XSync – Infrastructure for Native XML Data Synchronization.

Deepak Srinivasa, Snehit Prabhu
Technology Incubation Centre,
IBM Software Labs India

Introduction

XML is a language to structure information. Unlike text documents that have no structure, XML can impose a set a of user defined rules (using structure definitions like DTDs or XML Schemas) on the information stored, and is being widely used to maintain vast amounts of such information. This very same structure that is imposed on XML data, makes it easier to process, read, query and modify XML data while maintaining structure within a context. As a consequence, large volumes of information are being stored in the form of native XML rather than relational databases like DB2 or Oracle, forming what we term as "XML Repositories". The existence of a single repository, replicated over independent machines in a distributed environment, as is often the case, prompts the need for a synchronization framework that will help maintain consistency across these instances.

This project is aimed at synchronizing native XML data that exists independently on 2 or more machines in a distributed environment. We allow each entity to perform update operations on the data existing on their local repository images, and provide a mechanism to permeate the changes performed at each end to the other. At the end of the synchronization operation, the images of the repositories on both the entities are the same.

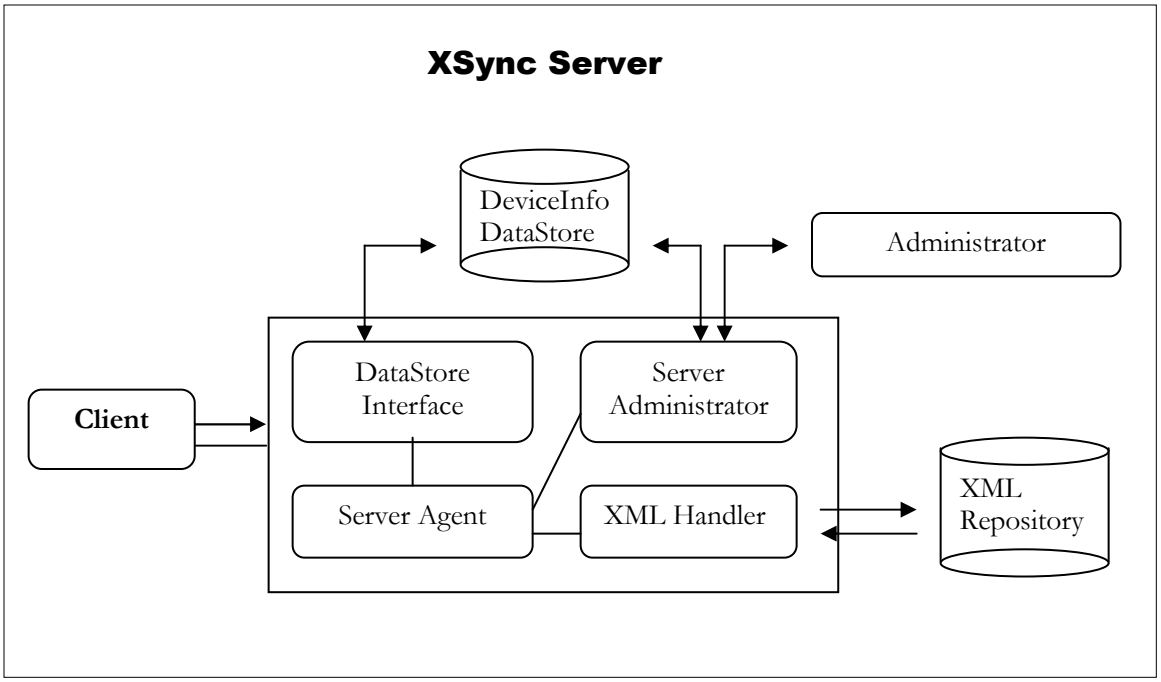
The SyncML[2] open synchronization protocol, functions for the expert purpose of synchronizing 2 or more database instances, on multiple devices. It works on the client-server architecture, and has been implemented to address the need presented by the above mentioned problem.

Synchronization Infrastructure

The Synchronization Markup Language (SyncML) protocol initiative was taken by IBM, Lotus, Nokia, Motorola and others, and resulted in the full specification of a protocol that has become the *industry standard for Database Synchronization.*The SyncML protocol defines the format of messages to be exchanged between the devices involved in the synchronization.

The synchronization operation is carried out between 2 sync agents : One the client, and the other the server. The client usually initiates the synch and sends the server its set of modifications since the last synchronization savepoint. The server handles these operations, resolves the modifications with its own using the sync engine, and sends back the client the changes made to its own image since the last savepoint.

The XSync SyncML Server and Client agents were developed using servlets running on a conventional server. When the Client sends its packages containing modifications to the XML file at its end, the SyncML Servlet is called up.



The Server agent decodes the client's packages and passes the sync operations to the XML database handler (which we will deal with in detail later). During the decoding of the client's message,the Server agent can access state information about the client in the data store, through the data store interface. This interface allows the Server agent to access information regarding the capabilities of the client, the local to global identifier mappings from the client to server repositories (used in case the client does not replicate the entire XML file locally, but only a view/subset of it) and other such information. The Server Administrator accesses the data store directly and handles the creation and maintainance of client XML databases that are registered with the server for synchronization.
The XML Clients maintain additional functionality like conflict resolution (in case of conflicting modifications made at the 2 ends) and the maintainance of *change logs*. We will deal with this in detail when we address the XML Handler separately.

