

## A VoD Application Implemented in Java

Accepted for publication in a special edition of *Multimedia Tools and Application*, Kluwers Academic Publishers, 1997

ALEX DE JONG, KAREN HSING, DAVID SU {alex.dejong,karen.hsing,david.su}@nist.gov  
*Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899*

*Received November, 1996*

### **Editor:**

**Abstract.** This article describes an implementation of a Video-on-Demand (VoD) system for the VoD interoperability test laboratory of the National Institute of Standards and Technology. It describes how Java can be used to implement the client side of a DAVIC based client-server system, which consists of a video server and a Set-Top Unit (STU) client. The communication between the video server and the STU is based on the Digital Storage Media Command and Control (DSM-CC) protocols. The VoD application is defined as a set of Java classes. These classes implement the graphical interfaces for user navigation and control. While this system is compliant with DAVIC specifications, it also permits an elegant integration of DAVIC services to the Internet environment.

**Keywords:** Video on Demand, Java, DSM-CC, DAVIC, Set-Top Unit

### **1. Introduction**

A recent major advancement in information technology is in the creation, distribution, and use of information. The content of information has changed from simple text data to mixed information involving video, graphics, text, and audio. These information sources are available remotely from distributed hosts and service providers. Users interact with information through information appliances which may look like a personal computer or television set-top unit. The seamless transfer of information from content developers to information providers, to network access providers, and finally to users is a major issue for the success of any new application. It is important to address the interoperability issues at an early stage of technology development, particularly those issues that arise when users access information through appliances with various capabilities, and through networks from many service providers.

It is expected that digital video services will become a major business sector in the near future. The National Institute of Standards and Technology (NIST) has established programs to foster interoperability of products and services from this emerging industry. NIST has set up an interoperability test laboratory for the Video-on-Demand (VoD) service, one of many digital video services. This laboratory allows vendors to test the interoperability of their products with NIST's standards-conforming implementations as well as other vendor's products. As part

of this activity, NIST is undertaking two major tasks: 1) development of a VoD reference implementation to facilitate testing, and 2) development of test methods for interoperability testing. The VoD service is based on the specifications of the Digital Audio-Video Council (DAVIC), an international consortium to develop implementation agreements for digital video applications.

“Video on Demand” means many different things to different people. In this article, VoD refers to a network delivered video service that offers the functionality of a home VCR (as a player only). By using a set-top unit, the service provides the users/consumers the ability to: 1) find and select a content item (e.g., a movie), and 2) interactively manage the viewing of the content such as fast forward, pause and resume the displaying of a movie. The first functionality is referred to as “navigation” and the second one “control”.

In addition to this basic functionality, the service may provide the users a choice of a language for audio and/or sub-titles (called presentation control by DAVIC). Functions that enable loading/removal of the content to/from the service for the providers are certainly required. Before entering VoD, a user should be able to make a selection of a service and a service provider. The list is by no means exhaustive. NIST’s VoD reference implementation initially focuses on a set of selected core functions to perform navigation and control.

This article describes the development of the VoD application at the client side of NIST’s VoD system which is based on a client-server architecture. Section 2 depicts the model of VoD systems and the protocol stacks included in this model as defined by DAVIC. Section 3 describes why Java is chosen to implement the VoD application and how DAVIC systems can be integrated with Internet. Sections 4 and 5 present the design of this VoD application and a sample implementation using Java respectively. Section 6 presents the VoD testbed configuration at NIST.

## 2. Architecture Based on DAVIC

DAVIC has adopted a framework for interactive services which includes VoD[1]. A system is described in terms of reference points and information flows as shown in figure 1. The model includes a Service Provider System (SPS) and a Service Consumer System (SCS)<sup>1</sup> connected via a delivery system. For a VoD system, the SPS corresponds to the video server while the Set-Top Unit (STU) corresponds to the SCS. Between the server and the STU several reference points (only A1 and A9 are shown) and information flows (e.g. S1 through S4) are defined. The information flows enable the connection/session set-up/tear-down (S3/S4), and the exchange of control information (S2) and data (S1).

After initial connection set-up and configuration of the STU—via DSM-CC User-to-Network (U-N) [2]—the DSM-CC User-to-User (U-U) protocol stack enables reliable exchange of control information via the S2 information flow. A VoD application utilizes the services provided by the DSM-CC U-U protocol to implement the retrieval of information from the server to be used at the STU.

