a Java server framework that allows the dynamic publishing of XML content using XSLT (XML Stylesheet Language-Transformation) [10] transformations. As we have mentioned before, the transformed data still needs to be processed by the client side. The XVM architecture can be used to implement not only the functionality of XSLT (transformation), but also more sophisticated processing of XML documents.

With more and more web applications exchanging information on the Web, we are facing the problem of how to integrate the heterogeneous and loosely coupled applications and data sources. Web services provide interoperability and extensibility by the use of XML. Since the XVM is an open architecture for XML applications, it can also be used to implement and deploy web services applications. For example, Dynamic XML documents [1] allow web service invocations embedded in XML documents. Parts of an XML document are generated by the results from web services. In such an application, the XVM can associate a component with the web service invocation elements, e.g., `fun`, to handle the embedded invocations. The Web Services Description Language (WSDL) [4] is a standardized XML format for describing network services. The description includes the name of the service, the location of the service, and how to communicate with the service. WSDLs can be stored in UDDI [23] registries and/or published on the Web. For web services defined using XML, web service modules can also be loaded by the XVM runtime to accept remote requests. XL [6] is an XML Programming Language whose only type system is the XML type system. Both the input and the output of an XL program are XML messages. Unlike XL, the XVM is not a programming language.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a component-based XML processing framework, the XVM. A component to XML element association connects the XML data with its behavior. By using this component-based technique the XVM provides high degree of extensibility and reusability. New components can be easily added to existing applications and new applications can reuse existing components without difficulty. These features enable an XML application to keep up with requirements and schema evolution, and to process compound documents. With this flexible component-based architecture application designers and developers have the freedom to choose the right component granularity for their applications.

By using the XVM application framework, the deployment of an application becomes simple and automatic. The registry mechanism provides component information for the XVM so that components can be dynamically downloaded and bound to a running application. Registering application components allows the framework to deploy and run these applications. The components of an application are dynamically downloaded and composed according to the information in the registry for the XML data being processed. The prototype implementation together with sample applications demonstrated the applicability of the XVM framework.

In the future, we plan to support SAX interface inside the XVM. With this it will not be necessary to load the whole document inside the XVM before running. Applications can use this interface to improve performance, which is especially important for server side applications. XML Events [25] is an example of a class of XML languages that provide document-level behaviors, and are expressed as a set of attributes rather than elements. They typically address non-functional, "cross-cutting" concerns such as messaging, synchronization, etc., making it difficult to incorporate as components into primarily functional decomposition models such as the XVM. We intend to investigate the techniques developed for Aspect-oriented Programming [11].

Since the components running inside the XVM are not always trusted, the security concerns must be addressed in the XVM framework. The security policy of Java can be used in the prototype implementation. To support components written in other programming languages, a comprehensive security framework is needed. For further adoption and deployment of XVM on the client side, we plan to implement XVM plug-in modules for currently widely used web browsers, such as Internet Explorer and Netscape/Mozilla. With such plug-in modules, XVM applications will be capable of running on most existing clients without the need of a full installation of XVM framework.

A complete XVM-based development process model, including tools for analysis, development, deployment, and performance evaluation, is an important future work for the XVM framework that we intend to investigate in the future.

7. ACKNOWLEDGMENTS

We would like to thank Dengfeng Gao for spending time to review and comment on this paper. We would also like to thank the anonymous reviewers for their valuable comments on the earlier draft of this paper.

8. REFERENCES

[1] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML Documents with Distribution and Replication. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 527–538, San Diego, California, June 2003.

[2] M. Bichler, A. Segev and J. L. Zhao. Component-Based E-Commerce: Assessment of Current Practices and Future Directions, In *ACM SIGMOD Record: Special Section on Electronic Commerce*, 27(4): 7–14 (1998).

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 second edition W3C recommendation. RECxml-20001006, *World Wide Web Consortium*, October 2000.

[4] R. Chinnici, M. Gudgin, J. Moreau and S. Weerawarana. Web Services Description Language (WSDL). *W3C Working Draft*, June 2003.

[5] J. Conallen. Modeling Web Application Architectures with UML. In *Communications of the ACM*, 42, No.10:63–70, 1999.

[6] D. Florescu, A. Grünhagen and D. Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. In *Proceedings of 11th International World Wide Web Conference* (WWW2002), Honolulu, Hawaii, USA, May, 2002.

[7] M. Gaedke, D. Schemph and H.-W. Gellersen. WCML: An Enabling Technology for the Reuse in Object-Oriented Web Engineering. In *Poster-Proceedings of the 8th International World Wide Web Conference*, Toronto, Ontario, Canada, May 1999.

[8] M. Gaedke and G. Graf. Development and Evolution of Web-Applications using the WebComposition Process Model. In *Proceedings of International Workshop on Web Engineering at the 9th International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000.

[9] H. Gellersen and M. Gaedke. Object-Oriented Web Application Development, In *IEEE Internet Computing*, Vol. 3, No. 1, pp. 60-68, January/February 1999.

[10] M. Kay. XSL Transformations (XSLT) Version 2.0, *World Wide Web Consortium*, May 2003.

[11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented Programming, In *Proc. of the 1997 European Conf. On Object-Oriented Programming*, Finland (June 1997), Springer-Verlag, LNCS 124.

[12] E. Köppen, G. Neumann and S. Nusser. Cineast - An Extensible Web Browser, In *Proceedings of the WebNet 1997 World Conference on WWW, Internet and Intranet*, Toronto, Canada, November 1997.

[13] C. Mascolo, W. Emmerich, and H. De Meer. XMILE: An XML based Approach for Programmable Networks, In *AISB Symposium on Software Mobility and Adaptive Behaviour*. York, UK. March 2001.

[14] F. Vitali, L. Bompani, and P. Ciancarini. Hypertext Functionalities with XML, In *Markup Languages: Theory & Practice* 2.4 (2001): 389-410.

[15] P. Vuorimaa, T. Ropponen, N. von Knorring, and M. Honkala. A Java Based XML Browser for Consumer Devices, In *SAC'02, Symposium on Applied Computing*, Pages 1094–1099, Madrid, March 2002.

[16] W. Zhao, D. Kearney and G. Gioiosa. Architectures for Web Based Applications. In *Fourth Australasian Workshop on Software and Systems Architectures*, February 2002.

[17] The Apache Software Foundation. The Apache XML Project, http://xml.apache.org/.

[18] The XML C parser and toolkit of Gnome, http://xmlsoft.org/-index.html.

[19] Composite Media Group. IBM T.J. Watson, IBM MPEG-4 Technologies, http://www.research.ibm.com/mpeg4/-index.htm.

[20] Sun Microsystems, Inc. Java 2 Platform, Enterprise Edition (J2EE). http://java.sun.com/j2ee/.

[21] The Mozilla Organization. Plugins. http://mozilla.org/-projects/plugins/.

[22] The Apache Software Foundation, http://www.apache.org.

[23] Universal Description, Discovery and Integration of Web Services, http://www.uddi.org.

[24] World Wide Web Consortium, http://www.w3.org/.

[25] XML Events, http://www.w3.org/TR/xml-events/.

[26] Bean Markup Language (BML), http://www.alphaworks-.ibm.com/tech/bml.