2003 추계학술발표회 논문집
한국원자력학회

Conceptual Study of the Application Software Manager Using the Xlet Model in the Nuclear Fields

Joon-Koo Lee, Heui-Youn Park, In-Soo Koo, Hee-Seok Park*,
Jung-Seon Kim*, Chang-Ho Sohn*
KAERI, *Samchang Enterprise Co., LTD.
150 DukJin Dong YuSung Gu, Taejeon

Abstract

In order to reduce the cost of software maintenance including software modification, we suggest the object oriented program with checking the version of application program using the Java language and the technique of executing the downloaded application program via network using the application manager. In order to change the traditional scheduler to the application manager we have adopted the Xlet concept in the nuclear fields using the network. In usual Xlet means a Java application that runs on the digital television receiver.

The Java TV Application Program Interface(API) defines an application model called the Xlet application lifecycle. Java applications that use this lifecycle model are called Xlets. The Xlet application lifecycle is compatible with the existing application environment and virtual machine technology. The Xlet application lifecycle model defines the dialog (protocol) between an Xlet and its environment

I. Introduction

In the nuclear field, all applications are programmed by the structural technique using the Fortran or C language. There is many waste time to change the application program. After modifying the application program, we have to compile and link all the application programs again.

In order to reduce the cost of software maintenance including software modification, we suggest the object oriented program with checking the version

of application program using the Java language and the technique of executing the downloaded application program via network using the application manager.

In order to change the traditional scheduler to the application manager we will adopt the Xlet concept in the nuclear fields using the network. In usual Xlet means a Java application that runs on the digital television receiver.

The Java TV Application Program Interface(API) defines an application model called the Xlet application lifecycle. Java applications that use this lifecycle model are called Xlets. The Xlet application lifecycle is compatible with the existing application environment and virtual machine technology. The Xlet application lifecycle model defines the dialog (protocol) between an Xlet and its environment through the following:

A simple, well-defined state machine
A concise definition of the application's states
An API to signal changes between the states

Application Manager is a part of a digital television receiver's software operating environment that manages Java applications. The application manager controls the lifecycle of an Xlet by signalling its state changes. An application manager is required on a receiver, but its precise behavior is implementation specific. Generally the requirements of application manager is as follows:

An Xlet can be destroyed at any time. An application manager is the entity on a digital television receiver that has ultimate control over the Xlets it manages. Therefore, the application manager must be able to destroy an Xlet at any time.

The current state of an Xlet will always be known.
An application manager is responsible for signaling Xlets regarding their current state. Xlets, however, can also change theirown states, but they must signal those changes back to the application manager.

An application manager can change the state of an Xlet.
The primary purpose of an application manager is to direct the state changes of an Xlet.

An application manager will know if an Xlet has changed its state.
One of the features of the Xlet application lifecycle API is that the Xlet can change its own state. Therefore, the application manager must be notified of this state change so it can track the state of the Xlet.

The states changes of an Xlet are handled by the Xlet itself, i.e., only the Xlet knows when the state has been successfully changed. The four Xlet states are Loaded, Active, Paused, and Destroyed. Xlets communicate with the Application manager about state changes via callbacks. The Xlet signals the success or failure of such changes with the return value of the callbacks. The definition of the four Xlet state is as follows;

Loaded

The Xlet has been loaded and has not been initialized. This state is entered after the Xlet has been created using new. The no-argument constructor for the Xlet is called and returns without throwing an exception. The Xlet typically does little or no initialization in this step. If an exception occurs, the Xlet immediately enters the Destroyed state and is discarded. Note: This state is entered only once per instance of an Xlet.

Paused

The Xlet is initialized and quiescent. It should not be holding or using any shared resources. This state is entered: From the Loaded state after the Xlet.initXlet() method returns successfully, or From the Active state after the Xlet.pauseXlet() method returns successfully, or From the Active state before the XletContext.notifyPaused() method returns successfully to the Xlet.

Active

The Xlet is functioning normally and providing service. This state is entered from the Paused state after the Xlet.startXlet() method returns successfully.

