A Web Service Architecture for Bidirectional Updating

Yasushi Hayashi Dongxi Liu Kento Emoto
Kazutaka Matsuda Zhenjiang Hu Masato Takeichi

Department of Mathematical Informatics, University of Tokyo
{hayashi,liu,hu,takeichi}@mist.i.u-tokyo.ac.jp
{emoto,kztk}@ipl.t.u-tokyo.ac.jp

We propose a Web service architecture for achieving bidirectional updating. This architecture consists of three tiers: clients, a Bi-X bidirectional transformation engine, and some content servers accessible through the Internet. The benefit of using this architecture is that users can harness the power of bidirectional transformations without the burden of installing and maintaining the bidirectional language package. We give several examples to demonstrate the usefulness of this architecture. Users can use this architecture to implement their own applications.

1 Introduction

XML is widely used as the de facto standard format of data exchange or repository. XML documents often need to be transformed for different reasons. For example, an XML file is transformed into HTML format for displaying in Web browser, or it is transformed into another small XML file, in which only interesting data for users is contained. In some cases, the target XML documents are probably modified, and it is desirable that these modifications on transformed documents can be reflected back into source documents.

The current popular XML transformation languages, such as XSLT [1] and XQuery [2], perform transformation only in one direction. Transformations written in these languages cannot help to reflect modifications on target documents back into the source data. In order to maintain data consistency between the source and target data, some ad-hoc methods have to be taken. For example, one possible yet boring method is to update the source data directly by first looking for the values that are changed in the target data and then making same modifications on them. However, the situation changes if bidirecitonal languages, such as `Bi-X` [3], are exploited to transform XML documents.

A bidirectional transformation program can be executed in two directions: forward direction and backward direction. The forward transformation transforms a source XML document into a target document, while the backward transformation transforms the updated target document together with the original source document into the updated source document. After the backward transformation, the modifications in the target document will be reflected back to the updated source document. That is, modifications on the target data can be propagated to the source data automatically by backward executions of bidirectional language programs.

In this paper, we propose a Web service architecture to promote the use of bidirectional transformation in the network environment. By this architecture, we hope users can harness the power of bidirectional transformation without the burden of installing and maintaining the bidirectional language package. For example, users do not need to concern whether there are new versions or bug patches for the bidirectional languages.

The architecture proposed in this paper includes three parts. The first part is *data viewer clients*, which present target data to users. The data viewer clients are chosen by users. For example, they can be ordinary Web browsers or XML editors. The second part is a bidirectional transformation engine, which provides the bidiretional transformation service based on Bi-X language. This part is supposed to be independent of users' application. We have run and maintained this engine on the transformation server, and are planning to make it accessible publicly. The third part is *content servers*, which provide XML data for transformation. This part is specified by users when they launch transformations. The requirement to this part is that it can not only provide XML files, but also can accept modified files and then update the data making up these files.

We will provide several examples to help users get some intuitiveness of this architecture. The first example uses the Amazon Web Service (AWS) [4] as the content server. In this example, users can get the book information from Amazon and then transform it into a form they like using the *transformation engine*. Since Amazon does not allow to update their data source, the *transformation engine* reflects the modifications from users back into a locally cached copy of the data from Amazon for demonstration purpose. In the second example, we use the eXist XML DB [5] to provide XML data. And we give several examples of XQuery (and XUpdate) that can update the whole DB according to users' modifications.

The remainder of this paper is organized as follows. Section 2 gives a detailed explanation of the architecture. Section 3 explains our implementation of the architecture. Section 4 presents two use cases. Section 5 explains our implementation of the client. Section 6 discusses the related work and Section 7 concludes the paper.

2 Architecture

2.1 Three-tier Architectue

Our Web service architecture for bidirectional updating consists of three tiers as shown in Figure 1, that is, clients, a Bi-X server, and the content server that provides XML data. The heart of this architecture is the Bi-X server, which has a bidirectional transformation engine based on an implementation of our bidirectional transformation language Bi-X. It receives a request from clients and applies a forward transformation to the specified source data originally fetched from content servers, or applies

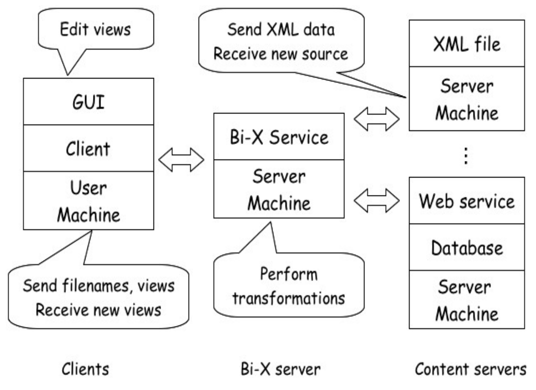


Figure 1: Three-tier Architecture

a backward transformation to produce the updated source data. Another role of the Bi-X server is to communicate with the content servers. It sends a request message to the content servers to get a Bi-X code and some XML data, and receives the retuned data, on which transformations are applied. It also sends update messages to reflect changes of the source data to content servers.

