Java Application for Creating a Shared Object Cache

I. Mejuev I. Abe
High Energy Accelerator Research Organization (KEK)
1-1 OHO, Tsukuba, Ibaraki 305, Japan

Abstract

The Java language is used to create thin GUI clients connected to a server implemented as a Java application. The server contains an object cache, which is updated by calls to an underlying system layer. Using object-connection technology, we establish connections between objects from the server cache and GUI clients' objects. The states of the connected objects are synchronized so that all changes in the objects' state are transferred from client to the server, and vise versa. Since only changes are transferred, the control network traffic is reduced and the performance increased. Software development is also simplified, since both the client and server don't have to take care of the objects' states synchronization. The sockets or distributed Java object systems, such as RMI or HORB, can be used to transfer the states. Since the system is implemented completely on Java, it can be made a multiplatform, and control clients can run on any Java-enabled browsers with minimum system requirements.

1 Introduction

The main advantages of applying Java for accelerator control are the use of a Java distributed software model and connectivity provided by this language. The benefits of using browsers as client software are also well known. Besides, currently, the majority of software support Java, and today we have an excellent choice of authoring and publishing software.

However, severe timing limitations and the nature of control programs pose additional requirements on the software and class libraries used for development. In control software it is often necessary to deal with a data stream, rather than with a simple request-reply communication model. The performance of applications based on device polling is often not sufficient if the number of devices is growing. This is especially true for Java applets, which are supposed to run on browsers with minimum system requirements. The solution to this problem is to use notification or "push" technology, where clients are informed by the server about any changes of relevant data.

In this paper we describe Java class libraries developed to provide notifications for clients' applets with extended capabilities, such as the creation of named data channels as well as subscription and synchronization of objects' states across a network.Further, we describe the design of a "Shared Object Cache" application based on those class libraries. This application is used as a distributor of data coming from devices to clients' applets, and also for providing the possibility of bi-directional data exchange.
The development of Java class libraries and applications is in line with our efforts to apply a variety of Internet and Intranet technologies for accelerator control [1], [2].

2 Data channel classes

A data channel class library provides basic functions, which are used by the other Java classes and applications described in this paper. The main idea of using channels is represented in the Figure 1. A Java application called a "channel server" is run on HOST 2, so that applets or applications running on other computers can create, delete, subscribe, unsubscribe and update channels on HOST 1.

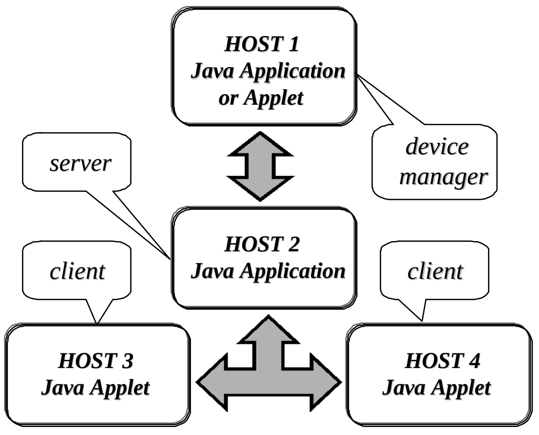


Figure 1

If the application or applet from HOST 1 updates a channel owned by HOST 2 and, that channel has been subscribed by HOST 3 and HOST 4, the applets running on those hosts receive a notification. A channel update means the submission of a primitive data type to a channel, followed by a value name, represented by an instance of the String class in Java.

The ChannelManager class provides the functionality described above; it uses an interface named ChannelUpdateListener to notify the user about channels' updates and other events. To indicate errors, a ChannelException instance is thrown.

2.1 ChannelManager class

ChannelManager class has two constructors. One creates a channel server with the possibility to own channels as well as to send requests to other hosts. Another constructor creates only a client, which can use channels served by another host. Since Java applets loaded from a remote host cannot listen for TCP connections, they can use only the second type of constructor. Below we describe the public methods of ChannelManager class in more detail. A destination (where) is represented in the form of a string, which contains the TCP port and host name.

- addListener(ChannelUpdateListener listener)
  Registers an object which has implemented the ChannelUpdateListener interface; through the methods of this interface the user receives notifications about the updates of channels' and other events.
- pingHost(String where)
  Verifies that there is a ChannelManager running at a specified location, and that it can accept connections.
- createChannel(String channel, String where)
  Creates a channel at a specified location.
- closeChannel(String channel, String where)
  Closes a channel at a specified location; if the channel doesn't exist, a ChannelException is thrown.
- subscribe(String channel, String where)
  Subscribes a channel at a specified location; if the channel doesn't exist, a ChannelException is thrown.
- unsubscribe(String channel, String where)
  Unsubscribes a channel at a specified location; if the channel doesn't exist, or it hasn't been subscribed, nothing happens.
- updateChannel(String channel, String where, String valueID, Object value)

Updates a channel at a specified location. The string valueID is used as a value name. If the channel exists, all subscribers receive a channelUpdated message.
The value can be an instance of the following Java classes: String, Integer, Long, Float, Double, Boolean or a Vector, comprising these types. It is possible to mix different types in a Vector. An attempt to pass an illegal data type to this function generates a ChannelException.