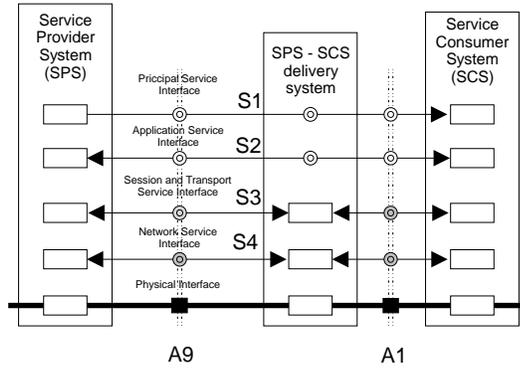


Figure 1. General DAVIC Model

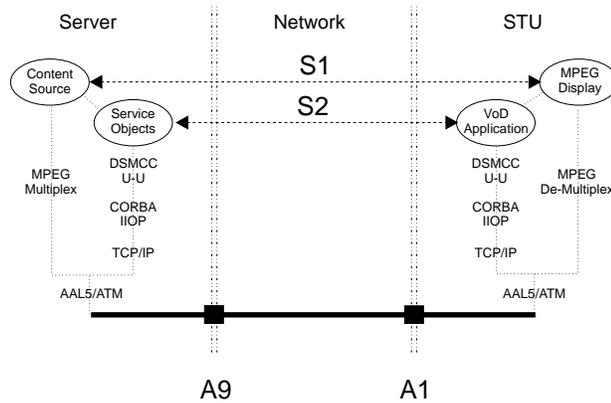


Figure 2. Video on Demand Model

Figure 2 shows a simplified model of an STU and a VoD server. Only information flows S1 and S2 are shown, including their full protocol stacks. The VoD application, which communicates with the server via information flow S2, uses the services provided by the DSM-CC U-U protocol. These services include *Session*, *File*, and *Directory* services. These services allow the VoD application to start a session and browse directories to retrieve files from the server. The files themselves can be used within the VoD application (e.g. images and bitmaps which comprise the user's interface).

Information flow S1 is used to carry MPEG 2 data. When a user selects to view a movie, an S1 connection is established (using information flow S3/S4 which is not discussed here) between the STUs and the VoD server. The STU's de-multiplexor/decoding device handles the incoming data and displays it to the user.

### 3. Why Java?

Interoperability between VoD services and STUs from different manufacturers is currently limited because of a variety of incompatibilities. DAVIC describes a framework (see previous section) and has adopted standards for the communications layers, including the DSM-CC [2]. These efforts should lead to interoperable products in the future. Assuming that interoperability is ensured at this level, there is still the problem of interoperability at the application level. In order to run a VoD application on a variety of platforms, such as STUs running Operating System OS9 or more general workstations running a UNIX based OS, a VoD server has to create dedicated applications for each platform. An option, which is embraced by DAVIC, is to assume that the client implements an MHEG-5 engine [3] to interpret generic application descriptions. Because of the complexity of an MHEG-5 engine, this solution might not be economically feasible on low-cost STUs. Another solution, which is also being studied in DAVIC, is to utilize the popular Java language used within many World Wide Web (WWW) applications. Using Java to implement the VoD application provides the following advantages:

- Platform independent
- Dynamic object loading
- DAVIC and Internet integration

The Java language is based on the Virtual Machine (VM) concept. The intermediate byte code generated by the Java compiler is interpreted by a Java interpreter<sup>2</sup>. The Java byte code is platform independent. Because of the language's simplicity, the interpreter can be kept relatively small. The basic Java interpreter is only 40 Kbytes in size; an additional 175 Kbytes is required to include basic standard libraries [4]. This makes Java an ideal solution to provide graphical interfaces for STUs with limited resources (up to 4 MBytes of memory) while working just as well on high end workstations with virtually unlimited memory (16 Mbytes or more and hard disk space far beyond this).

Since Java byte code is interpreted at run-time, dynamic additions of code to the run-time environment are relatively easy to implement. This means that an STU only has to retrieve objects that it needs to initiate a particular piece of an application while remaining objects are retrieved whenever they are required.

DAVIC specifications 1.1 already include direct Internet access. This means that a DAVIC based STU can access the Internet in addition to DAVIC based servers. Java enables the use of the popular Internet browsers to access DAVIC based servers. Products are already available that enable Java integration with a Common Object Request Broker Architecture (CORBA). An Internet user can access a service provider by its Universal Resource Locator (URL); after which a Java applet is invoked. This applet in itself is a CORBA client which allows seamless integration between Internet and DAVIC based services<sup>3</sup>.

