GX-GUI: A General Extensible Technique for 2-D Interaction with VR Applications

Bita Gorjiara, Falko Kuester, Pai Chou and Mehrdad Reshadi

Technical Report CECS-03-46
January 2003

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425
(949) 824- 1421
{bgorjiar, fkuester, chou} @ece.uci.edu, reshadi@ics.uci.edu

GX-GUI: A General Extensible Technique for 2-D Interaction with VR Applications

Bita Gorjiara, Falko Kuester, Pai Chou and Mehrdad Reshadi

Technical Report CECS-03-46
January 2003

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425
(949) 824- 1421
{bgorjiar, fkuester, chou} @ece.uci.edu, reshadi@ics.uci.edu

Abstract

Continuously increasing complexity of collaborative virtual environments demands new interaction paradigms. In particular, interactions such as object selection and manipulation, information query and data augmentation can be made available using customized 2D interfaces for 3D environments. When combined with hand-held devices these interfaces allow user-centric control and customized access to information in multi-user environments. Current development techniques for 2-D interface applications need expertise in complex programming languages and client/server concepts that can impede the widespread implementation of this paradigm. This paper introduces GX-GUI (General eXtensible Graphical User Interface) for 2-D interface development. GX-GUI uses a combination of XML and XSL (eXtensible Stylesheet Language) to keep data and its view. It hides client/server programming issues and eliminates the need for using complex programming languages from interface developer. Using this model, a prototype client-server application has been developed and tested. In order to test the integration of XML data and XSL views an Active Server Application (ASP) is developed that mimics the behavior of the client software and shows the appropriate views independently of the VR environment.

Contents

List of Figure

GX-GUI: A General Extensible Technique for 2-D Interaction with VR Applications

Bita Gorjiara, Falko Kuester, Pai Chou and Mehrdad Reshadi
University of California, Irvine
{bgorjiar, fkuester, chou} @ece.uci.edu, reshadi@ics.uci.edu

Abstract

Continuously increasing complexity of collaborative virtual environments demands new interaction paradigms. In particular, interactions such as object selection and manipulation, information query and data augmentation can be made available using customized 2D interfaces for 3D environments. When combined with hand-held devices these interfaces allow user-centric control and customized access to information in multi-user environments. Current development techniques for 2-D interface applications need expertise in complex programming languages and client/server concepts that can impede the widespread implementation of this paradigm. This paper introduces GX-GUI (General eXtensible Graphical User Interface) for 2-D interface development. GX-GUI uses a combination of XML and XSL (eXtensible Stylesheet Language) to keep data and its view. It hides client/server programming issues and eliminates the need for using complex programming languages from interface developer. Using this model, a prototype client-server application has been developed and tested. In order to test the integration of XML data and XSL views an Active Server Application (ASP) is developed that mimics the behavior of the client software and shows the appropriate views independently of the VR environment.

Keywords:

Human Computer Interaction, 2-D Interface Design, GUI, Virtual Reality, XML, XSL, Hand-Held Device, User Interface Development Technique.

1. Introduction

A common approach to developing user interfaces for 3-D applications is based on 3-D interaction techniques. While 3-D user interfaces work well for spatial interaction, many other types of interactions can be represented more efficiently with 2-D interfaces. For example, text view and annotations, item selection, parameter adjustment, generating complex commands, accessing system information, etc [Bowman et al. 2001; Hartling et al. 2002].

A 2-D GUI can be run on a hand-held device for user interaction or on a desktop workstation for runtime supervisory or debugging purposes. There have been many previous attempts on using handheld devices for interaction with VR applications [Park et al. 2001; Greenhalgh et al. 2001; Chen 2001; Benelli et al. 1999; Cheverest et al. 2000; Benford et al. 2001; Hill and Cruz-Neira 2000]. Common to these techniques is that they promote ad-hoc, application-specific clients. This means that much effort can be potentially wasted in adapting or redesigning the clients to new applications.

In order to extensively use 2-D interfaces in future virtual reality applications, it is desirable to develop new User Interface Development Techniques (UIDT) that reduce the complexity of 2-D UI development. At the same time reusability of user interface components is of primary concern and must be supported in the next generation design schemes.