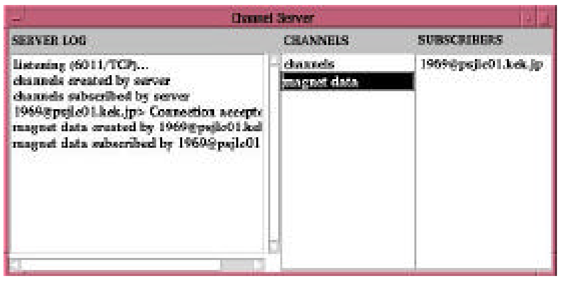
2.2 *Channel update listener interface*

An applet or application using ChannelManager implements functions of this interface to receive the updates of channels and other kinds of notifications. Throwing an exception in these interface methods generates an error reply; it can be used to create some kind of security. Abstract methods of the Channel Update Listener interface are described below.

- channelUpdated(String channel, String where, String valueID, Object value, String byWhom)
  Called by ChannelManager when a remote or local channel is updated, and if that channel has been previously subscribed. It is possible to prevent updating of a local channel by throwing an exception.
- channelClosed(String channel, String where, String byWhom)
  Called when a previously subscribed channel is about to be closed. It is possible to prevent the closing of a local channel by throwing an exception.
- channelSubscribed(String channel, String byWhom)
  Called when a remote host or a local listener subscribes a local channel. Throwing an exception can prevent a subscription.
- channelCreated(String channel, String byWhom)
  Called then a local channel is created by a remote host or a local listener. Can be prevented by throwing an exception.
- channelUnsubscribed(String channel, String byWhom)
  Called when a local channel is unsubscribed by a remote host or a local listener.

2.3 *Channel server application*

Clients' applets use connections to the Channel Server application in order to transfer commands and to exchange data. Additional functionality, which is provided by this application, includes a graphical user interface, logging, and the possibility to inform applets about what channels are currently available on a server.



```
listening (6011/TCP)...
channels created by server
channels subscribed by server
1969@psjlc01.kek.jp> Connection accepte
magnet data created by 1969@psjlc01.kel
magnet data subscribed by 1969@psjlc01
```

Figure 2

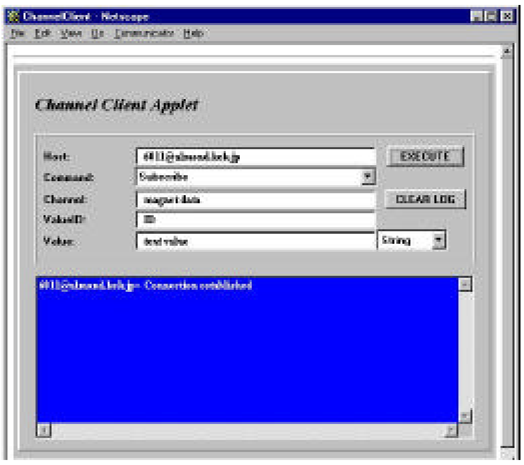
2.4 *Channel client applet*

To simplify the diagnostics and testing of software we also developed a Channel Client Applet, which provides the possibility to manually execute channel commands using a convenient graphical interface.

3 Object connection classes

Data channels can be used for a wide range of control and other applications; however it is often necessary to synchronize the states of objects distributed across a network. For these purpose we developed classes implementing such synchronization. These classes use a channel library for data transfer. The following public classes have been developed: ObjectConnectionManager and CMObject.

The source code for a client using an object connection is very simple:



Host: 4011@almond.kek.jp
Command: Subscribe
Channel: magnet data
ValueID: ID
Value: test value String

```
4011@almond.kek.jp: Connection established
```

Figure 3

manager = new ObjectConnectionManager();
object = new CMObject(“name”, manager);
object.connect(“port@host”);

This code creates a connection between local object and server object on “host”.
If either server or client modifies the local object executing for instance:
object.update(new Float(1.1));

the change in object’s state is transferred across network and method updated() is called for a remote object.

4 Object cache application design

Object Cache Application belongs to a middle layer of the control software. It provides device managers and console
or remote clients with the possibility to share a common object database. Since sockets (TCP) are used for data transfer, the application performance is better if compared with SQL databases or distributed object systems.

Applets or applications, providing a graphical interface for the users, communicate to a shared object cache using Object Connection classes, which are described above. It Significantly simplifies the development of graphical Interface software, and also reduces the upgrade and maintenance cost.

Data providers, such as various device managers, while communicating with a server use a data channel interface, which is more suitable for handling data streams. Data coming from the devices are cached in the server memory, thus significantly increasing the clients’ applets performance.

The design layout of the Object Cache application is represented in Figure 4.

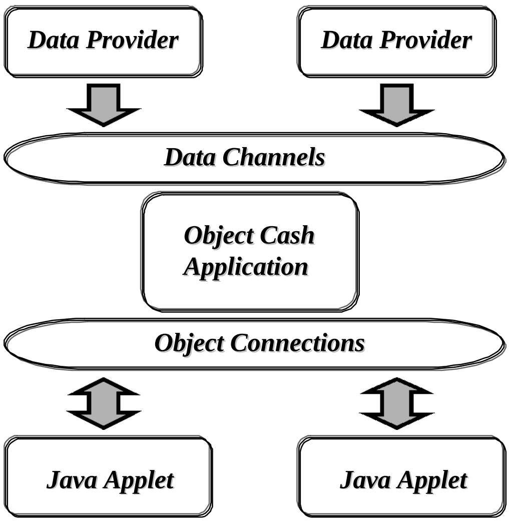


Figure 4

References

[1] The Use of Virtual Reality For a Multimedia Informational System Development; Igor Mejuev and Isamu Abe; International Workshop on Controls for Small- and Medium-Scale Accelerators, 1996, KEK, Tsukuba, Japan.

[2] Applications of Internet and Intranet Technologies for Distributed Control; I. Mejuev, I. Abe; Proceedings of the 22nd Linear Accelerator Meeting in Japan; September 9-11, 1997, Sendai, Japan.