Clients of Bi-X services can be any XML data viewer, but typically web applications to display the target data of a Bi-X transformation in a formatted view equipped with some editing environment. They receive 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.

The content servers provide XML documents for transformations. This part is specified by users when they launch transformations. The requirement to this part 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 engine 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 DB, then the users must guarantee the modifications in this file can be put back into the XML DB, for instance, by preparing some special query for updating the DB.

In order to make it clear the roles of the Bi-X ser-

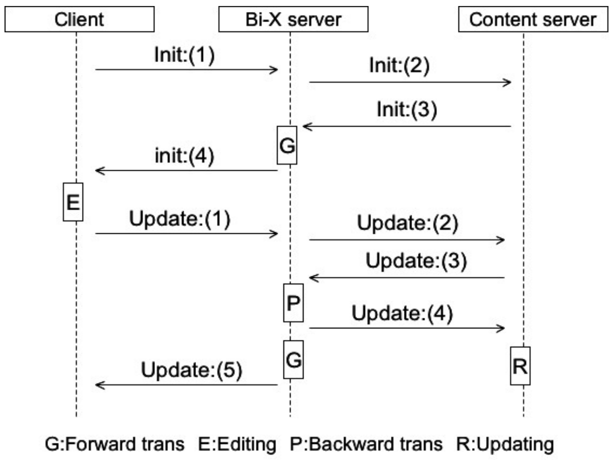


Figure 2: Communication Pattern

vice provider and the Bi-X service user within this web service architecture, we summarize the responsibility of each role.
Responsibility of user role:

1. Providing the code and the XML data to be transformed;
2. Determining how to use the updated data according to their private business rules.

Responsibility of provider role:

1. Performing a transformation based on the source data file and code file specified by users;
2. Sending back the updated source data (to the server from which the original soruce data comes).
3. Sending the new view to the data viewer client.

2.2 Communication Protocol

The communication protocol among three components in the data updating process is illustrated by a diagram of the communication pattern in Figure 2. The protocol consists of two phases, that is *Init* phase and *Updating* phase. The steps of each phase are described as follows.

Init phase

Step (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.

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

Step (3): Machines specidied in URI1 and URI2 process HTTP Get method and return the specified files. (Note that it is the responsibility of users to guarantee these machines exist and can work as expected.)

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

Step (5): Bi-X server discards the soruce data and the code after finishing this forward transformation task.

Updating phase

Step (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.

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

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

Step (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. The machine specidied in URI1 can process HTTP Post method and it should know the data in this HTTP Post message is the udpated source data. (Note that it is the responsibility of users to know the meaning of this HTTP Post method and determine how to use this updated source data according to their private business rules.)

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

Figure 3: Syntax of the Underlying Language

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

Step (6): Bi-X server discard all data.

2.3 Bi-X - A Bidirectional Transformation Language

Bi-X is a domain-specifc language for bidirectional XML transformation. In this section, we briefly introduce the Bi-X language. The formal definition of Bi-X and programming examples can be found in [6, 3]. The syntax of a fragment of Bi-X is given in Figure 3.

Basic transformations *BX* perform some particular transformations on source data: `xid` transforms the source data into the same target data; `xconst` 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 based on other transformations: `xseq` 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` 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` transforms the sequence source data by applying X to each item in the sequence; `xif` 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` binds the source data to the variable *Var*, which is valid until it is released by `xfree`; `xload` accesses the bound value of a valid variable.

The predicate `xwithtag` holds of the source data is an element with tag *str*, and any transformation can be used as a predicate for `xif`.