With the rapidly increasing performance of commodity graphics hardware, ever more complex VR applications are being developed for large-scale data sets. In order to manage large-scale models efficiently, hierarchical schemes are used to manage, store, access and render relevant information. **Figure 1** shows a logical hierarchy of objects. This hierarchical representation facilitates reusing objects in new applications. At the same time the UIDT should allow the same reusability for interface data and views.

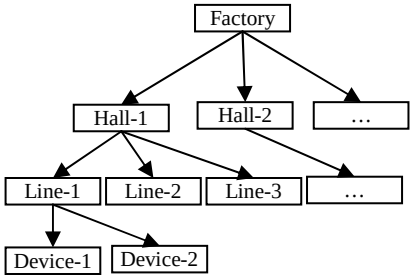

Figure 1: Objects of a 3D environment that models a factory

Different UIDTs have targeted extensibility and reusability in development of 2-D interfaces. [Watsen et al. 1999] has developed an applet loader, which runs on the client and executes applets developed for interacting with different parts of the system. Each time that user enters to a specific part, the interface of that part migrates from the server to the client and the loader will load the applet in the client. [Hartling et al. 2002] has developed a similar system that runs Java Beans on the client, and deploys a collection of technologies (C++, Java, Java Beans, XML and CORBA) to make interface development process systematic and extensible.

On the other hand, there are some commercial and non-commercial attempts at reducing the complexity of VR application development by eliminating the need for complex programming languages such as C or C++. EON Reality [EON 2002] has designed a graphical development environment that allows the user to construct a hierarchy of objects and define action for these objects. They also use scripting languages for complex events and behaviors. [Griepp and Cruz-Neira 2002] has designed an XML schema for rapid development of synthetic environment applications. The schema works with XML editors for auto completion and error checking. This base description is then compiled into application code for execution.

For VR applications most of the available 2-D interface development methodologies are using different complicated programming technologies. In this project we present a new approach, General eXtensible Graphical User Interface (GX-GUI) that uses a combination of XML and XSL to maintain the data and the interface description. The technique is similar to certain web services that automatically generate interfaces for their applications. Using this technique, the interface design becomes simpler and ultimately faster by changing it into web site design rather than programming. Because of the simplicity of this approach and its text-based format, it can be seamlessly integrated with visual and interpretive development tools as well as traditional programming-based development environments.

Using this model, a prototype client-server application has been developed and tested. In order to test the integration of XML data and XSL view, an Active Server Application (ASP) is developed that mimics the behavior of client software and shows the appropriate views independently of the VR environment.

Section 2 presents a short introduction to XML and related technologies used in this project. In Section 3, GX-GUI is introduced and the format of messages is defined. Implemented components and system functionality are described, by example, in Section 4.

2. Introduction to XML, XPath, XSL, DOM

In this section we briefly introduce XML and its related concepts used in GX-GUI. We also show some examples to illustrate the concepts.

2.1 XML Format

The first standard version of Extensible Markup Language (XML) was introduced in 1998 by the World Wide Web Consortium (W3C). Like HTML, it is based on Standard Generic Markup Language (SGML), a mark-up language designed for storing very large structured documents. Tags and attributes in XML describe the structure and meaning of the data and in fact, XML is both data and structure. XML grammar relies on regular expressions and consequently its grammar is simple and it is fast to process.

In XML every object is represented by a begin-tag and an end-tag and each begin-tag can have some attributes. Tags that appear between a matching pair of a begin-tag and an end-tag, represent children objects. An object that does not have any child can be represented with a single tag, called empty-tag. XML is very powerful in representing tree-like data structures.

