Master Project Report

An Object Oriented Database for Network Management Data

by

Qifeng Zhang

Department of Computer Science

Rennselaer Polytechnic Institute

May 25, 1999

1. INTRODUCTION

The goal of this project is to provide persistent storage for the Distributed Online Object Repositories (DOOR) project. The DOOR collects real time data as requested by its clients. Historical data is also useful, for example, in diagnosing network health or determining traffic pattern. Some clients may request data of the past or current data one batch at a time, so we need a database to store historical data for some period of time.

This project builds a small object oriented database in Java using ObjectStore PsePro for Java. The database should be small enough to be distributed with DOOR, it should also require little CPU time and memory to avoid putting too much burden on the host computer.

2. SELECTING A DATABASE MANAGEMENT SYSTEM

In choosing a Database Management System (DBMS), several factors have to be considered: size, time, platform dependence. This project is intended to be incorporated into the DOOR project [1]. The DOOR project uses Java as its primary development language, to make the code portable and to provide support for mobile agents. The DOOR project is developed to be platform independent, so this supporting database must also be platform independent. As a result Java is the language of our choice.

Relational DBMS provides robust and reliable data storage, but mapping Java objects into rows and columns slows performance and increases the amount of code to be written. A relational DBMS provides all the functionality of an advanced database system, but usually has a size in the tens of megabytes range. However, for our purpose, we need only simple storage and retrieval, so we don’t need full DBMS functionality. If DOOR contains a DBMS of size in the tens of megabytes range, its distribution and installation would be greatly hampered.

PsePro for Java is pure Java, pure object database for embedded database and mobile computing applications. PsePro has a small footprint (as small as 300K) so it can be easily packaged with applications, even with applets. Java provides object serialization, which is a simple yet

extensible mechanism for storing objects persistently. Serialization enables one to flatten objects into a stream of bytes that when later read, can recreate objects equivalent to those that were serialized. When the byte stream is a couple of megabytes in size, storing objects via serialization is slow, because serialization has to read/write the entire graph of objects at a time.

ObjectStore PsePro combines the simplicity of object serialization and the reliability of a database management system, together with in-memory-like performance for accessing persistent objects. PsePro improves on serialization, as evident in several key areas: improved performance for large number of objects, reliable object management and query. In addition, PsePro allows access to as little as a single object at a time.

For the forth mentioned reasons, PsePro is our choice of DBMS.

3. DESIGN OF DATABASE

The Management Information Base (MIB) of Simple Network Management Protocol (SNMP) [2] is organized as a deep and narrow tree. Such organization is to group related variables together and to support a well defined naming convention. Although MIB can be easily mapped onto object hierarchy, for a database, a deep and narrow tree would hurt its performance, a wide and shallow tree is more suitable.

The database consists of a RouterManager class, a Router class, a SNMPVariable class, a Time-Value class and a SNMPDatabaseException class. These classes are analogous to tables defined in a relational database. Each class contain data members that identify this class, which are analogous to keys in a table. Each class also contains references (pointers) to other classes, which are analogous to foreign keys.

There is no inheritance relationship among these classes, rather they have a indirect containment relationship. A top down relationship is RouterManager --> Router --> SNMPVariable --> Time-Value, each class on the left manages a set of objects of the class on the right. Please notice that

the RouterManager contains a collection of Router objects, not directly the Router objects. The same goes for the other two relationships.

Objects of Router class, SNMPVariable class and TimeValue class will be stored in the database, so these three classes need to be annotated by the PsePro class file postprocessor to make them persistent capable.

The ovals denote classes and their corresponding objects. The rectangulars denote collections. The arrows denote a relationship of containment.

There is also a TestDriver class. This class is not part of the database. It provides a simple commandline environment for testing and demonstrating the various functionality of the database. It can also serve as an example on how to use the methods provided by the other classes.

The RouterManager class has two collections. One is the allRouters collection. All the routers managed by this repository will be put into this collection. A particular router can be accessed by its name. The other is the allSNMPVariables collection. This collection stores all the SNMP variables of the routers in the allRouter collections. These two collections are top level accessible.

The Router class contains a collection of SNMPVariable objects. Each SNMP variable is accessible by its name under a particular router. Although in a MIB, not all SNMP variables are at the same level of the tree, in the database all SNMP variables are promoted to be just under the Router for fast performance.

