
DE-FG03-01ER83312

Principal Investigator Svetlana G. Shasharina

April 5, 2002
Table of Contents

Introduction ................................................................................................................................3
Task 1. WISE Specifications ......................................................................................................3
Task 2. WISE API and Examples of Implementation .................................................................4
  BaBar Framework and BdbCopyJob package ........................................................................4
  WISE API for BdbCopyJob ..................................................................................................5
  Parsing Logical Expressions ...............................................................................................6
Task 3. WISE GUI Development .............................................................................................6
Task 4. WISE Server Development ..........................................................................................12
Task 5. Evaluation of Schedulers ............................................................................................13
Task 6. Final Report .................................................................................................................14
Conclusions ..............................................................................................................................14
Introduction

For Phase I we formulated the following tasks:

Task 1: Develop specifications for the WISE application as a whole.

Task 2: Develop a CORBA/IDL interface and the C++ base class for a skim algorithm abstraction. Create an example of skim complying with the interface that will skim through a single data file.

Task 3: Augment the TechEcho client with a graphical user interface allowing users to select a remote database, choose a particular skim and set its configuration options.

Task 4: Augment the TechEcho server by adding capabilities to work with skimming applications and deliver the derived data to the client.

Task 5: Evaluate existing schedulers and their application programming interfaces for future integration with the server.

Task 6: Write the final report.

TechEcho mentioned in the tasks is a collaborative client-server system previously developed at Tech-X and used as a foundation of the WISE application. In what follows we give a description of the Phase I accomplishments and conclusions on the feasibility of the proposed project.

Task 1. WISE Specifications

This task was of paramount importance as its objective was to identify needs of HEP community in the field of data distribution and shape our vision for the project in Phase II. After communication to many HEP “data” people from the Fermi National Laboratory, Lawrence Berkeley Laboratory, SLAC, University of Colorado at Boulder, Colorado State University, and University of California at Santa Cruz, we decided to start a collaboration with the BaBar data distribution group. There were many reasons to do so. BaBar produces the most HEP data in the US and has developed an extensive set of software tools to extract and distribute data to its collaborators. These tools have a potential to be reused in planned experiments at LHC. Finally, the BaBar Computing System Coordinator Dr. Jim Smith is also professor of University of Colorado. Tech-X Corporation has close ties with CU and is located in the same town. Exchange of ideas and information about BaBar databases with Jim Smith and Stephen Gowdy - Deputy Computing Coordinator – were invaluable and formed our ideas about the application specifications.

The detailed specifications are given in Phase II tasks. Here we provide just a brief outline.

The WISE application will be a distributed application for the Web access to the BaBar data distribution tools BdbServer++. As such, it will allow for “visual” control and use of the BaBar tools for data selection and distribution in the form of Objectivity and KANGA
collections. We envision providing several client-server sub-systems. Three subsystems will deal with data location and selection, data copying and data extraction and distribution. To streamline the development of these sub-systems, we develop technologies allowing for auto-generation of GUIs from XML files describing the controlled applications.

A special bookkeeping subsystem will be devoted to monitor jobs status and to deal with failure. It will work in conjunction with the first three systems. Finally, we will provide collaborative features in each of the subsystems allowing for direct communication between collaborators and creating persistent shared documents. All the subsystems will be enhanced with authentication and authorization mechanisms.

Each interface will be implemented as accessible via Web browsers. Each interface will allow users to set input parameters and options to the tool on the server through GUI facilities. Options and parameters will be given natural default values. All tools will have help pages with the input/output descriptions and lists of available input options and parameters.

Task 2. WISE API and Examples of Implementation

Here we describe the WISE API developed in Phase I to interface the BaBar software framework. First we give some background on this framework and proceed with description of the API for a particular tool of this framework: BdbCopyJob. Finally, we discuss our approach for parsing logical strings, which will be used in Phase II for setting collections via tag bits.

BaBar Framework and BdbCopyJob package

To access the BaBar software framework one starts with creating a user release. This process is managed by the Software Release Tools (SRT). A release consists of a consistent set of packages together with the libraries and binaries created for various machine architectures. The basic principle behind SRT, from a user's point of view, is that users create their own private releases, similar to those found under $BFROOT/dist/releases, where BFROOT is the environment variable pointing to the local installation of the BaBar software.