2.2 Xpath

XPath is a search language for addressing specific elements in an XML file [Goldfarb 2000; W3]. The XPath data model views a document as a tree of nodes, leans heavily on familiar description of a document and uses genealogical taxonomy to describe the hierarchical makeup of an XML document. It refers to children, descendents, parents and ancestors. **Figure 2** shows a simple XML example and the corresponding tree.

```
<?XML version="1.0"?>
<class name="test">
  <protection type="public">
     <param type="int"
name="m_count"/>
     <param type="char"
name="m_char"/>
  </protection>
</class>
```

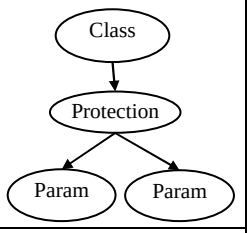

Figure 2: Familial Tree of an XML code

In XPath the first "/" selects the root and the tree can be traversed using element names. For example "*/class/param*" selects the *param* element whose parent is *class*. Symbols "@", "*", and "." select *attributes*, *all* and *current* element(s), respectively. A very important part of XPath is its predicates, which is useful in conditional selection of tree nodes. These predicates are XPath expressions enclosed in brackets ([ ]). For example */class/*[@type="int"]* selects every child of class for which the attribute named type has a value of *int.* Also */class/param[2]* selects the second *param* of every *class* child of the root. Predicates are also cascadable; for example */class/param[2][@size]* selects the second *param* children of any *class* that the selected *param* element has an attribute named *size.*

2.3 XSL

Extensible Style sheet Language (XSL) [Harold 1999] is a powerful tool that operates on XML and generates another XML or text formats. It applies a set of recursive rules to tags to generate the new format. In rules the "key" for finding tags is an XPath expression. For example, suppose that we want to generate C++ header code for XML representation of a class shown in **Figure 2**. **Figure 3** shows the XSL format and its corresponding output after being applied on the XML.

```
<xsl:template match="class">
class <xsl:value-of select="@name"/>
{
        <xsl:apply-templates select="protection"/>
};
</xsl:template>
<xsl:template match="protection">
<xsl:value-of select="@type"/>:
        <xsl:apply-templates select="param"/>
</xsl:template>
<xsl:template match="param">
        <xsl:value-of select="@type"/> <xsl:value-of
select="@name"/>;
</xsl:template>
```

```
class test
{
public:
        int m_count;
        char m_char;
};
```

Figure 3: Example: XSL for generating C++ code and the output

2.4 Programming APIs

Every XML structure must be processed using a program. W3C, the World Wide Web Consortium, which supports XML and related standards, provides an API for XML programmers. This helps the programmers use any implementation of this API for loading and saving XML files. This API is called DOM (Document Object Model) and provides a library of functions and classes for creating and accessing XML data structures in the memory. Compared to XPath and XSL, it is more flexible but requires more programming efforts.

A tree containing interface data corresponding to the object hierarchy (**Figure 1**) can be captured in the XML format. Each object contains its own data, and when generating the data structure for the whole environment, every object asks its children to generate this XML format. The tree is stored in the server application and a sub-tree will be sent to the client. The sub-tree generated for the client will vary based on user navigation in the environment. The parts that are not available can be transmitted to the client on demand. We load the XML data to the DOM data structure for future use. As we mentioned before, XSL can transform XML format to many other text formats. We have chosen HTML as the output format that can be viewed graphically by the client.

3. The GX-GUI

The General eXtensible Graphical User Interface (GX-GUI) uses XML format to model data and XSL for representing views. Several views may be associated with a given data model in XML. To create each page of interface one of these XSL descriptions will be used to generate the HTML view for the data modeled in XML.

Figure 4 shows a block diagram of GX-GUI model. In the beginning, the VR application sends an XML file along with some XSL files. The XML file is loaded into the DOM data structure and the XSL files are loaded into a library object. The first XSL is then applied to XML to create an HTML view. As the user clicks on the hotspots (hyperlinks and buttons), data in the

hyperlink expression will be used to update the DOM data structure, to update the status of the server application and to choose the next XSL for the next view.

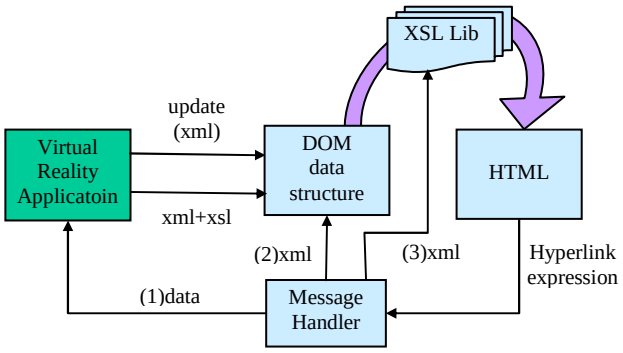

Figure 4: Block Diagram of the GX-GUI model

3.1 Data Exchange Formats

In regular HTML pages there is a tag under each link that keeps the information of the next page. In the rest of the paper we refer to this information as "hyperlink expression." Here is an example:

```
Please <a href="http://www.uci.edu/index.htm"> click
here </a>.
```

