A Web Service Architecture for Bidirectional XML Updating

Abstract. We propose a Web service architecture for achieving bidirectional XML updating. The update mechanism exploits the power of bidirectioanl transformations so that users can update XML data by editing a view that is generated by some transformation of the XML data. This architecture consists of three tiers: data viewer clients, a bidirectional transformation engine, and some content servers accessible through the Internet. Due to the use of standard Web service technologies, data viewer clients and contents servers can be easily replaced by the ones chosen by users. Users can use this architecture to implement their own applications that exploit the power of bidirectional transformations without the burden of installing and maintaining the bidirectional language package.

1 Introduction

XML is widely used as the de facto standard format of data exchange. In recent years, the role of XML as format of data repository on the Internet is becoming more important because the amount of resources stored in XML format are rapidly increasing. This trend comes from many reasons. First, many kinds of recent application software have supported functionality to export their data in XML format. Second, native XML database is getting more popular and many of relational database systems provide facilities to manage and generate XML data. Finally, some kinds of data that were not be expressed in XML format is now beginning to be expressed in XML format. As a lot of XML resources are stored on the Internet and used by various kinds of applications, the existence of an environment that provides an efficient and easy way to update XML data is critically important for efficient use of XML resources.

Some observations reveal that XML data is rarely retrieved for own sake but more often subsequently transformed to other format to be processed by some Web applications. For example, a Web server runs XSL [1] processor to generate HTML data from XML data for displaying in a Web browser. When the users look at the data in the transformed format in some application, it is often convenient and efficient if we can make modifications of the data by editing on the transformed data, then reflect it to the source XML data rather than by updating the original XML data first. A technique for this process is called "view updating". In the view updating, the source data stored in the database is first transformed to other format called "view" for which the user can understand its meaning more easily and intuitively. The modifications are made on the view directly and they are reflected back to the source data according to some predefined updating policy.

Updating the remote source data on the Internet through various views means that the update mechanism should be adaptable to both the database system that stores the source data and user's application that processes views. This requires the update system to be a modular and extendable component that works efficiently with other components in distributed computing systems. One of key technologies on the current Web, called "Web Services" can address this problem by the standard technologies such as SOAP [2] and WSDL [3].

Our goal in this paper is to develop an XML data update system which can be used to update remote XML data through transformed formats, and has high modularity and extendability to be used with various kinds of applications. This is achieved by combining the two technologies mentioned above, that is view updating and standard Web service technology. In the view updating, we use our bidirectional transformation language Bi-X, that is developed by extending the expressiveness of the existing bidirectional languages proposed in [12, 13] so that it can be used in the architecture for general-purpose XML processing. This is used to get both the view from the original source and the updated source from the modified view. Our main contributions in this paper can be summarized as follows.

- We proposed a novel SOAP-based bidirectional XML update server, which has highly modularity and extendability for general use of XML updating
- We designed a three-tier architecture with a communication protocol to achieve bidirectional updating service
- We have implemented our architecture and demonstrated its usability by a use case

The remainder of this paper is organized as follows. Section 2 gives a detailed explanation of the three tier architecture. Section 3 explains a bidirectional language Bi-X. Section 4 describes the protocol to achieve Bi-X service in the three tier architecture. Section 5 explains implementation of the Bi-X server. Section 6 gives an example of the use of client and content server. Section 7 gives a use case that uses our implementation. Section 8 summarizes related work. Finally, Section 9 gives conclusions.

2 Bi-X Service Architecture

2.1 A Simple Example

We start by giving an simple example that illustrates how to proceed view updating with our service. Suppose we have the following XML data about some bibliography information, which is accessible by giving its URI through the Internet.

```
<bib>
    <book year="2005">
        <title>The Art of Computer Programming</title>
        <author>Donald E. Knuth</author>
        <publisher>Addison-Wesley</publisher>
        <price>19.99</price>
    </book>
     . . .
    <book year="1989">
        <title>Connection Machine</title>
        <author>W. Danny Hillis</author>
        <publisher>MIT Press</publisher>
        <price>25</price>
    </book>
</bib>
```