The SNMPVariable class contains a VectorList. This VectorList stores the TimeValue objects, ordered by their time stamps. The TimeValue has two data members, one is the time stamp, the other is the value of the SNMP variable.

4. DATABASE RELATED METHODS OF THE CLASSES

A few methods of importance will be described here. These methods are the ones that perform database operations. Only methods in the RouterManager class are publicly accessible, all other methods are internal to this implementation and are marked with "*".

4.1 Opening and closing a database

(1). RouterManager.initialize(String dbName)

This method does the following operations:

- create a session
- create or open a database
- start a transaction

Before you can create and manipulate persistent objects with PsePro, a session must be created. A session is the context in which PsePro databases are created or opened, and in which transactions are executed. Only one transaction at a time can exist in a session.

(2). RouterManager.shutdown()

This method closes the database, invokes the PsePro Garbage Collector to clean up the persistent objects and terminate the session gracefully. The Java VM garbage collection only clean up after transient objects.

4.2 Storing data

(1). RouterManager.putData(String routerName, String[] snmpVariableNames, Vector data)
**Router.putData(String[] snmpVariableNames, Vector data)*

Since there are multiple Routers under one RouterManager, so a RouterManager uses the parameter "*routerName*" to identify one of its Routers and then invokes the method *putData()* of that Router.

The method *Router.putData()* pairs up the time stamps and values in "*data*", and stored the pairs under the corresponding SNMP variables.

Parameters:

routerName -- the name of the router to receive data.

snmpVariableNames -- an array of strings. Each string is a name for one SNMP variable.

data -- a vector of string arrays. Each element of the Vector is a string array. The first element of each string array is a time stamp, followed by one value for each variable in "*snmpVariableNames*". The values are positionally matched with the variables.

4.3 Retrieving data

(1). RouterManager.getDataBetween(String routerName, String[] snmpVariableNames, long startTime, long endTime)

**Router.getDataBetween(String[] snmpVariableNames, long startTime, long endTime)*

**SNMPVariable.getDataBetween(long startTime, long endTime)*

These three methods retrieve data between the time interval specified, startTime is inclusive, endTime is exclusive. A RouterManager uses the parameter "*routerName*" to identify one of its Routers and then invokes its *getDataBetween()* with the remaining three parameters. A router uses each of the names in the parameter "*snmpVariableNames*" to identify an SNMP-Variable and then invokes its *getDataBetween()* with the remaining two parameters. If there are multiple SNMP variables specified in *snmpVariableNames*, *Router.getDataBetween()* will take care of combining the data together. In the current implementation, if data returned by different SNMP variables can not match up, error will be reported.

The return type of all three functions is Vector, which has the same format as the Vector passed to the *putData()* methods.

Parameters:

startTime -- time since epoch in milliseconds.

endTime -- time since epoch in milliseconds.

routerName -- the name of the router to get data from.

snmpVariableNames -- an array of strings. Each string is a name for one SNMP variable.

(2). RouterManager.getDataBefore(String routerName, String[] snmpVariableNames, long endTime)

**Router.getDataBefore(String[] snmpVariableNames, long endTime)*

**SNMPVariable.getDataBefore(long endTime)*

These methods retrieve data before the endTime specified, endTime is inclusive. Other specifications are the same as the corresponding *getDataBetween()*.

(3). RouterManager.getDataAfter(String routerName, String[] snmpVariableNames, long startTime)

**Router.getDataAfter(String[] snmpVariableNames, long startTime)*

**SNMPVariable.getDataAfter(long startTime)*

These methods retrieve data after the startTime specified, startTime is inclusive. Other specifications are the same as the corresponding *getDataBetween()*.

4.4 Removing data

RouterManager.removeDataBetween()

RouterManager.removeDataBefore()

RouterManager.removeDataAfter()

**Router.removeDataBetween()*

**Router.removeDataBefore()*

**Router.removeDataAfter()*

**SNMPVariable.removeDataBetween()*

**SNMPVariable.removeDataBefore()*

**SNMPVariable.removeDataAfter()*

These nine methods have the same specifications as the *getData()* functions, except that these function return void. And since data to be removed don't have to pair up, no error checking is done. Everything that falls into the time interval specified will be removed. These methods only remove the references, the actual clean up is done by the PsePro persistent object garbage collector.