Figure 5: Example: Hyperlink expression

In GX-GUI "hyperlink expression" is composed of three parts (marked bye (1), (2) and (3) in **Figure 4**). By designing a good format for these three parts, the entire program will be very general and flexible. As seen in **Figure 4** the "data" part of "hyperlink expression" will be sent to the server. This part does not have any predefined format and the developers of server application can define their own. The format of the second part, which is used for updating the DOM object, is shown in **Figure 6**.

```
<mssgs>
  <msg type='…' xPath='…' ID='…'> … </msg>
</mssgs>
Message Types:
addChild
addSibling
Delete
deleteAllChildren
Replace
ChangeAttributeValue
```

Figure 6: Format for part 2

This part has a predefined XML format and sends different basic messages to DOM data structure, such as "addChild," "delete" and "changeAttributes." Using XPath expression a particular node will be selected and modifications will be applied to it. Each message (<msg>), based on its type, may or may not have a child node. For example for "addChild" message we need a tag for the new child that should be added to the data structure. **Figure 7** shows an example of "addChild" message.

```
<msg type='addChild' xPath='/*[1]/*[2]' >
  <shape color='blue' type='circle' width='100'
height='200' />
</msg>
```

Figure 7: Example of part 2

The last part of the hyperlink expression is another predefined XML format that is used for selecting the next XSL from the library (Figure 8). Attributes of <info> can be used for passing parameters to XSL files.

```
<info XSLFile='FileName' param1='…' … paramN='…'/>
```

Figure 8: Format of part 3

To maximize flexibility and generality, we impose minimum constraints on the XML and XSL formats. The initial XML file (goes from server to client) does not have any particular restriction; it should be just a valid XML. The XSL formats can generate any desired HTML, but the only requirement is that hyperlink expressions should follow the format of **Figure 9**.

```
xmlFileName=...&
submit=...&
mssgs=<mssgs>
<msg type="" xpath=""> <…xml data…></msg>
        </mssgs>&
nextPage=<info xslFile="FileName" param1=""
param2=""..../>
```

Figure 9: Format of hyperlink expression

Some update messages can also be sent from the server to the client to initialize and update available DOM structure. These messages should follow the format of **Figure 6** as well.

3.2 The Model Benefits

Integrating the XML, XSL and HTML technology has made this approach very powerful and flexible. In addition to flexibility, there are also several other advantages:

- By keeping a local database of required data and getting different views of it, the client minimizes reference to the server.
- The data and the view of application come to the client based on the user demand and there is no need for loading the information of the areas that user does not explore.
- Designing the interface for a particular part of program is now simplified tremendously. It is more like designing a web site rather than writing a program and getting involved with network concepts. This is the most important advantage and can have a great effect on implementation time.
- It is easy to integrate the interface to VR design tools because it uses a text-based format.

In order to enhance interactivity, we can use scripting languages in HTML files.

4. Implementation

In the following we explain different components of applications developed to verify this model. We also show the process of interface development for an object editor example.

4.1 Client application

The modules of the client program are shown in **Figure 10**. The HTML viewer is a Pocket PC standard control and displays generated HTML pages. The Socket handles the connection to the server and sends and receives data. The Message Handler receives the messages from the Socket and HTML viewer and either handles them or sends them to the "Server Stub." The Server Stub then loads the initial XML file, updates the DOM data structure, selects XSL from the library, applies it to the data structure and generates the desired HTML format. This module uses MSXML library to parse XML and works with DOM.

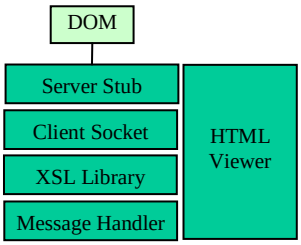

Figure 10: Modules of GX-GUI Client

4.2 ASP application

We have developed an ASP server that mimics the behavior of the client application for manipulating data and generating views, in order to test the interface independently of the VR application. This setup can greatly assist interface debugging.