Suppose we are interested in only title and authors. By sending URIs to access the source XML data and the Bi-X code to transform it, we want to translate the source XML data into an XHTML view so that the title and the authors in books can be displayed as an ordered list on an XHTML editor on the local machine. This can be done by calling *Init* service provided by Bi-X server. The result is the following XHTML document:

```
<html>
  <body>
    <ol>
      <li>
        <div>The Art of Computer Programming</div>
        <div>Donald E. Knuth</div>
      </li>
       ...
      <li>
        <div>Connection Machine</div>
        <div>W. Danny Hillis</div>
      </li>
    </ol>
  </body>
</html>
```

On this view, we can modify titles or authors, insert or delete a list item. Then we invoke *Update* service provided by the Bi-X server so that these modifications are reflect back into the source data. For example, suppose we change the title of the first book from "The Art of Computer Programming" to "The Art of Computer Programming, Volume 1" and insert a new list item that includes the information of a new book whose title is "The Art of Computer Programming, Volume 2" after the first list item. After getting the Update service, the corresponding title in the source data is also changed to the new title and the

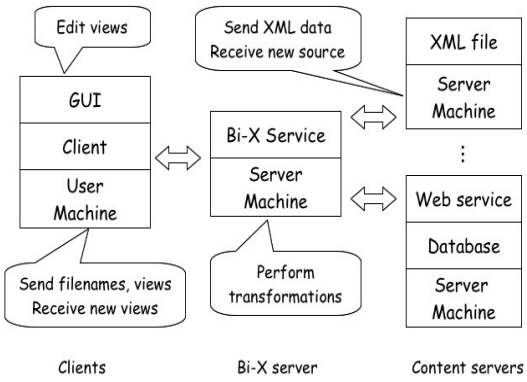

Fig. 1. Three-tier Architecture

new book information will appear after the first book information in the source data.

2.2 Three-tier Architecture

Now we explain the structure of the Web service architecture behind the view updating process. Our Web service architecture consists of three tiers as shown in Figure 1, that is, clients, a Bi-X server, and the content servers that provide XML data. The heart of this architecture is the Bi-X server, which has a bidirectional transformation engine based on an implementation of bidirectional transformation language Bi-X. It receives a request from clients and applies a forward transformation to the specified source data that was fetched from content servers in order to generate the view, or applies a backward transformation to produce the updated source data.

A client of Bi-X services can be any XML data viewer, but typically Web application to display the target data of a Bi-X transformation as a formatted view that is equipped with some editing environment. It receives information given by users, such as parameters to specify a Bi-X code or XML data to be fetched and editing information such as modified target data, and then sends some request messages with the parameters to the Bi-X server.

A content server provides XML data and a Bi-X code for transformations. This is specified by users when they launch transformations. The requirement to a content server is that it can provide XML files, and also can accept modified files and then update the data making up these files. For example, if the file sent to the Bi-X server is just an existing XML file on a machine on the Internet, then when a modified file comes back, the existing file is simply replaced by this modified one; if this file is obtained from a Web service by querying some XML database, then the users must guarantee the modifications in this file can be put back into the XML database, for instance, by preparing some special query for updating the database.

$$
\begin{array}{lll}
X & ::= & BX \mid XC \mid CM \\
BX & ::= & \texttt{<xid>}[] \mid \texttt{<xconst>}[S] \mid \texttt{<xchild>}[] \\
XC & ::= & \texttt{<xseq>}[X_1, ..., X_n] \mid \texttt{<xchcont>}[X_1, ..., X_n] \\
 & & \mid \texttt{<xmap>}[X] \mid \texttt{<xif>}[P, X_1, X_2] \\
CM & ::= & \texttt{<xstore>}[Var] \mid \texttt{<xload>}[Var] \\
 & & \mid \texttt{<xfree>}[Var] \\
P & ::= & \texttt{<xwithtag>}[str] \mid X
\end{array}
$$

Fig. 2. Syntax of the Underlying Language

3 Bi-X: A Bidirectional Transformation Language

The current popular XML transformation languages, such as XSLT [1] and XQuery [4], perform a transformation only in one direction. Transformations written in these languages cannot help to reflect modifications on the view back into the source data. In our XML transformation language Bi-X, while the code for a forward transformation is written in the similar way as in XSLT and XQuery, the code for the backward transformation is automatically derived. The derived code takes the modified target view and the original source as the inputs and generates the updated source as the output, propagating the modifications given on the view. In this section, we briefly summarize the basic of the language, and then discuss the properties of bidirectional transformations.