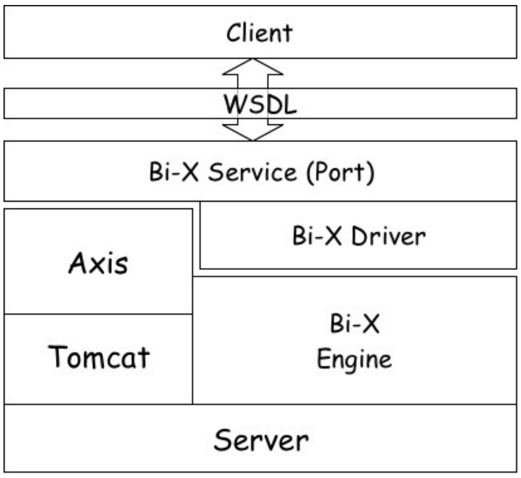


Figure 4: Structure of Bi-X Service Implementation

3 Implementation

Our Bi-X services used for the use cases in Section 4 has been implemented in Java, using the standard web service technologies such as SOAP, WSDL. The structure of the implementation of Bi-X server is shown in Figure 4. They are provided on Tomcat (a servlet container) with Axis (SOAP implementation), and Axis2 (SOAP and REST implementation). They use a Java implementation of bidirectional transformation language Bi-X, which are wrapped by *Bi-X Driver* to perform bidirectional transformations. For data viewer client, we first tested on a simple tree viewer developed in Java and next used Justsystem xfy [7] to develop more sophisticated applications. The client by xfy is explained in Section 5.

Apart from use cases mentioned in this paper, our group has been developing an application tool for Web page updating based on bidirectonal transformation, in which our implementation of Bi-X server is used as its transformation engine. This work is reported in [8].

4 Use Cases

4.1 Amazon Web Service

The first use case makes use of Amazon Web Service (AWS) provided by Amazon as the content server. Users can get the book information from Amazon and then transform it into a form they like using the Bi-X transformation engine. Since Amazon does not allow to update their data source, the Bi-X engine reflects the modifications from users back into a locally cached copy of the source data in the Bi-X server for demonstration purpose. Here, we explain how to use AWS with the Bi-X transformation service.

In the Init phase, the URI1 parameter of Init message sent by the client specifies URI for requesting a REST service provided by AWS. For example, this could be AWS search with a given category and keywords like

```
http://webservices.amazon.com/onca/xml?Service=
AWSECommerceService&Version=2005-07-26&Response
Group=Medium&Operation=ItemSearch&SubscriptionI
d=************&SearchIndex=Books&Keywords=ajax
```

with 'Books' as the category and 'ajax' as the keyword. When the Bi-X server receives an Init message, it accesses AWS by REST using the URI1. AWS performs the search and sends the resulting XML data that includes the book information to the Bi-X server. The Bi-X engine transforms it into a form according to the Bi-X code specified as the URI2 parameter in Int message, and then send the result to client to display. Since AWS does not allow to update their data source, even if the Bi-X server sends the updated source obtained by an update transformation, AWS will discard it. As a temporally solution for demonstration purpose, Bi-X server can provide a local cache for keeping updated data source that reflects user's modifications. Instead, we can use an XML database in place of AWS as a content server that already stored the data from AWS to demonstrate our system can update the date in the database of the content server using the technique explained in the next subsection.

4.2 Using XML DB as Content Server

In this section, we give another use case, where the content server uses the eXist XML DB [5] 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*. Users must prepare an *updating query* for each XQuery expression that can be used to serve source data.

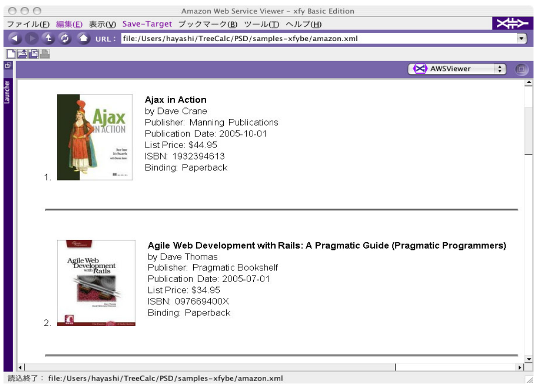
We will illustrate *updating queries* by an example. Suppose the content server provides source data using the following XQuery expression:

```
<bib>
 {
  for $b at $i in doc("/db/bib.xml")/bib/book
    where ($b/publisher = "Addison-Wesley"
      and $b/@year > 1991) return
    <book newyear="{ $b/@year }" index="{$i}">
     { $b/title }
    </book>
 }
</bib>
```

Then, an *updating query* for the above expression is defined as follows.

```
let $a1 := doc("query-result.xml")/bib return
  <bib>{
    for $b at $i in doc("/db/bib.xml")/bib/book
      where($b/publisher = "Addison-Wesley"
                 and $b/@year > 1991)
      return
        let $a2 := $a1/book return
          for $ub in $a2[@index = $i] return
            <book newyear="{ $b/@year }"
                          index="{$i}">
              {update value $b/@year
                          with $ub/@newyear,
               update replace $b/title
                          with $ub/title,
              $b/title}
           </book>}
  </bib>
```

In this *updating query*, the file "query-result.xml" contains the updated source data from the *transformation engine*.