The modules of the ASP application are highlighted in **Figure 11**. Once the user sends a query to the ASP server by clicking on a link or a submit button (in the format of **Figure 9**), the ASP application loads the database file to the DOM data structure, applies changes to it, selects and loads one XSL from the library, applies the XSL to DOM, generates HTML, and finally saves DOM to XML for future use. The generated HTML is then forwarded to the user's browser.

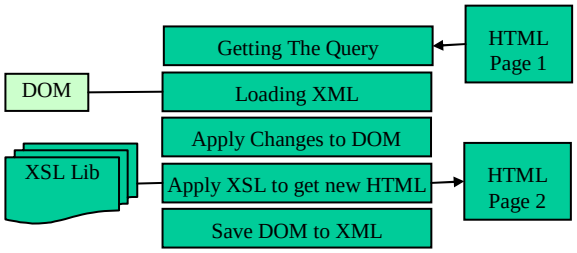

Figure 11: Modules of ASP application

4.3 Example: Object Editor

In this section, we show the process of interface development for a simple application. Suppose that we have a scene and we want to put different kinds of objects in this scene and place them at the desired positions, using a 2-D interaction application. The first step is to determine the data that should be kept on the client and also determine the views that we want to see.

Figure 12 shows the XML file and two views of this XML. One of the views is a tree that shows the available objects in the scene and the other is the edit form for the object properties. When the user clicks on a link usually a different view of the XML database is shown and there is no interaction with server, but when the user clicks on a button, an event will be sent to the server and the status of the DOM structure should be updated.

```
<?XML version="1.0" encoding="UTF-8"?>
<scene name="scene 1" lastID="1" ID="0">
<obj name="cube1" x="100" y="100" z="150" w="100"
h="200" depth="50" yaw="10" pitch="5" roll="10"
type="cube"  ID="1"/>
</scene>
```

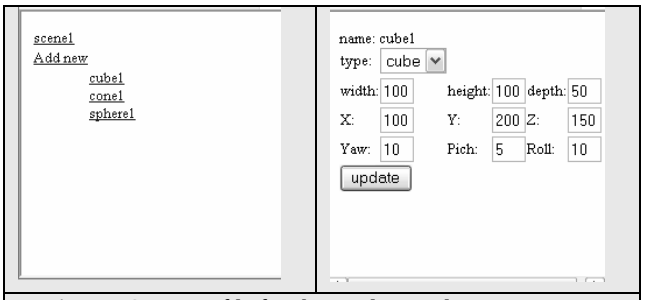

Figure 12: XML file for shape editor and two HTML view

Three XSL files must be designed to generate three HTML files: One for the menu page, another for the "Add New" page and the third one for the "Edit" page. **Figure 13** shows the XSL script for retrieving the menu page and corresponding HTML.

```
<xsl:template match="scene">
<a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=&lt;info xslFile='showSceneInfo.xslt'/&gt;">
<xsl:value-of select="@name"/> </a><br/>
<a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=&lt;info xslFile='addShape.xslt'/&gt;"> Add new </a>
        <br/>    <ul><xsl:apply-templates
select="obj"/></ul>
</xsl:template>
<xsl:template match="obj">
<li><xsl:element name="a"><xsl:attribute
name="href"><![CDATA[
?xmlFileName=&submit=&mssgs=&nextPage=<info
xslFile='showShapeInfo.xslt' ID=']]><xsl:value-of
select="@ID"/><![CDATA['/>]]> </xsl:attribute>
        <xsl:value-of select="@name"/></xsl:element>
</li>
</xsl:template>
```

```
<a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=<info xslFile='showSceneInfo.xslt'/>">scene 1</a><br>
<a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=<info xslFile='addShape.xslt'/>"> Add new </a><br>
<ul>
<li><a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=<info xslFile='showShapeInfo.xslt' ID='1' />
">cube1</a></li>
<li><a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=<info xslFile='showShapeInfo.xslt' ID='2' />
">cone1</a></li>
<li><a
href="xmlFileName=&amp;submit=&amp;mssgs=&amp;nextPage
=<info xslFile='showShapeInfo.xslt' ID='3' />
">cube2</a></li>
</ul>
```

Figure 13: An example of XSL file and corresponding HTML

In the **Figure 14** the client application, which is running on a hand-held device is shown.