3.1 Bi-X Syntax and Basic Transformations

The syntax of a fragment of Bi-X is given in Figure 2, where each language construct is represented as an XML element. Basic transformations *BX* perform some particular transformations on source data. <xid>[] transforms the source data into the same target data. <xconst>[S] transforms any source data into the constant target data S. <xchild>[] accepts an element as source data, and returns its content.

Transformation combinators *XC* are used to build more complex transformations by gluing simpler transformations together. $<\texttt{xseq}>[]$ is composed transformation that applies its argument transformations $X_i (1 \leq i \leq n)$ in sequence, and the target data of the transformation X_i will be used as the source data of its successive transformation X_{i+1}. <xchcont>$[X_1, ..., X_n]$ accepts an element as source data, and returns this element with its contents replaced by the result of applying transformations $X_i (1 \leq i \leq n)$ to empty values. <xmap>$[X]$ transforms the sequence source data by applying X to each item in the sequence. <xif>$[P, X_1, X_2]$ applies X_1 to the source data if the predicate P holds over this source data, otherwise X_2 is applied.

The transformations *CM* are to manage or use the transformation context. They provide the variable binding mechanism for the Bi-X language. <xstore> [*Var*] binds the source data to the variable *Var*, which is valid until it is released by <xfree>[*Var*]. <xload>[*Var*] accesses the bound value of a valid variable.

```
<xseq>[
  <xstore>[$src],
  <xconst>[<html>[]],
  <xchcont>[
    <xseq>[
      <xconst>[<body>[]],
        <xchcont>[
        <xseq>[
          <xconst>[<ol>[]],
          <xchcont>[
            <xseq>[
              <xload>[$src],
              <xchild>[],
              <xmap>[
                <xseq>[
                  <xstore>[$var],
                  <xconst>[<li>[]],
                  <xchcont>[X  Y],
                  <xfree>[$var],
                ]]]]]]]],
  <xfree>[$src]
]
```

Fig. 3. Bi-X Program Example

```
<xseq>[
  <xconst>[<div>[]],
  <xchcont>[
    <xseq>[
      <xload>[$var],
      <xchild>[],
      <xmap>[
        <xif>[
          <xwithtag>[title],
          <xchild>[],
          <xconst>[],
        ]
      ]
    ]
  ]
]
```

Fig. 4. Bi-X Code for X

The predicate <xwithtag>$[str]$ holds of the source data is an element with tag *str*, and any transformation can be used as a predicate for <xif>$[P, X_1, X_2]$.

Using the Bi-X syntax, a Bi-X code for the transformation to make the view in the example in section 2 is given in Figure 3 and Figure 4. The code is divided into two parts for readability. The code for X in Figure 3 is given in Figure 4, that extracts the titles from the source data. The code for Y in Figure 3 similarly extracts the authors from the source data, and omitted here.

As you can see, codes of Bi-X generally tend to become long compared to the one that is written in other popular XML transformation languages. In order to reduce coding efforts, an XQuery interpreter has been provided with Bi-X language. Using this tool, the user can write only XQuery code for the forward transformation and automatically get an equivalent Bi-X code for the bidirectional transformation.

3.2 View Updating Property of Bi-X

In this section, the view updating property of Bi-X language is illustrated informally to help users to understand the results of backward transformations. That is, given an updated view, what the updated source document should be after backward transformation. For the brevity of presentation, the updates here only include modifications to the XML text contents and tags.

During a session of forward and backward transformation, there are two pairs of documents involved: the original source document and the source document

after updating it, and the original view and the updated view. Each pair of documents has same structure since we concern only modifications here. The property of Bi-X is defined on the differences between the original document and the updated document. The differences of two documents are represented as a multiset of pairs, and each pair consists of two different strings, which are either element tags or text contents. A pair represents a modification, that is, its first component is changed to the second one.

