On the Design and Implementation of Interactive XML Applications

Jeff Brown, University of North Carolina Wilmington, USA
Rebecca Brown, University of North Carolina Chapel Hill, USA
Chris Velado, University of North Carolina Wilmington, USA
Ron Vetter, University of North Carolina Wilmington, USA

ABSTRACT

This paper describes issues and challenges in the design and implementation of interactive client-server applications where program logic is expressed in terms of an extensible markup language (XML) document. Although the technique was originally developed for creating interactive short message service (SMS) applications, it has expanded and is used for developing interactive web applications. XML-Interactive (or XML-I) defines the program states and corresponding actions. Because many interactive applications require sustained communication between the client and the underlying information service, XML-I has support for session management. This allows state information to be managed in a dynamic way. The paper describes several applications that are implemented using XML-I and discusses design issues. The software framework has been implemented in a Java environment.

Keywords: Extensible Markup Language, Short Message Service, SMS, Software Architecture, XML

INTRODUCTION

Client-server applications are often written as customized software that is dedicated to providing the information service desired. Each new service is a new application (or procedure) that implements the service. Like the basic web protocol (http), the client-server environment is often stateless — meaning that the environment won’t do anything to preserve the session state. Each request and response is a new one and not related to any other. Therefore when writing interactive client-server applications, session management is a major part of the effort since one must implement stateful sessions in a stateless environment.

Multi-tier web development architectures have become commonplace in modern web applications. A multi-tier, or more specifically a 3-tier, architecture refers to a client-server architecture in which the presentation, application processing, and the data management layers are...
logically separate processes (Wikipedia, n.d.). The idea is to modularize the software application into functional units with well-defined interfaces so that changes and upgrades to any software layer (tier) can be made independently. Web application frameworks, such as Java Enterprise Edition (J2EE) and Ruby on Rails, are often used to support the development of dynamic web sites, web applications, and web services. These frameworks help alleviate the overhead associated with common activities performed in web development, such as providing libraries for data access, security, and session management (Wikipedia, n.d.).

In a recent article, a flexible framework for developing interactive short message service (SMS) applications was presented (Brown et al., 2010). This framework was called XML Interactive or XML-I. In this paper, we show how XML-I can be used not just for creating interactive SMS applications but also for creating any interactive client-server web application that requires state information and state control. Although this work is a relatively new, it builds upon the established work of more general web application frameworks. With XML-I, program logic is expressed in terms of an XML document which contains the program states and corresponding actions. Because many interactive applications require sustained communication between the client and the underlying information service, XML-I also has support for session management. This allows state information to be managed in a dynamic way. This paper describes several applications that have been implemented using XML-I and discusses design issues. The software framework has been implemented in a Java environment.

WHAT IS XML-I

XML Interactive is a specification, in XML format, of a software application. Thus, XML-I can be used for both the conceptual and the actual design of interactive SMS and web applications. The software framework of XML-I contains properties that can be easily implemented by simple programs in a web server environment, using Java’s application programming interface (API). By establishing XML-I in a multi-tier architecture, web services can be extended to create complex client-server conversations based on the input received from a mobile originated (MO) SMS message or more traditional web (HTTP) request.

Before the introduction of XML-I, many interactive SMS services were implemented by responding to a single and specific keyword contained in the MO message. The SMS applications would generate mobile terminated (MT) messages based upon the specific keyword received. With XML-I, users are able to carry out multifaceted conversations with the underlying information service without the need to specify a keyword for each client-server interaction. For example, a user might navigate through several menu options (responding via SMS messages) in order to request more specific information from a large data set.

Just as standard XML is based on tree structures, XML-I code contains specific nodes and attributes that control the navigation within the XML along with key commands to enable client interaction. When a user sends a message, the content within that message determines the new “state” of the application. Moving to a new state will determine what actions will take place. Basic functionality such as switching state or terminating conversations is also part of this system. Changing an existing application is easy – one simply updates the states and actions in the XML document.

SMS APPLICATION – DRUG TRIAL PRESCREENING

Mobile Education LLC has established the hardware and software infrastructure needed for implementing interactive SMS applications (Brown & Vetter, 2010). One of these applications involves prescreening applicants who are interested in participating in experimental drug trials. The basic idea behind this application can be described as follows.
Participants are sought for an upcoming drug trial. Ads say “Text TRIAL to 90947” to see if you qualify. In this simple example, we are going to look for participants who are females between 18 and 24 years of age. However, we have an upcoming trial for which we want males in the same age range. The user sends an MO with the word TRIAL to 90947.

The system sends an MT with the text, “Welcome to the drug prescreening system. First, you need to answer some basic questions. How old are you? a) Under 18 b) 18-24 c) Over 24.” If the response is ‘a’ or ‘c’, the user is thanked for their interest and told that they are not in the right age range and the conversation is over. If the response is ‘b’, the system responds with the text, “What is your gender? a) Female or b) Male.”