Figure 14: Client application running on iPaQ

Conclusion and future works:

This paper presents a general model for developing 2-D interface applications running on hand-held devices to interact with VR environments. The model uses a combination of XML and XSL to maintain data and the interface description respectively. A client application takes care of the process of loading and manipulating the XML format, and extracting different views. The interface designer defines the type of manipulation in a predefined message format. The model is deigned to be extensible for increasingly more complex VR environments by separation of interface data and its view.

Although designing the interface using this approach is not as simple as creating HTML files but it is much simpler than designing the entire interface in programming languages. Besides, there exist many powerful tools that can help developers in this way. It can also be used on desktop computers for runtime supervisory.

References:

[1] Benelli, G., Bianchi, A., Marti, P., Not, E. and Sennati, D. 1999. HIPS: Hyper Interaction within Physical Space. In *Proc. of International Conference on Multimedia Computing and Systems (IEEE ICMCS'99)*, p.1075-1078.

[2] Benford, S., Bowers, J., Chandler, P., Ciolfi, L., Flintham, M., Fraser, M., Greenhalgh, C., Hall, T., Hellström, S. O., Izadi, S., Rodden, T., Schnädelbach, H. and Taylor, I. 2001. Unearthing virtual history: using diverse interfaces to reveal hidden virtual worlds. In *Proc. of International Conference on Ubiquitous Computing (UBICOMP'2001)*, p.225-231.

[3] Bowman, D., Kruijff, E., LaViola, J., and Poupyrev, I. 2001. An Introduction to 3D User Interface Design. In *Teleoperators and Virtual Environments Journal*, vol. 10, no. 1, p.96-108.

[4] Chan, W., 2001. Project Voyager: Building an Internet Presence for People, Places, and Things. Masters Thesis, Department of Engineering and Computer Science, Massachusetts Institute of Technology.

[5] Cheverst, K., Davies, N., Mitchell, K., Friday, A. and Efstratiou. 2000. Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In *Proc. of Human-Computer Interaction (ACM CHI'2000)*. p.17-24.

[6] EON Reality: http://eonreality.com

[7] Goldfarb, C.F., Prescod, P. 2000. The XML Handbook. Prentice Hall, Second Edition, Ch. 59.

[8] Greenhalgh, C., Benford, S., Rodden, T., Anastasi, R., Taylor, I., Flintham, M., Izadi, S., Chandler, P., Koleva, B. and Schnädelbach, H. 2001. Augmenting Reality Through The Coordinated Use of Diverse Interfaces. Technical Report, University Of Nottingham.

[9] Griepp, T., Cruz-Neira, C. 2002. XJL: A XML Schema for the Rapid Development of Advanced Synthetic Environments. In *Proc. of* the *Immersive Projection Technology Symposium (IPT'2002)*, p.294-303.

[10] Hartling, P., Bierbaum, A. and Cruz-Neira, C. 2002. Tweek: Merging 2D and 3D Interaction in Immersive Environments. In *Proc. of the World Multi-conference on Systemics, Cybernetics, and Informatics*, Volume VI, p.1-5.

[11] Hill, L.C., Cruz-Neira, C. 2000. Palmtop Interaction Methods for Immersive Projection Technology Systems. In the *Proc. of the International Immersive Projection Technology workshop (IPT2000).*

[12] Park, K. S., Leigh, J., Johnson, A., E., Carter, B., Brody, J. and Sosnoski, J., 2001. Distance Learning Classroom Using Virtual Harlem. In *Proc. of the International Conference on Virtual Systems and Multimedia (VSMM'2001)*, p.489-498.

[13] Watsen, K., Darken, R. P, Capps, M. V. 1999. A Handheld Computer as an Interaction Device to a Virtual Environment. In *Proc. of the International Immersive Projection Technology workshop (IPT'1999).*

[14] World Wide Web Consortium http://www.w3.org/XML