To represent modifications more precisely, tags and text contents in source documents are assigned unique identifiers, while tags and text contents in `xconst` are associated with a specific identifier, say `c`. Identifiers are kept unchanged during transforming source documents and modifying views. A modification is called a bad modification if it contains strings with the identifier `c`. This means data from the transformation code cannot be modified. Two string components in a good modification must have the same identifiers and no two good modifications on one document have the same identifiers. Two modifications are said equal if they make the same changes on the strings with the same identifiers.

We write $\texttt{diff}(od,md)$ for the differences between the original document od and its modified document md. For two documents with the same structure, their differences can be easily obtained by traversing the document structure, and comparing each tag and text content. The view updating property of Bi-X is described as follows.

Suppose sd is a source document, X a Bi-X transformation, td a target document of sd transformed by X, td' is obtained from td with only good modifications. Then the following condition holds after backward transformation of td' using X:

$$\texttt{diff}(sd, sd') = \texttt{diff}(td, td')$$

where sd' is the updated sd generated from the backward transformation.

Intuitively, this property says after backward transformations all modifications on views are reflected back to the corresponding tags or text contents in source documents.

4 Communication Protocol

The communication protocol among three components in the data updating process is illustrated by a diagram of the communication pattern in Figure 5. The protocol consists of two phases, that is *Init* phase and *Updating* phase. The Init phase and the Update phase are performed by invoking Init service and the Update service respectively, both of which are provided by Bi-X service. Between the Init phase and the Update phase, the user edits the view on the client. The steps of each phase are described as follows.

Init phase

Init(1) Clients send an Init message to the Bi-X server with two arguments: one is the URI1 for the source data to be transformed, and the other is the URI2 for the code.

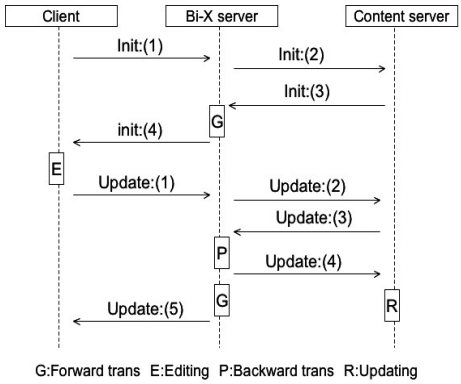

Fig. 5. Communication Pattern

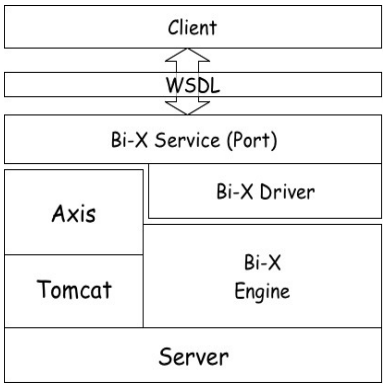

Fig. 6. Structure of Bi-X Service Implementation

Init(2) Bi-X server requests the files specified by URI1 and URI2 using HTTP Get method.

Init(3) Machines specidied in URI1 and URI2 process HTTP Get method and return the specified files.

Init(4) Bi-X server makes a forward transformation and sends the view to the client.

Updating phase

Update(1) Clients send an Update message to the Bi-X Server with three arguments: the first is the URI1 for the source data, the second is the URI2 for the code, and the thrid is the changed view.

Update(2) Bi-X server requests the source data to be updated and code according to URI1 and URI2 using HTTP Get method.

Update(3) Machines specidied in URI1 and URI2 process HTTP Get method and return the specified files.

Update(4) Bi-X server performs the backward transformation to get a updated source data, and sends the updated source data back to URI1 using HTTP POST method.

Update(5) Bi-X server performs the forward transformation using the updated source data, and sends the new view to the client.

5 Bi-X Service Implementation

Our Bi-X service used for the use case in Section 6 has been implemented in Java, using the standard web service technologies such as SOAP [2], REST [5] and WSDL [3]. The structure of the implementation of Bi-X server is shown in Figure 6.