In the user's case, a release is typically a very lightweight collection of directories and is tied to a full BaBar release. With the initialization and creation of a user's release comes a GNUmakefile, which "knows" where to look for binaries, libraries, and include files of the official BaBar release. After creating the release, one needs to explicitly check out the needed packages from the CVS repository. A package is a self-contained piece of software intended to perform a well defined task. Each package has a unique name and its own library and include files.

In our case, we added the BdbCopyJob package to the release. This package allows users to select a range of events from an input Objectivity collections and copy them to output collections within the BaBar event store. BdbCopyJob package consists of a complete executable and a set of Tcl and the extended C shell (tcsh) scripts to configure it for particular copy jobs.

By convention, names of the Tcl/tcsh scripts have the form XXXBdbReadYYY.tcl and XXXBdbWriteYYY.tcl, where “XXX” denotes the system identifier and “YYY” specifies the
type of data, such as Raw, Micro, Rec, etc. Our release contains the CopyMicro.tcl and the CopyMicro.tcsh scripts. The CopyMicro.tcsh script sets the needed environment variables, the input and output collection names, and then runs the BdbCopyJob executable with the appropriate Tcl files to copy the events. The scripts also check for the existence of the output collection and by convention they create it if it does not exist. The written events will be appended to the output collection if it exists.

In order to run BdbCopyJob, one has to determine correct names for the input and output collections. This can be done by getting collection lists through BdbInspector, bfreport, colldb, or skimdata commands. We used BdbInspector to query specific collections. The set of tokens in the collection names can be considered as a part of a directory tree. The tokens specify a number of things but the first two are used to account for disk and tape usage, the /groups/Stream17 and /users/dad in our case. The rest of the tokens can be picked by users to create a specific data structuring as needed. These tokens are used to denote the release, the range of events selected in the collection, and other relevant information. For example, the names of the input and output collections used in our Phase I exercise are: /groups/Stream17/0002/P10.2.3gV00fb/00024770/cb001/allevents and /users/dad/Stream17/0002/P10.2.3gV00fb/00024770/cb001/allevents.

WISE API for BdbCopyJob

The client server system TechEcho used as a foundation for our work allows bringing a command line emulator into a Java applet. Through this command line, users can run remote applications. Each application is governed by an object of the type Distributor. To accommodate new capabilities associated with the BdbCopyJob, we extended the Distributor hierarchy as shown on Fig. 2.

The Display interface is at the root node of this diagram and allows a client and a server to exchange and display messages and images. The Distributor interface declares methods for connecting to clients and distributing information (texts and images) to them. AppDistributor has methods to run applications by sending commands and sets of commands from clients to the server. Here is the snippet of the AppDistributor interface:

```
#include "Distributor.idl"
module techoidl {
    interface AppDistributor : Distributor {
        void sendLineToApp(in string from, in string line);
        void sendTextToApp(in string from, in string text);
    };
};
```

This code is written in OMG-IDL (Interface Definition Language). IDL is language neutral and is used by the IDL compiler to generate stubs (the client proxies of the servant objects) and skeletons (base classes for the server implementation).

TxCopyDistributor adds a capability to list available collections, so that users could provide names for input and output collections for BdbCopyJob:

```
#include "AppDistributor.idl"
module wiseidl {
```
interface TxCopyDistributor : techoidl::AppDistributor {
void listAvailableCollections(in string from);
};

TxDistributor allowed us to create a new session ”BdbCopyJob” in the application and bring in the BaBar command prompt into the Java client (see the next session for more details on the interface), from which we could run BdbCopyJob commands.

Parsing Logical Expressions

Logical expressions arise when selection criteria are imposed. We found that a very efficient solution of this problem is to represent any user supplied logical expressions as a valid Python code. We then use the Python interpreter to parse and evaluate these expressions. The Python parser is extremely powerful and well tested. It has an extensive error handling capabilities. The parser is very generic and can evaluate other expressions that involve non-logical types. This is a feature that is required for evaluation of expressions that consist of Boolean and numerical types.

Task 3. WISE GUI Development

We describe here components of the graphical user interface needed to access selected tools of the BaBar software framework: BdbCopyJob used for copying collections and bbftp used for file transfer. These components are written in Java and are accessed through an applet, so that the GUI can be Web invocable. The components are: connection interface, the session list interface, the interface to the BdbCopyJob and the interface to the bbftp.