If the response is ‘a’, then the system sends the text: “Thank you for your interest. You will be contacted shortly.” There are also three additional actions taken:

1. A text is sent to a company representative with a message that includes that user’s phone number.
2. An email is sent to a company representative with a message that includes that user’s phone number.
3. The user’s phone is associated with a text message group to whom broadcast messages can be sent.

If the response is ‘b’, the user is told that they are not needed for the current trial, but they might be suitable for an upcoming trial. In this case the action is to associate the phone number with a text message group for future contact. Figure 1 shows the XML-I for the drug trial application.

WEB APPLICATION – LEARNING TREES

A learning tree depicts a series of choices or decisions that are possible by structuring learning content into the form of a tree (Wikipedia, n.d.). Leaves denote final answers and the branches represent the choices that lead to those answers. Although the composition of learning content into a learning tree is an active area of research (Woda, 2006; Witten & MacDonald, 1998), one simple strategy for developing a learning tree might be described as follows:

- Start with a series of questions on one topic.
- For each question choose two or more answers.
- Each wrong answer corresponds to a new question sequence which, when completed successfully, leads to the same place you would have reached with the right answer.

For example, a learning tree that depicts several mathematical problems for elementary school students is shown in Figure 2. Each question in the tree has two edges below it, and the edges are labeled with numbers. This means that the question is stated as multiple-choice with two answers. The question at the root node might be stated as “Fill in the blank: \[3 + 28 = \_\]. Answer 25 or 31.” If the student answers something other than 25 or 31, then the application can be set so that it either states the problem in another way, or it sends the same question again.

The three initial questions are the root node, the root’s right child, and that child’s right child. So answering the three initial questions correctly takes you down the right side of the tree to the node labeled 3. This tree is binary – every non-leaf node has two children. It is likely that a real example would have questions with more than two answers, but it is easier to see the overall scheme with a binary tree.

It is also likely that a real-world example might require more than one right answer to correct a series of wrong answers. For example, suppose your answers are 25, then 11, then 13. This leads you to a node labeled “go to 1”, which means you go to the node labeled with the 1 corresponding to getting the first answer right. Having missed two and gotten one right, at this point there might be another question that
would have to be answered correctly before jumping to node 1.

The nodes labeled ‘Give up’ indicate that you have three wrong in a row, and that you might need to be directed to another place. Figure 3 shows partial XML-I code for the mathematics learning tree.

To implement learning trees, there must be a simple way to put a set of questions into production, that is, construct the tree and generate the XML-I code. The example above was started by drawing a tree on paper. Real-world examples will probably contain enough branching that drawing the tree by hand will be difficult. Thus, there is a need for software that guides the input of questions and answers. Having a clear strategy will make it easier to design applications that let pedagogy experts build trees in a wizard-like environment. It is also important to note that the introduction of wildcard route attributes in XML-I allow us to create learning trees with questions that are not multiple choice questions. This is explained in more detail in the section on Design and Implementation Issues.

The strategy used for this learning tree resulted in one leaf node being the desired goal, and there is only one path through a tree leading to that goal. The jump feature in XML-I essentially turns the XML tree into a graph. This means that a learning tree (or graph) can describe complex navigation schemes, and hence complex applications can be specified.

The text message context is sufficient for the very basic mathematics in the learning tree example, but it is not suitable for more advanced mathematics. This is what led us to start thinking about extending XML-I to support web applications. In the web version of XML-I, the message attribute is replaced by the URL of a page to display. Links and text inputs are used in place of MO text messages.
Figure 2. Learning tree for mathematics

![Learning tree for mathematics](image)

Figure 3. Partial XML-I code for math learning tree

```xml
<xmlInteractive id='root' message='3 + 28 = _ Reply either 25 or 31'>
  <state id='w1' route='25' message='14 + 3 = _ Reply 11 or 17'>
    <state id='w2' route='11' ... />
  </state>
  <state id='r2' route='17' jump='r1'/>
  <state id='r1' route='31' message='Right! 0 + 78 = _ + 73 Reply 78 or 5'>
    ... 
  </state>
</xmlInteractive>
```
We have implemented a simple maze as a web XML-I application that can be seen at http://
mymobed.com/xmli. Each page of the maze application contains the name of a room and
links to adjacent rooms. The goal is the make it to the room called “You made it to the End!”
Note that this maze uses the jump feature of XML-I, and the link called “Back” takes you
to the parent of the current node.