Axis and Tomcat Our Bi-X service uses Axis2 [6] for SOAP and REST implementation, and is deployed on Axis2 running on servlet container Tomcat [7]. Since Bi-X service uses these standard softwares, what is needed for installation of Bi-X services is only registeration of an archive file of Bi-X service to the containers. Thus, users can easily install Bi-X service on their own machines.

Bi-X Driver Bi-X Driver wraps Bi-X engine Bi-X that is a Java implementation of bidirectional transformation language Bi-X. Bi-X Driver itself is also written in Java. The driver provides the engine with network communications with contents servers to transfer XML documents. These communications use HTTP GET message to get XML documents (i.e. sources and codes) from contents servers, and use HTTP POST message to put modified XML documents (i.e. new sources) to contents servers.

Bi-X Service Port and WSDL This part provides users with methods such as *Init* and *Update* available through the Internet to utilize the bidirectional transformations. Types of these methods and data structures of their arguments are provided within WSDL to users. Users can easily make their SOAP clients for these Bi-X service methods by putting the WSDL to auto program generators such as WSDL2Java of Axis, wsdl2ruby of SOAP4R [8], and so on. Also, users can use REST interfaces of these Bi-X service methods thanks to the power of Axis2. In this case, to use Bi-X service, users do not need anything but a method to access specified URLs.

6 An Example of Client and Contents Server

In our Bi-X service architecture, the client and the content server can be any systems that satisfy the requirements explained in Section 2. In this section, we give an example of the client and content server, which is used in our applications.

A Bi-X service client that calls methods provided by the server can be easily prepared using the standard Web service technologies. All necessary information for this can be obtained from the WSDL description of the Bi-X service. In the client program, we first create Service and Call objects. These are standard JAX-RPC objects. Next we set up the endpoint URL, which is destination of the SOAP message. We also define the method name of the web service, and give a statement to invoke the desired service, passing in an array of parameters. Another way to make a client program is to use WSDL2java tool included in Axis. It generates client stubs code for SOAP communication from WSDL description. The client simply uses the stub to invoke the web service as if it was a regular Java object in the same address space.

As our client for testing examples, we use Justsystem xfy [9], which is an "Integrated XML Application Development Environment" developed by Justsystem Corporation. An advantage of using xfy is its capability to handle various kinds of XML vocabularies in optimized and sophisticated way. For example, texts in XHTML vocabulary are directly editable on the xfy browser.

Our client program is incorporated with xfy to work as a part of an xfy plugin, so that request messages to the Bi-X server are built and sent through user's

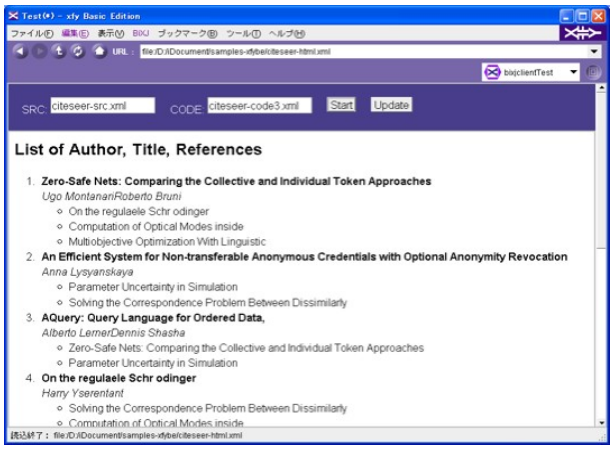

Fig. 7. A CiteSeer View on the xfy

action on the xfy's interface, and the result from the Bi-X server is displayed on xfy's browser.

The requirement to content server is that it can provide XML files, and can accept modified files. For example, we can use the eXist XML DB [10] to provide source data. In this case, when receiving a request for source data, the content server extracts the source data from the DB with XQuery, and sends them to the *transformation engine*, and afterward, when getting the updated source data, this server updates the DB according to the updated source data by executing some *updating queries* prepared by users. The XQuery in eXist extends the standard XQuery with some update statements that can be used to make *updating queries*.