#### 4. Design of the VoD Application

The VoD server consists of a server directory structure which contains: movie streams, Java objects implementing the user menus and interfaces, and bitmaps/graphics files used within the interfaces. Users connect their STU to the server, either by using an Internet browser (and starting an Applet) or a small dedicated Java program (which could be obtained via the DSM-CC *Download* protocol [2]) to obtain the initial application: *Init*. The *Init* application enables the user to attach to a VoD server via the DSM-CC *Session* service and obtain the actual VoD application by invoking *File* and *Directory* services.

The VoD program, depicted in the shaded area of Figure 3, is built on the DSM-CC U-U services. The upper half consists of 2 components: a static *Init* component that performs initial connection set-up, and the actual VoD program. Note that only the *Init* component resides on the client before a session has been established. The VoD program which implements user menus is loaded from the server dynamically, which allows a service provider to change the program without changing anything at the client side.

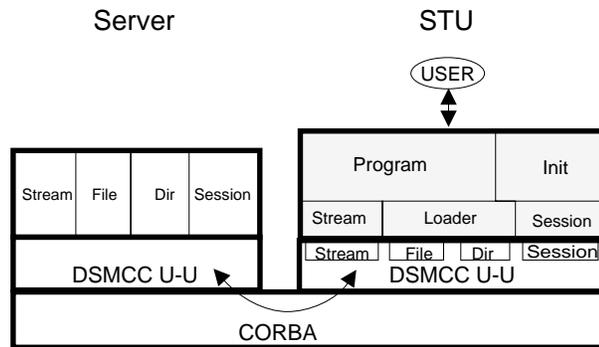


Figure 3. VoD application architecture

The *Session* component at application level provides attach and detach functions for *Init*. These functions are mapped directly into DSM-CC U-U *session* calls. The loader implements file retrieval functions for classes as well as data. Classes will be retrieved from the server and added to the run-time environment dynamically. Data files such as bitmaps and graphics can be retrieved for use by the program. The loader uses *File* service to retrieve requested information together with the *Directory* service to open/close directories on the server. A stream interface is layered directly on the *Stream* service in order to control a stream. All stream functions are implemented as application actions which can be attached to a button click or some other user event.

The DSM-CC U-U library contains the stubs to access the *Session*, *Directory*, *File*, and *Stream* services as specified in [2]. The Common Object Request Broker Architecture (CORBA) provides the underlying communications between the server and STU. The client-side library stubs will be generated directly from the DSM-CC Interface Definition Language (IDL) specifications by the IDL compiler [5]. The objects that implement the DSM-CC services are executed on the server.

## 5. Implementation of VoD Program

The VoD program is implemented as a set of Java objects. Java uses the O-O methodology and includes a class (or object) library which contains basic data types and system input/output capabilities. A powerful set of graphical interface classes is also included. These graphical interface classes are extended with VoD specific classes, such as VCR “play” or “stop” buttons, to provide a powerful set of so-called base classes<sup>4</sup>. The base classes themselves do not implement an actual user interface but provide the components to be used in the Graphical User Interface (GUI).

The VoD GUI consists of derived classes that actually implement the VoD menus and are specific to the VoD server contents. Only a derived class needs to be implemented to cover the details of a particular menu. For example, an addition of a new movie to a VoD server normally requires a movie specific entry in one of the menus which gives the user detailed movie information (e.g. actors, short description, etc.). The generic base classes will be re-used when new movies are added to the movie database (or within a completely different application) and or not VoD content specific. The base classes implement a model that consists of: *objects* and *actions*.

---

```
class AppObject
    class AppObjectButton
    class AppObjectLabel
    class AppObjectEntry
    class AppObjectImage
```

---

Each *object* represents a graphical piece of information on the screen. The Java pseudo-code<sup>5</sup> for these classes is listed above. Simple examples of such objects are: button, label, image, etc. All objects share some common attributes such as background color and sensitivity of the object to mouse clicks. Each object can define an associated *action* when the object is selected (either by mouse movement or by clicking).