Amazon Web Service Viewer - xfy Basic Edition
ファイル(F) 編集(E) 表示(V) Save-Target ブックマーク(B) ツール(T) ヘルプ(H)
URL: file:/Users/hayashi/TreeCalc/PSD/samples-xfybe/amazon.xml
AWSViewer
Launcher
読込終了： file:/Users/hayashi/TreeCalc/PSD/samples-xfybe/amazon.xml

1. **Ajax in Action**
by Dave Crane
Publisher: Manning Publications
Publication Date: 2005-10-01
List Price: $44.95
ISBN: 1932394613
Binding: Paperback

2. **Agile Web Development with Rails: A Pragmatic Guide (Pragmatic Programmers)**
by Dave Thomas
Publisher: Pragmatic Bookshelf
Publication Date: 2005-07-01
List Price: $34.95
ISBN: 097669400X
Binding: Paperback

Figure 5: A Snapshot of an xfy Browser

5 Client by xfy

In our Bi-X service architecture, the client of Bi-X services can be any XML viewer/editor. However, it would be a typically web application, which has a capability of displaying the target document as a formated view, and provides an editing environment to edit the target document through the view. For our implementation, we use Justsystem xfy [7], 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, including user-defined one in optimized and sophisticated way. This enhances the usability of the client applications that use our bidirectional transformation service, because it can display various kinds of target XML documents in the way that users can easily understand the meaning of data, and also provides the editing environment for changing data optimized for the characteristics of each vocabulary. For example, the texts in XHTML vocabulary can be directly editable on the xfy browser, hence achieving WYSWYG style of editing.

xfy can also customize its user interface such as menu and buttons to invoke some operations for using bidirectional transformation services by defining it in the script language called XVCD. Our client program using Axis is incorporated 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 action on the xfy's interface, and the result from the Bi-XJ server is displayed on the xfy's browser. A snapshot of an xfy application used for the AWS use case in 4.1 is shown in Figure 5. Currently, the update operation is invoked when the user selects the update command from the menu on the xfy window.

6 Related Work

The Bi-X language used in this work takes similar bidirectional transformation style as those work [9, 10]. These languages are designed for their particular purposes. As discussed in work [3], they 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.

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) [11], 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 [12] as an application based on Web, and thus users do not need to install and maintain a local copy of such application.

7 Conclusion

In this paper, we propose a Web service architecture for bidirectional updating. Our purpose is to promote the use of bidirectional transformation. In this architecture, users can exploit bidirectional transformation by telling the *transformation engine* their transformation code, source data and updated target data, and *transformation engine* then produces the target data or the updated source data for them. The benefit is that users need not to install and maintain bidirectional language package.

Acknoledgement

We would like to thank Justsystem Corporation for providing us with some helpful technical information about xfy. This research is supported by the *Comprehensive Development of e-Society Foundation Software* of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] W3C Draft. XSL Transformations (XSLT) Version 2.0 . http://www.w3.org/TR/xslt20/, 2005.

[2] W3C Draft. XML Query (XQuery) . http://www.w3.org/XML/Query, 2005.

[3] Dongxi Liu, Zhenjiang Hu, Masato Takeichi, Kazuhiko Kakehi, and Hao Wang. A Java library for bidirectional XML transformation. In *The 22nd Conference of Japan Society for Software Science and Technology*, 2005.

[4] Amazon. Amazon E-Commerce Service. http://aws.amazon.com.

[5] Wolfgang Meier. eXist: Open Source Native XML Database. http://www.exist-db.org/.

[6] Dongxi Liu, Zhenjiang Hu, and Masato Takeichi. Bidirectionalizing XQuery – Updating XML through Materialized XQuery View . Technique Report METR 2006-25, Department of Mathematical Informatics, University of Tokyo, 2006.

[7] Justsystem Corporation. xfy technology. http://www.xfytec.com.

[8] Keisuke Nakano, Akimasa Morihata, Zhenjiang Hu, and Masato Takeichi. Web page updating based on bidirectional transformation. In *The 23rd Conference of Japan Society for Software Science and Technology*, 2006.

[9] 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.

[10] 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.

[11] Keith H. Bennett and Jie Xu. Software services and software maintenance. In *7th European Conference on Software Maintenance and Reengineering*, pages 3–12, 2003.

[12] Google. Google Spreadsheet. http://spreadsheets.google.com.