XUpdate Module

Treating native XML files as a set of data stores, calls for providing a convenient set of APIs that give them all the functionality that traditional data stores should have. While there are several Object Model frameworks in place that provide API's to read, query, and write to XML files, none of these were found to be convenient enough to provide "single call" operations on the files. Most of these frameworks require the programmer to use the given set of API's in combinations with one another to effect changes (parse the file, write logic to retrieve xml fragments, for writes : make the modifications into the DOM tree in memory, then persist the tree back onto disk by using java constructs like writers).
The XUpdate module is a library that works on DOM4J[1], and provides a set of convenient write APIs to the end user, that permit him to effect changes to the XML document instantaneously. This module, when integrated with the XML file repositories, provides easy update/modification faclilties to them on the same lines as updating a relational database table, or a Java object. Both the human client making modifications to his local repository, as well as the XSync Server and Client modules

that require to make structural changes to their files during a sync operation, will use this module to obtain write access to the XML repositories.
An obvious fallout of providing such specific easy access to XML (parse, read, modify, and write to disk in one atomic step), means restriciting the library`s range of functionality to far less that other DOM APIs provide, but it suitably addresses all the requirements posed in this (and hopefully other such) context. Currently the library provides write capability only for elements, attributes and text data content. Other XML objects like processing instructions, comments, etc. have not been write enabled, since these do not usually represent data that represents required business information, though the library is being extended to do this. Sample code illustrating the simple use of the library is given below . For more information please refer to [3].

```
/* Here root directory is the path to the base directory that represents the XML repository.*/
XUpdateInterface xmlHandler = new XMLUpdateHandler(String
rootDirectory);

/*The relative path to the file within the repository, and the xpath query to the node set to be
modified. Returned is an interface to a node set implementation. */
XObjectUpdateInterface xobject =
xmlHandler.getXMLObjectsForUpdate(String relativeFilePath,String
xPath);

/* Start performing update operations on the node set. Each call is atomic, and control will be
returned only after the change to all the nodes in the set has been persisted onto the disk as well. */
xobject.deleteChild(String childName);
xobject.deleteAttribute(String attrName);
      etc.
```

XML Synchronization

An XML Handler is a structure maintained by each XML Repository that needs to be synchronized. They encapsulate a whole range of functionality like maintaining modification lists (or change logs) for the repository since the time of the previous sync, and implementing conflict resolution schemes.

The XML Handler module is the interface between the SyncML server and the database that it is trying to synchronize. Handlers present the XSync Server and Client agents the necessary interfaces to retrieve all the "units of change" to the repository image (at one end) that need to be sent to the image (at the other end of the synchronization framework) .
The Handler is responsible for passing the client repository's modifications, on request, to the Client Agent, which sends these as a list of mod-elements to the Server Agent. The Server Agent in turn, parses the list of mod elements and passes them on to its corresponding repository's Handler. Here the mod elements are implemented on the server's repository image, or in the case of a conflict, they are resolved against the particular conflict resolution scheme that the Handler is implementing.

Modification lists, may simply be a trace log of the XUpdate activities performed on the XML Repository, with each entry being stored as an XML fragment of a predefined structure, in a log file.

Conflict Resolution, unlike modification list maintainence however, is a more involved process.
For example, the client might intimate the Server of an insert on a particular node in its local repository. The server however, may have deleted the very same node from its image of the repository. The decision on whether the node will be reinserted on the server end, or a delete of the node (and its modification) at the client end, or some third alternative, is governed by the Conflict

Resolution Scheme that the Handlers on either end implement (and they must agree upon this, to ensure consistency). Timestamps on the modifications prove quite useless in this case because of the distributed environment.

Another unique difficulty presented by this problem, was because of the semi-strucutred nature of native XML data. Data identification across repositories, as a consequence, is not as trivial an activity as it sounds. The identification of XML nodes is done without enforcing the use of id attributes on the XML files that are to be synchronized, or using xpath queries to identify a node (since more than one xpath query can result in the same node).

Status and Plan

The XSync project deals with several modules, most of which are wither complete, or a considerable way into development stages. A few modules still need to be implemented.
The SyncML Synchronization framework, involving the XSync Server and Client agents has been implemented. The interfaces presented by the framework will be implemented on the repository classes to be synchronized.
The XUpdate library that will provide the write modification to the XML Repositories is complete, though additional functionality like validated modification is being added. Currently the module does not discriminate between schema validated and invalid updates to the XML files, but assumes all user modifications are legal. Also extended support for processing instructions, comments, and other XML constructs is to be added.
The design aspects of the XML handler like modlist maintainence and conflict resolution schemes (a trivial one like "server always wins" scheme will be implemented to start with) and node identification still need to be finalized from the list of options available.

References

[1]. DOM4J Open Source XML Framework for Java. http://www.dom4j.org
[2]. SyncML Synchronization Protocol v1.1 http://www.syncml.org
[3]. XUpdate Library v1.0 http://speechbuilder.in.ibm.com/w3/w3TIC.nsf/pages/xupdate