Actions implement a mechanism similar to call-backs in traditional GUI programming. The difference, however, is that every action is a class in itself (versus

---

```

class AppAction
    class AppActionWindowNew
        class AppActionWindowQuit
        class AppActionStreamPlay
        class AppActionStreamPause
        class AppActionStreamResume
        class AppActionStreamStop

```

---

a function call). To associate an action with a button or graphics object, an action needs to be created and attached to an object. Whenever the object becomes activated, this action will be performed.

The application window class is intended to be used as a base class from which the VoD program classes are derived. The base application window class implements and manages the screen and the objects displayed on it. A simple example of a GUI with a background image and 2 buttons, *Continue* and *Quit*, can be denoted as follows:

---

```

// my application class window derived from base class AppWindow
class MyApplication extends AppWindow {
public void init(){
    setTitle("My Application");
    add(AppObjectImage (0,0,"bg.gif"));
    add(AppObjectButton(10,35,"Continue", AppActionWindowNew( NextApp )));
    add(AppObjectButton(30,35,"Quit", AppActionWindowQuit()));
}
}

// the next window has only a Quit button
class NextApp extends AppWindow {
public void init(){
    setTitle("Next Window");
    add(AppObjectImage (0,0,"bg.gif"));
    add(AppObjectButton(10,35,"Quit", AppActionWindowQuit()));
}
}

```

---

An example of a VoD program implementation using the application base classes as explained above is shown in figure 4. The opening screen shows the user a *Continue* and a *Quit* button.

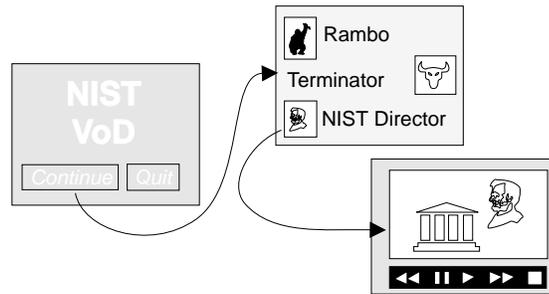


Figure 4. Example of a VoD program

After selecting *Continue*, a new window is shown which lists the movies available on the server. Each movie entry is listed with a preview picture and some additional information such as the movie title and actors. Selecting a movie opens up a VCR-like interface. It allows the user to start playing the movie, to fast forward, or invoke other common VCR playback functions.

Below is a list of classes that shows what is needed to implement the example shown in figure 4. Each class implements a window that can actually be displayed on the screen. Upon user selection different windows are displayed to allow the user to browse the VoD server's contents.

---

```
class OpeningWindow
class MovieListingWindow
class VCRWindow
```

---

## 6. NIST VoD System

The Java approach as outlined in the previous sections is implemented in NIST's VoD system. The system consists of a VoD server and two STUs. The VoD server is configured with MPEG 2 storage space and delivery capability. NIST's server is a Unix based workstation with 16 GBytes of external disk storage for content (i.e. users menus and MPEG 2 Transport Streams). The server's DSM-CC is implemented using CORBA 2.0 . The system is connected to the Internet and an

experimental optical fiber Asynchronous Transfer Mode (ATM) network, as shown in figure 5.

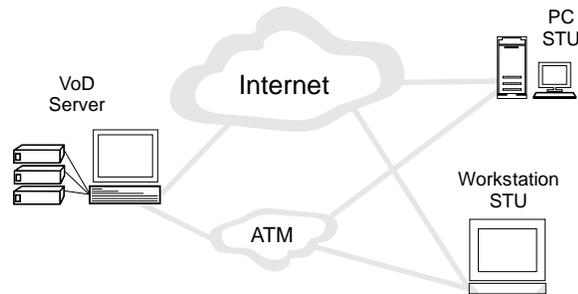


Figure 5. NIST VoD Testbed

The STUs run the VoD application and provide the capability of decoding and displaying MPEG 2 streams. Two STU systems demonstrate the Java based navigation and control. A complete software based STU<sup>6</sup> is implemented on a Unix workstation and a PC-STU uses a hardware decoder for MPEG 2 decoding.

A session with NIST's VoD server is established over the Internet with a WWW browser by simply selecting the VoD server's WWW home page. An applet that can be launched from the home page establishes the S2 control flow with the DAVIC based VoD server. If the STU does not have Internet capabilities but is a DAVIC compliant STU, the STU simply establishes an S2 control flow with the VoD server.