5. LIMITATIONS AND ASSUMPTIONS

5.1 Limitation of ObjectStore PSE Pro for Java Release 3.0

PsePro provides a mechanism for querying its COM.odi.util.Collection objects. A query applies a predicate expression to all the elements in a collection. The query returns a subset collection that contains all elements that satisfy the predicate.

The result returned by PsePro query is not ordered in any fashion, even if the underlying collection is ordered. Ordering in SQL query can be easily achieved by specifying descending or ascending order on a particular field, because numerical or alphabetical order is well established and understood. Objects are composition of multiple fields, so to determine its ordering we need to implement some method to compare them. In this release, PsePro query doesn't take any such method as its query input or allow us to specify ordering based on a data member. One other limiting factor is that query results are stored into a collection for generic objects, this collection for generic object is the only return type of PsePro query. In this process, any prior ordering information will be lost again. For our purpose, ordering the data according to its time stamp is of utmost importance, so we have to work around this.

Other limitations are that PsePro doesn't provide operations comparable to joining or projecting in SQL. Although relational database and object oriented database are very different, some of the basic operations are needed in both.

5.2 Assumptions in database implementation

Since data returned by the routers are naturally ordered in ascending order according to their time stamps, the following assumptions are made upon this:

- Data passed to the database are already in ascending order by their time stamps
- Data already in the database are older than those that are going to be inserted

Under these two assumptions, data in the database and data returned by queries will be sorted in ascending order by their time stamps.

In future release of ObjectStore PsePro, we can expect it to provide more query functionality to overcome its current limitations.

6. SETTING UP THE EVIRONMENT

This section will briefly discuss how to install PsePro on UNIX systems. For more information, please read the README.htm file in the top level directory where PsePro is located.

6.1. Setting up the CLASSPATH

PsePro requires certain entries in the CLASSPATH environment variable to run PsePro applications and to develop PsePro applications.

To use PsePro, the following entries must be added to CLASSPATH:

- The pro.zip file. This allows the Java VM to locate PsePro.
- The PsePro tools.zip file. This allows the Java VM to locate PsePro files for the Class File Postprocessor and other PsePro database tools.

These zip files must be explicitly listed in the CLASSPATH, and not in the usual way of listing the directory that contains them.

To develop and run PsePro applications, certain entries are required in the CLASSPATH variable to allow Java and PsePro to find

- Source directory
- Annotated class file directory

In order to make a class to be persistence-capable or persistence-aware, PsePro must postprocess the class file. PsePro takes these “.class” files and produces new “.class” files that contain annotations that are required for persistence.

The annotated files can be placed in the same directory as the source files or in a separate directory. If the annotated files are placed into a separate directory, the location of the annotated class files must precede the location of the original class files in the CLASSPATH. This is extremely

important, because it allows Java VM to find the annotated class files before it finds the original class files.

6.2. Compiling and postprocessing

PsePro applications are compiled the same way as any other java application. Just issue “javac <filename.java>” to produce the byte code files, i.e. the files with “.class” extension. For the classes that are to be persistence-capable, the class file postprocessor should be used to annotate them. To run the program, the annotated class files must be used and not the original class files. Complete instruction about using the postprocessor is in ObjectStore PsePro Java API User Guide. One example of postprocessing in place is:

```
osjcfp -inplace -dest . <filename.class>
```

When the option ‘-inplace’ is specified, it means that the annotated files will overwrite the original class files, which implicitly is in the current directory. Yet the option ‘-dest .’ is still required to specify that the destination is the current directory.

7. ACKNOWLEDGEMENTS

I would like to thank Professor Szymanski for his continuous guidance, helpful discussions and constructive advice. I also appreciate this precious opportunity so I could have the chance to learn and use an Object Oriented DBMS.

8. REFERENCES

[1]. A. Bivens, L. Gao, M.F. Hulber and B. K. Szymanski. *Agent-Based Network Monitoring*, Proc. Autonomous Agents99 Conference, Workshop 1, Agent Based High Performance Computing: Problem Solving Applications and Practical Deployment, Seattle, WA, May 1999, pp. 41-53.

[2]. B. Forouzan. *Introduction to data communications and networking*. McGraw-Hill, 1998.