7 A Use Case

We have tested our architecture by using several use cases. In this section, we show one of the test cases that uses CiteSeer [11] database to demonstrate the usability of our system. CiteSeer is a scientific literature digital library and search engine that focuses primarily on the literature in computer and information science. It crawls and harvests academic and scientific documents on the web and uses autonomous citation indexing to permit querying by citation. The web site of CiteSeer have correction pages to correct the information of a given document such as the title, abstract, summary and author(s). Any users can contribute to the correction of the source data through the form-based web interface, by editing these contents and submitting them. This kind of application is suitable to our view update system.

To demonstrate the view updating, we saved a part of the original XML data of the CiteSeer library and perform view updating with Bi-X server. Figure 7 is a snapshot of the view on the xfy window. The user provides URIs of the

source XML file and the Bi-X code to transform it. By pressing *Start* button that invokes Init service, the HTML view is generated by Bi-X code and displayed on xfy browser. In this example, view includes the list of the document information, and each of items has the title, author and list of the titles of the cited documents. Note that xfy widow allows to edit a text in the HTML view directly. The modifications made to texts are reflected to the source by pressing *Update* button, which invokes the Update service. Thus, the user can make a view that includes only interested contents in a suitable format by coding Bi-X code, and can edit some contents on the view to update the remote source XML data.

8 Related Work

The Bi-X language used in this work takes similar bidirectional transformation style as those work [12, 13]. These languages are designed for their particular purposes and have several limitations to be used as the general XML processing languages. Bi-X has addressed their limitations and thus can be used in this architecture for general-purpose XML processing.

Many XML update systems that use a database are closely connected to the database system and they are not easy to re-implement to work with a different system. Bi-X server is a generic tool for the XML update that can be easily connected to contents servers and web applications, and can be reused.

This architecture is similar to the traditional three-tier client-server architecture, where the user interface, the server for business logic and the data storage are developed and maintained as three independent components. The difference is that in our architecture the data storage is not a private back end of the transformation engine, and instead it is open and can be specified by users.

From the perspective of software maintenance, the architecture in this paper belongs to the group Software Service Providers (SSP) [14], that is, to provide the operational software component. We believe this is a trend for providing application software to users. Similar to our purpose, Google now provides spreadsheet software [15] as an application based on Web, and thus users do not need to install and maintain a local copy of such application.

9 Conclusion

In this paper, we proposed a Web service architecture for bidirectional updating. Our approach is based on view updating in the sense that users can update the source data by editing the target view that is generated by some transformation. In this architecture, users can exploit bidirectional transformation by telling the *transformation engine* their transformation code, source data and updated target data, and the transformation engine then produces the view or the updated source data for them. Due to the use of standard Web service technologies, the data viewer client and contents servers can be easily replaced by the ones chosen by the user.

References

1. W3C Draft. XSL Transformations (XSLT) Version 2.0 . http://www.w3.org/TR/xslt20/, 2005.
2. W3C. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/soap, 2000.
3. W3C. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.
4. W3C Draft. XML Query (XQuery) . http://www.w3.org/XML/Query, 2005.
5. Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. Chair-Richard N. Taylor.
6. Apache Software Foundation. Apache Axis2/Java. http://ws.apache.org/axis2/.
7. Apache Software Foundation. Apache Tomcat. http://tomcat.apache.org/.
8. Hiroshi Nakamura. SOAP4R. http://raa.ruby-lang.org/project/soap4r/.
9. Justsystem Corporation. xfy technology. http://www.xfytec.com.
10. Wolfgang Meier. eXist: Open Source Native XML Database. http://www.exist-db.org/.
11. College of Information Sciences and The Pennsylvania State University Technology. CiteSeer. http://citeseer.ist.psu.edu/.
12. J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2005.
13. Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. In *Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, 2004.
14. Keith H. Bennett and Jie Xu. Software services and software maintenance. In *7th European Conference on Software Maintenance and Reengineering*, pages 3–12, 2003.
15. Google. Google Spreadsheet. http://spreadsheets.google.com.