Destroyed

The Xlet has released all of its resources and
terminated. This state is entered: When the destroyXlet() method for the Xlet returns successfully. The destroyXlet() method shall release all
resources held and perform any necessary clean up so it may be garbage collected; or When the XletContext.notifyDestroyed() method returns successfully

to the Xlet. The Xlet must perform the equivalent of the Xlet.destroyXlet() method before calling XletContext.notifyDestroyed. The Xlet state machine is designed to ensure that the behavior of an Xlet is as close as possible to the behavior television viewers expect, specifically:

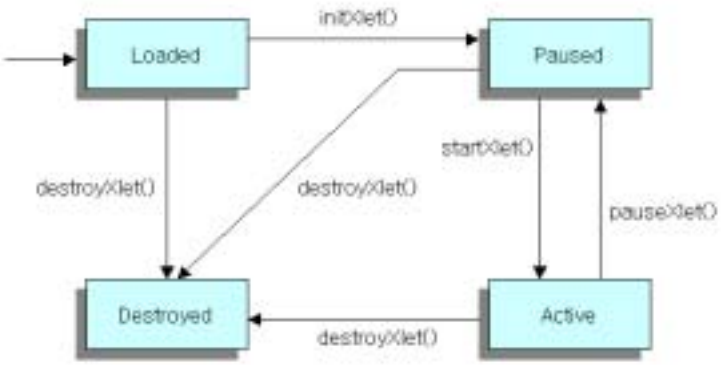

FIGURE 1 Xlet State Machine Diagram

II. System Design

II.1 System Configuration

Proposed system configuration consists of maintenance server, application manager (Xlet Manager), event listener, and hash table as shown in Figure 2.

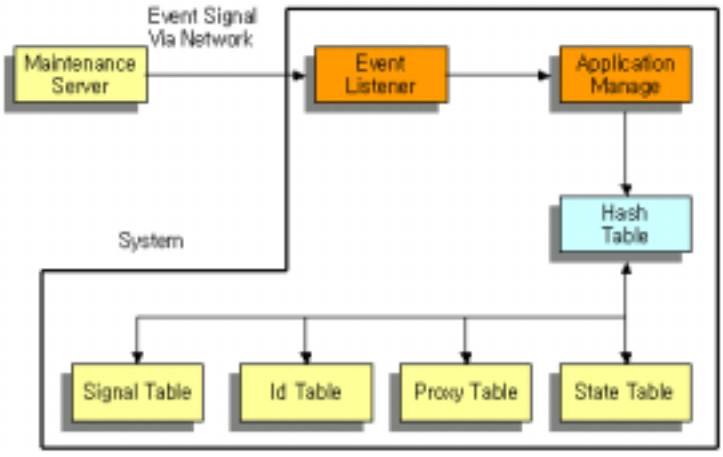

Figure 2 Proposed System Structure

II.2 Application Manager

The application manager is based on the Xlet application model. The classes implement the functionality for the Xlet state machine, send notification about state changes, load Xlet classes, and communicate between Xlet and Application Manager as shown in Figure 3. And Data Flow Diagram for Application Manager to execute the application instances is shown in Figure 4.