In both scenarios, after S2 flow establishment, the STU loads the server's *Init* program which, after execution, performs subsequent DSM-CC *File read* and *Directory open/resolve* operations to retrieve the VoD program—consisting of Java byte code and images—from the NIST VoD server. When executed on the STU, the retrieved byte code provides the user with menus for navigation and control. A simple menu could show the user the server's movie listing from which a selection can be made. When a movie is selected for viewing, an S1 data flow is established from the server to the STU. A distributed stream pump implements the DSM-CC *Stream* state machine which handles transport of MPEG 2 data to the STU. The stream pump is capable of transporting MPEG 2 data over the Internet protocols and over the ATM network by using ATM Adaptation Layer 5 (AAL5)[6].

New movies can be added to NIST's server by uploading them via the Internet (e.g. ftp) or DSM-CC protocols. Only some additional Java code (See the examples in the previous section) needs to be added to make the new movie appear in the VoD menus to provide the user with detailed movie information.

## 7. Summary

This article described the use of Java to implement a VoD application based on the DSM-CC standard. A short description of the DAVIC framework and a more detailed VoD model were introduced. The VoD program implements the user menus to provide navigation and control for the VoD service. Using the Java language to implement the VoD application provides several advantages. The first advantage is that Java is platform independent. Secondly, Java allows dynamic additions of objects to the runtime environment which reduces resource requirements on the STU. Thirdly, and most importantly, Java provides a means to integrate the still expanding popular Internet with DAVIC services.

An example is presented to demonstrate the feasibility of using Java to implement a VoD application. This VoD application is implemented as a demonstration prototype at NIST. The prototype is integrated with NIST's VoD testbed, which is connected to the Internet as well as an experimental ATM network. The VoD testbed provides the infrastructure for NIST's interoperability test laboratory.

## Notes

1. The Content Provider System (CPS) which is part of the DAVIC model is not shown here.
2. The Java interpreter itself is platform dependent.
3. Bandwidth requirements are not taken into consideration.
4. The base classes could implement MHEG-5 classes. However, a simple VoD application does not need the flexibility of an MHEG-5 engine. Using a proprietary set of base classes provides more than enough flexibility for the service provider. Only interoperability of the Java interpreter is required to ensure proper functioning of an STU with multiple vendor's servers.
5. The pseudo-code included is based on general Object-Oriented concepts.
6. See "<http://mace.ncsl.nist.gov/vod/mpeg2ts.html>" for more information

## References

1. DAVIC, DAVIC Specifications 1.0, December 1995.
2. ISO/IEC, Information Technology—Generic Coding of moving pictures and associated audio information, Digital Storage Media Command & Control, International Standard, ISO/IEC 13818-6, August 1996.
3. ISO/IEC, Information Technology—Support for Base-Level Interactive Applications, Coding of Multimedia and Hypermedia Information, Draft International Standard, ISO/IEC 13522-5, December 1995.
4. G. Cornell and C.S. Horstmann, Core Java, SunSoft, Sun Microsystems, Prentice Hall, 1996.
5. IONA, Orbix 2 Distributed Object Technology, Programming Guide, IONA Technologies Ltd., Release 2.0, August 1996.
6. ATM Forum, Audiovisual Multimedia Services: Video on Demand Specifications 1.0, January 1996.

Received Date

Accepted Date  
Final Manuscript Date

## Table of Contents: Volume ()

### Number 1

### Numbers 2/3 (Special Issue on Computational Learning Theory)

### Number 4

A VoD Application Implemented in Java .....	275
..... <i>Alex de Jong, Karen Hsing, and David Su</i>	
A Bayesian Method for the Induction of Probabilistic Networks from Data	
.....	
..... <i>Gregory F. Cooper and Edward Herskovits</i>	309
Learning Boolean Functions in an Infinite Attribute Space... <i>Avrim Blum</i>	373
Technical Note: First Nearest Neighbor Classification on Frey and Slate's	
Letter Recognition Problem .....	<i>Terence C. Fogarty</i> 387

Volume , No. ,

---

A VoD Application Implemented in Java .....	
..... <i>Alex de Jong, Karen Hsing, and David Su</i>	275
<hr/>	
A Bayesian Method for the Induction of Probabilistic Networks from Data .	
..... <i>Gregory F. Cooper and Edward Herskovits</i>	309
<hr/>	
Learning Boolean Functions in an Infinite Attribute Space .....	<i>Avrim Blum</i>
	373
<hr/>	
Technical Note: First Nearest Neighbor Classification on Frey and Slate's	
Letter Recognition Problem .....	<i>Terence C. Fogarty</i>
	387

---