INTEGRATED SMS/ WEB APPLICATION
The flexibility of XML-I allows for the de
velopment of comprehensive interactive applications. For example, we have developed a
fully functional SMS/Web application, called TakeTextPoll (Brown et. al., 2010b), which
uses SMS technology to deliver interactive text message surveys and polls to clients on their
cell phone. Users text a keyword and receive questions as a series of text messages. As users
respond to messages the system tallies the responses and results are made available in
real-time on a secure web site. Text message based surveys and polls can be used for market
research, call-to-action campaigns, electronic voting, and more.

TakeTextPoll is set up to be a web application that requires prior approval before its
services are available. Once a client registers through the web site and is approved by the
administrator, he or she is able to log in and start creating original polls. The Create a Poll
page (Figure 4), allows the user to select a poll title, visibility (public or private), keyword,
and a multiple choice question. With the use of JavaScript, a live count of the number of
characters entered is automatically updated and displayed. This is a key issue for creating SMS
polls since a text message has a limit of 160 characters (we limit the maximum number of
characters to 155 to allow for question number to be sent in the text message as well).

Once the user has completed the first ques
tion, they can proceed to add either another
multiple choice question or a free response
question. A free response question is one that
allows the poll taker to respond with an open
response (any text) for the answer to the ques
The application currently allows for up to
five questions per poll.

Users can deploy and un-deploy polls. When a user presses the deploy button, the
poll is translated into XML-I, and sent to
Mobile Education server. Mobile Education’s
ter executes the XML-I application and
relays responses back to the TakeTextPoll
web application. Figure 5 shows an ex
ample of the XMI-I generated from a typical
poll application.

The TakeTextPoll application is imple
mented as a multi-tier web application (Figure 6). The presentation tier consists of two main
user interfaces: a web page and a SMS inter
face. The web page represents the portion
where clients are able to visually create and
manage their polls through web forms.
Within that same tier, a mobile user is able to
send a keyword to begin a SMS poll. The
logic tier includes communication functional
ity. One computer server performs operations
that include communicating between the web
page and poll database, and between the poll
web site and Mobile Education’s server. The
Mobile Education server performs operations
that include communication between a poll
taker’s mobile device and web server. Finally,
the data tier contains a database that stores
account, poll, and poll response information.

An alternative view of the TakeTextPoll
application is depicted by an activity diagram
shown in Figure 7. An activity diagram is a
graphical representation of workflows of the
stepwise activities and actions of a system
(Wikipedia, n.d.).

In the learning tree example the jump
feature was used to provide multiple paths to
the same node, in effect turning the tree into
a graph. The XML-I code produced by Take
TextPoll uses jump to avoid duplicating mate
rial in the XML document.
DESIGN AND IMPLEMENTATION ISSUES

Mobile Education’s SMS system is based on keywords. When a user sends a text message to the short code 90947, the first word of the message identifies the program that will process the message. An important feature of XML-I is that the user does not have to use the keyword after the initial MO message. This feature makes XML-I more user-friendly and more robust. Consider, for example, the gender question from the drug trial prescreening application, “What is your gender? a) Female or b) Male.” If the response needed to begin with the keyword TRIAL, it would be awkward to state the options, and user input errors would be much more common. Mobile Education’s SMS system implements sessions with a program called the Interceptor (Figure 8). When a user starts an XML-I conversation, the user’s phone number, the identifier of the XML-I application, the timeout for that application, and the current time are stored in a database. When a message enters the system, the Interceptor checks the database to see if the phone number is in a session that has not timed out. If so, it appends the appropriate keyword to the front of the message and continues processing.

XML is the natural choice for holding the hierarchical data needed by XML-I applications. Another reason for having the application specified by XML is that other applications can create XML-I applications simply by writing XML. TakeTextPoll is one example of this kind of application and we expect to create more examples in the future.

Figure 4. Create a poll web page
The tree structure of XML also fits well with the question/answer nature of XML-I applications. We have seen in the learning tree and TakeTextPoll applications that it can be necessary to go from the current state to another state that is not a child the current one. The jump feature of XML-I makes it much more flexible.

The current XML Schema used to validate XML-I applications is shown in Figure 9. The schema requires the root element to be named `xmlInteractive` and to have an attribute named id with value ‘root’. All other elements in the document are either state tags or action tags. State tags must have unique id attributes. As the user interacts with the application, the id of their current state is stored in a database.

State tags also need an attribute named route, which is used to determine the next state for the user. If the expected response to a question is either a or b, then the current state tag should have a child with route attribute a, and another with route attribute b. A state’s optional error attribute is used as a message to the user when they send an unexpected response. For example, look again at the drug trial prescreening application. The root element’s message attribute is “Welcome to the drug prescreening system. First, you need to answer some basic questions. How old are you? a) under 18 b) 18-24 c) over 24,” so we expect to receive a response starting with either a, b, or c, and there are child nodes with those route attributes. If we receive any response other than a, b, or c, the optional error attribute is sent to the user. The idea is to restate the required response in a way that might be clearer. In this example the error attribute is “Reply a, b, or c. a) for under 18 years, b) for 18-24, or c) for over 24.”