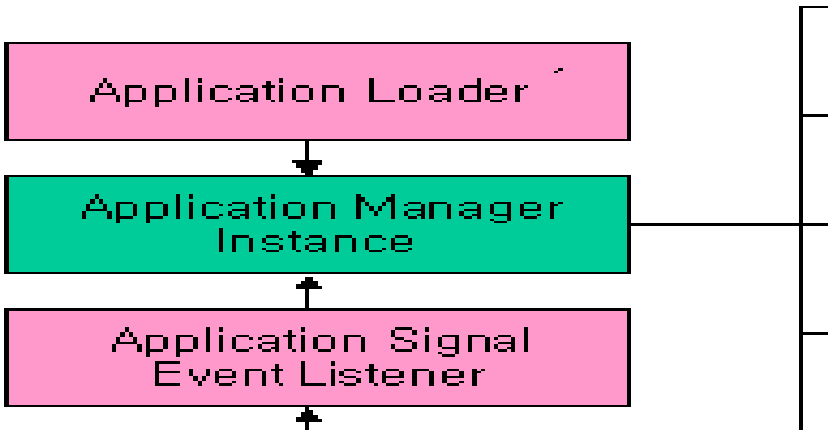

Figure 3. Application Manager

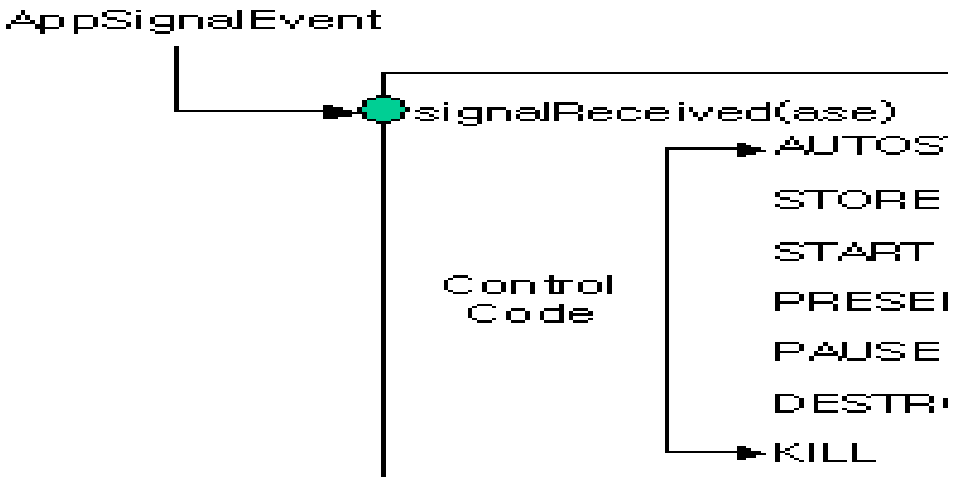

Figure 4. data Flow Diagram in the application Manager

II.3 Signal Event

The Application Signal Event class is used by the service selection classes to notify the Application Manager that the current service includes an Xlet to be signaled for execution.

II.4 Event Listener

Application Signal Event Listener is an event listener interface implemented by classes to receive notification of Application Signal Event objects.

II.5 Hash Tables

Hash table is composed of signal table, id table, proxy table and state table. Signal table contains the information for execution of downloaded application program including application control code which is described in the Table 1.

Table 1. Application Control Code

Code	Identifier	Semantics
0x01	AUTOSTART	The Object Carousel module containing the class implementing the Xlet interface is loaded, the class implementing the Xlet is loaded into the VM and an Xlet object is instantiated, and the application is started.
0x02	PRESENT	Indicates that the application is present in the service, but is not autostarted.
0x03	PREFETCH	Indicates that the receiver should try to prefetch the application. Exact semantics to be defined.
0x04	DESTROY	When the control code changes from *AUTOSTART* or *PRESENT* to *DESTROY*, the destroy method of the Xlet is called by the application manager and the application is allowed to destroy itself gracefully.
0x05	KILL	When the control code changes to *KILL*, the application is terminated by the application manager.

Id table has identifiers for the application program. And proxy table

contains all necessary information of the application for execution of application instance, i.e, application loader, application manager, application context, request, result, etc.. Finally state table contains the state of application instances, loaded, paused, active, and destroyed.

III. Conclusion

In the nuclear field, all application program is controlled bythe scheduler without checking the version of application programs. All application program have to be changed, recompiled, and linked again if there is a minor change in the algorithm.

Two concept is suggested in this paper. One is the concept of object oriented programming with checking the version of application program and the other is the concept of executing the downloaded application program via network using the application manager depicted in this paper.

The cost of the maintenance of application program will be deducted by the adoption of these concept in many aspect of the nuclear fields. We can change and modify the application program in ease by the object concept without compiling and linking all applications. Also there is no waste time because of executing the downloaded application program with the application version via network.

References

[1] "Digital Video Broadcasting (DVB) Multimedia Home Platform (MHP) Specification 1.0", ETSI TS 101 812, V1.1.1, July 2000.

[2] "Java TV API Reference Implementation Porting Guide", Version 1.0, Nov. 15, 2000, Sun Microsoft.

[3] "The Essential Guide to Digital Set-top Boxes and Interactive TV", Gerard O'Driscoll.

[4] "Java TV APIs", Version 1.0, Sun Microsoft.

[5] "Java TV 1.0", Sun Microsoft