The connection interface (Fig. 3) can be accessed either via a web browser or through the java virtual machine. The start up interface of the client in this case is running on a machine in Tech-X Corporation. A java applet is loaded in the Netscape Navigator. A user should enter the name of the host on which the WISE server is running and to which to connect to. This can be done either by using the pull down menu to select from a list of available host names or to directly type the name of the host in the top-level text field. In this example, we use a host name
of a machine in SLAC.

The second text field allows users to enter the port number for the network connection. However a default port number is provided in the start panel. This should be the port number used by the WISE server to listen for connections. The last text field is for the client’s user name. We are ready to connect to the server after we have provided the appropriate information in these three fields. The connection is initiated when we click on the “CONNECT” button. Provided the connection is successful, a new window is displayed that lists the available sessions on the server. For the connection in our example, this window is shown in Fig. 4.

Users are able to choose to which session to connect via this interface. There are four available sessions listed in Fig. 4. The first session is to start a bbftp session for efficient transfer of files from the site of the server, SLAC in this case. The second session is to run remotely the BdbCopyJob tools. The last two sessions are for chatting with other connected users and for remotely running the Python interpreter. The two text fields in the bottom left corner of the figure are for sending messages to all or selected users connected to the same session. The text area on top of them is to display messages sent to this client.

We have selected the “new_BdbCopyJob” session. To connect to this session we use the “Sessions” pull-down menu on the menu bar and select “Connect to Selected”, not shown in Fig. 4. The successful completion of this connection displays the BdbCopyJob session interface in a separate window. This is shown in Fig. 5.
Fig. 3. The GUI of the client’s start panel.
There are a number of commands to be executed to copy from one Objectivity collection to another. The traditional way to execute this operation is to place the needed commands in a Tcl file and then to source the file to the BdbCopyJob. The command for this is displayed in the “Process:” text field in Fig. 6. The Tcl file we use is CopyMicro.tcl and in this case it resides on the server side. Our interface provides the capability, through the “Windows” menu in Fig. 6, to open a file that resides on the client side and to send its contents to the BdbCopyJob on the server for processing. In this way, users can develop their own Tcl copy scripts and then to submit them to the BdbCopyJob running remotely.

The completion of the copy job is displayed in Fig. 7. The CopyMicro.tcl script was configured to extract 33 events from the input collection and to copy them to the output collection. These events were appended to the output collection. At the end of the copy we have run the BdbInspector again and listed the output collection to check the new number of events in it. The output from BdbInspector displays that the number of events now is 10166.

The last graphical user interface we developed in the Phase I was for the bbftp session. bbftp is used for efficient and secure transfer of files from the server site to a remote location. This interface is shown in Fig. 8. It contains a text list dialog area in which files from a specific
Fig. 5. The client GUI to the BdbCopyJob tools running on the server.

Fig. 6. Using the BdbInspector remotely from the WISE client.
export directory on the server host are listed. In our example these are some of the database files exported from the output collection we created with the BdbCopyJob session. Users select which of these files are to be transferred to a remote location. Then, the user name under which account these files will be transferred has to be provided, the remote host name, aubisque.Colorado.edu in Fig. 8, and the number of parallel streams to be used by bbftp. The file transfer job is initiated on the server by pressing the “Put File(s)” button. The response from the
bbftp job on the server will be displayed in the top text area of the left panel in Fig. 8.

The complete set of graphical user interfaces we created during the Phase I allows users to access a tier-A site, copy events from one Objectivity collection to another, and finally to transfer an extracted collection from a tier-A to a tier-C site. We demonstrated a working prototype with the server at SLAC and a transfer of database files from SLAC to the University of Colorado.

**Task 4. WISE Server Development**

Here we describe extensions to the server needed to bring BaBar data manipulation packages into WISE. These extensions take place in the server configuration, OMG-IDL interfaces, server implementation classes and wrappers for accessing the executables. As an example, we incorporated partial capabilities of the BdbCopyJob and bbftp packages. By accomplishing this, we demonstrated that using WISE, a remote client can copy events from one collection to another, and to transfer an extracted collection to a remote site. This proves that the proposed technical approach is adequate.

To register a new application within WISE, one has to create a new configuration file. The configuration file contains a set of “name = value” pairs, one per line. This file defines a path to the application, the working directory, the string that tells the application to quit, whether the application generates images, etc. This file is read by the server at startup and this makes the application available to clients.