The route attribute of a state node can also be the wildcard character *. If the current node has a child with route = ‘*’, and the response does not match the route attribute of any other child, then the child with the wildcard route becomes the current node. For an example of

```
<xmlInteractive id='root' message='(1/3) Which do you think TakeTextPoll could be of use? Text A. for Business use, B. for Scientific use, C. for Education use, D. for None of the above' error='reply A. Business use B. Scientific use C. Education use D. None of the above'>

<action type='relay' to='http://TakeTextPoll.com/...'/>

<state id='q2' route='a' message='(2/3) Please give an example of what you could see using TakeTextPoll in the future.'/>

<action type='relay' to='http://TakeTextPoll.com/...'/>

<state id='q3' route='*' message='(3/3) How has this experience been? Text A. for Great, B. for Ok, C. for Bad' error='reply A. Great B. Ok C. Bad'>

<action type='relay' to='http://TakeTextPoll.com/...'/>

<state id='q4' route='a' message='Thank you for your input. For results, check www.TakeTextPoll.com!'>

<action type='relay' to='http://TakeTextPoll.com/...'/>

</state>

<state id='1a3' route='b' jump='q4'/>

<state id='2a3' route='c' jump='q4'/>

</state>

</xmlInteractive>
```
how this can be used, consider the learning tree application for mathematics. Suppose the question is “Fill in the blank: $3 + 11 = \_\_ + 12.” There are three child nodes with route attributes 2, 14, and *. The correct answer is 2. The instructor expects some children to answer 14, because they think of the equal sign as an instruction to do a calculation, not as part of an equation. So, 14 is an expected incorrect answer. All other answers result in the user being sent to the node with wildcard route. Wildcard routes are also used in polls and surveys where a free response is solicited.

One way to extend the capabilities of XML-I is to add new actions, and we have done that after discussing application requirements with new user groups. Another means of extension is to use the relay action, which sends the user input and phone number to another server. TakeTextPoll uses the relay action to process user input to tally the survey results. So relay actions are used to transmit all users input to a second server for processing.

Another issue involves the creation of learning trees. We can use XML editing software (Wikipedia, n.d.) to guide the input of questions and answers, and to take care of syntax details. XML editors also ensure the data supplied is consistent with the document type definition or schema provided. Using an XML editor is also faster and more convenient than entering the XML by-hand. However, because most XML editors are designed to edit documents with very general tags and structures, they do not provide a good solution for creating XML-I applications with a very limited set of tags. What is needed is an XML authoring tool that shows the XML-I states and branches in a graphical interface and that lets pedagogy experts build graphs in a wizard-like environment. The development of such a tool is part of our on-going work.
Figure 7. Activity diagram for TakeTextPoll application

Figure 8. Flow of control and data
Figure 9. XML-I schema: actions, message types, and states

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="actionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="email"/>
      <xs:enumeration value="text"/>
      <xs:enumeration value="register"/>
      <xs:enumeration value="emailUser"/>
      <xs:enumeration value="relay"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="msgType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="140"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="xmlInteractive">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="state" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="action" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute name="id" type="xs:string" use="required" fixed="root"/>
    <xs:attribute name="message" type="msgType" use="required"/>
    <xs:attribute name="error" type="msgType"/>
  </xs:element>
  <xs:element name="state">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="state" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="action" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="message" type="msgType"/>
      <xs:attribute name="route" type="xs:string" use="required"/>
      <xs:attribute name="error" type="msgType"/>
      <xs:attribute name="jump" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="action">
    <xs:complexType>
      <xs:attribute name="type" type="actionType" use="required"/>
      <xs:attribute name="to" type="xs:string" use="required"/>
      <xs:attribute name="subject" type="xs:string"/>
      <xs:attribute name="message" type="msgType"/>
      <xs:attribute name="includePhone" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
CONCLUSION AND FUTURE WORK

The XML-I framework is implemented in Java and deployed in Tomcat 6, an open source JSP/Servlet engine. It also uses a mySQL 5 database to store application configuration and session state. The use of the Java platform worked well because its web components, JSP and servlets, are well suited for service-oriented applications that process requests and construct responses. JSP pages not only allow text-based documents to be rendered as servlets but also allow a more natural approach to creating content. As in many modern applications, XML proved to be an excellent way to package and exchange data. XPath was also an obvious choice since it follows the industry standard way to parse and locate information in XML documents.

The XML-I engine is continually being enhanced to allow for new actions and program states. We are currently working on the development of an XML authoring tool that shows the XML-I states and branches in a graphical interface and that lets pedagogy experts build educational applications in a wizard-like environment. We are also investigating software support tools to help automate the addition of new actions and element types.

REFERENCES


