Developing an XML Database in AgentTeamwork

Intermediate Report
Spring 2006

Cuong Ngo
Professor Munehiro Fukuda
June 2, 2006
This intermediate report is based on work accomplished during the period of March 27, 2006 to June 9, 2006. The tasks finished and difficulties of each phase will be detailed below.

**Phase 0: Understand current database, existing code and interworkings of AgentTeamwork.**

The current database, eXist, was used by AgentTeamwork, but there were issues with it. Because there was little documentation, it was hard to configure and use. It took awhile to understand how use and it took reading the available documentation and installing eXist on my home desktop computer to understand it completely. AgentTeamwork was a little easier to understand how the agents communicated with each other and what messages was sent from one agent to the next.

**Phase 1: Database specifications & implementation and creation of Applet-based GUI to.**

The database I created is called **XDBase** and it uses sockets to communicate with **XCollection**, the database interface class. It allows you to:

- create and delete a collection
- store and delete a resource document
- query for information such as IP addresses
- update resource document(s)
- synchronize by writing back to the local disk
- shutdown the entire database

The database uses DOM object nodes to store the XML resource documents. An issue arose when using Java version 1.4 as the DOM object implemented in this version (1.4 uses the Crimson Project from the Apache group) was not serialized and was harder to traverse and manipulate. Java version 1.5 uses the Xerces Project from Apache and the DOM object is much more flexible. It allows the DOM object to be serialized to pass the object
between sockets and it is easier to manipulate in order to get the required information needed. Once Professor Fukuda and I agreed upon using Java version 1.5 for my implementation, the database was much easier to design and implement.

I followed a similar design that was quite popular among XML databases. I created a **Collection** class that holds an unlimited number of **Resource** objects that represents the XML document. The **Resource** class contains the actual XML document store in a DOM object and the date of its last modification. To allow for easier extensibility, I separated out the interactions with the database into separate **Service** classes. The **DatabaseManagementService** lets the users create a **Collection**, synchronize the database by writing back to the local disk and shutdown the database. The **StoreService** class lets the user store a resource document and a whole collection. The **DeletionService** class enables the user to delete a resource document and an entire collection. The **RetrievalService** lets the user retrieve an XML document or an entire **Collection**, the last modification of a resource document and a resource list for a particular **Collection**. The **QueryService** enables the user to query the database for IP addresses using Xpath expressions and update the database. If other services are needed later, it would only take adding another service class with the function it requires. The **XDBase** class receives request from the various service classes though socket connections, parses the request and then performs the required task.

The Applet-based GUI was created to check the correct functionality of the database. It was rather hard, in the beginning, to figure out how to understand and use Java Swing components. It was much different than what I was used to while coding in C++ and C# using MFC and Windows Form, respectively. Using Microsoft Visual Studio.NET allowed for dragging and dropping of GUI components into the form, without using any layout managers. Borland JBuilder did offer a similar interface but it wasn’t working correctly. In the end I eventually looked at similar examples for what I wanted the GUI to look like and used trial and error to create the graphic interface.
The GUI allows the user some similar functionality found in **XCollection**. The **XDBaseGUI** lets the user connect to the database, view the collections and their available resource documents, view the entire XML resource document, store and delete resource documents.

The database had to be changed a little to accommodate the addition of a graphic interface because the applet-based GUI had to be informed of changes of the database. At first, the database operated on one thread that basically waited for a connection and then parses and executes the request. Now, the database has two threads, one that does what is stated above and another that monitors the state of the database and informs the GUI of changes. If the database is changed in anyway (the database is dirty), the GUI is updated with the new data.

**Phase 2: Implement new XCollection, port ResourceAgent and verify interaction between CommanderAgent and ResourceAgent.**

The new **XCollection** class is much simpler than the previous due to using the new database. Most of the methods in **XCollection** are in the form of creating the correct service object and then calling the correct method of that service.

The **ResourceAgent** class was also changed a little bit. Previously, to update the remote node status, each element was updated one at a time and this caused some unnecessary database connections. It has been changed to update all the elements of the node at one connection. In other words, the database is passed the name of the collection, the name of the resource document, a string array of elements to change in the form of Xpath expressions and a string array of the corresponding new values.

Another feature added is a FTP timestamp stored in a resource document. To accommodate this feature, before any connection is made to the FTP, the resource agent checks the FTP timestamp of the last access to the
FTP server and checks to see if it has been over 24 hours. If it has been over 24 hours, then the resource agent will connect to the FTP server and download all the XML resource files.

**Work In Progress**

The next step is to work with Emory to see what kind of formatted response his updated CommanderAgent needs from the ResourceAgent. From there, we will verify that the commander agent can receive a list of resources from the resource agent.

Another feature that still needs to be implemented is the pinging of remote nodes. The remote nodes will only need to be pinged once, when the resource document that corresponds to the remote node is first uploaded to the local database. This will be done in the ResourceAgent.