Applications are implemented as servants of the **Distributor** class hierarchy as shown in Fig. 9. Some of the classes shown on this diagram are generated by the CORBA IDL compiler from the OMG-IDL interfaces discussed in Task 2. These classes are called skeletons and do not provide implementation but take care of networking and de/marchalling. Implementation classes, DistributorImpl, ConnDistributorImpl and AppDistributorImpl, derive from them and provide needed implementation. They have methods to create a session, connect a client to a session, etc. AppDistributorImpl adds the functionality needed for starting up an application, directing messages to it, and receiving messages from it. It also does the necessary copying of files to the web server directory. This hierarchy is duplicated from the interface files.

To provide new capabilities associated with running BdbCopyJob and bbftp, we added new OMG-IDL interfaces to the Distributor hierarchy: **TxCopyDistributor** and **TxBBFTPDistributor**. Implementation for these is provided **TxCopyDistributorImpl** and **TxBBFTPDistributorImpl** derived from them.

In order to run applications on the server, the WISE server spawns them as child processes. For example, the server spawns the BdbCopyJob when the client requests the creation of a new BdbCopyJob session. We use pipes and/or the pseudo-terminal utility to handle the input/output streams from the parent to the child processes and in the reverse direction. In this way the messages initiated on the client are first transferred to the server and then the server communicates them to the spawned application that runs as a child process. The server process listens for output from the child process and when such output is available, it is read and
transferred back to the client.

Our experience shows that the most rapid and flexible way to provide an interface between the server and existing executables on the server side is to use the Python programming language as the glue between them. For this reason we have written Python scripts to handle the inter-process communications between the server and various executables that we want to provide access to from the client. The Python scripts are very easy to read and maintain.

**Task 5. Evaluation of Schedulers**

As part of our work in Phase I, we investigated a number of schedulers/batch queuing systems. These systems are an essential part in the management of multiple user jobs running on the server. The multi user environment in BaBar requires proper management of user submitted jobs such as BdbCopyJob, skimming and BdbServer jobs. Jobs of these types that take a long time to complete are queued and managed by schedulers/batch queuing systems. We designed our server so that it could communicate with existing schedulers used on the HEP servers. This feature of the WISE server will give users the control to submit and monitor their jobs remotely.

We studied the following schedulers: the Portable Batch System (PBS, developed by Veridan Systems for NASA),¹ the Maui scheduler,² developed in the Maui High Performance Computer Center, and the LSF batch queuing system.

The Maui scheduler is an advanced reservation based high performance computing scheduler. It is known to run on SP and Unix/Linux based clusters. It is used to extend the functionality and improve the efficiency of computing sites that run the PBS batch system. The combined use of Maui and PBS gives the power to impose flexible user policies and reservations while efficiently managing job queues and computer resources. The problem with the PBS is that its graphical user interface is not cross-platform.
We determined that the LSF\textsuperscript{3} tools are currently used in SLAC as the batch and job scheduling systems. LSF is a suit of job management utilities developed by the Platform Computing Corporation. LSF has the capabilities to analyze, monitor, and manage the workload for a network of computers from different platforms, including UNIX and NT computer systems. LSF gives the flexibility to consider the whole network of computers it manages as a single system. A job submitted to the LSF system will be run on the best available host from the underlying network of hosts. LSF can manage both single and parallel jobs running in interactive and batch mode.

Of particular importance to our WISE server is the API that LSF provides. This interface consists of a set of programming libraries in C as well as a tool kit of programs for writing shell scripts. Since the WISE server is written in C++ we can interface directly with the LSF API, which is written in C. In this way we have direct access to the LSF API libraries. We expect that any job control and management features to be provided through the WISE server will be done by interfacing with the LSF suite of tools.

**Task 6. Final Report**

The final report is written.

**Conclusions**

All the objectives of Phase I have been achieved, and all the tasks have been completed. We have identified the features of WISE requested by the HEP community: to provide a uniform API and Web invocable graphical user interface for the BaBar tools for data selection and distribution. We concluded that the application should be secure and collaborative and satisfy needs of users with various sets of skills: novices and experienced users. We have started a collaboration with the BaBar data distribution group, which will provide us with access to databases and data distribution tools, instruct us on the use of these tools and test the application as it progresses.

We have familiarized ourselves with the BaBar software framework and incorporated several tools from this framework into WISE. This demonstrated that our technical approach is adequate. Successful demonstration of remote file copying and transferring (from SLAC to University of Colorado) via WISE demonstrated that the proposed project is feasible.

\[1\] http://www.openpbs.org
\[2\] http://mauischeduler.sourceforge.net
\[3\] http://www.platform.com/products/wm/LSF/index.asp