JAVA — applications in Digital Audio Broadcasting

Antonio Barletta
Sony (Germany)

Digital Audio Broadcasting (DAB) is preparing itself to meet the new challenges of the digital era. The Internet, DVB, UMTS and DAB are all competing to provide flexible data services to millions of new users.

Through the use of Java language APIs, the DAB system is now able to offer flexible and dynamic solutions for delivering exciting new content to mobile users.

This article summarizes the potential offered by Java technology in the radio broadcasting environment, and concludes with a description of the Java DAB API framework, and the software architecture of the Java DAB platform.

"New challenges in the digital era …"

The last decade has seen a growing competition to deliver new digital information services to millions of new users.

The Internet, as a leading communication system, was – up to a few years ago – the only and ultimate channel for providing sophisticated data information services. Several internet technologies were developed and tested, leading to the introduction of new techniques for distributing data. Web technologies (e.g. Java, TCP/IP, XML and HTML) became the de-facto solutions for distributing data over the Internet.

Along with new tools, new flexible ways of creating applications were developed. When compared to the traditional Client/Server paradigm (based on data transmission) the **mobile code model** has shown enormous potential: not only is the data transmitted but also the logic for interpreting the data itself.

Nowadays, the increasing advances in network technology are pushing the information revolution towards new telecommunication systems: DAB, DVB (in the case of television systems) and UMTS (on the mobile phone networks) are now offering even more sophisticated digital channels than the Internet; side by side with the traditional (audio and video) content, new data services (such as EPGs, eCommerce, digital archives, games, etc.) can be delivered over these broadcasting and mobile phone networks.

A very aggressive competition is developing among these newer digital systems to deliver flexible, cheap and ubiquitous data services. In such a moving world, the objectives of the content providers are constantly being revised, so new solutions have to be implemented at a very fast pace.

"... And DAB?"

In these new scenarios, the DAB system has tremendous potentialities for providing both traditional and modern content in an advanced digital format.

In brief, DAB is a **broadcast wireless digital network** with the capability of delivering high-quality audio streams and digital data content to a wide range of personal devices: from desktop PCs to laptops, from PDAs to electronic kiosks and in-car units.

What are the advantages of DAB in comparison with the other digital systems, and why is it so important to extend the DAB system to include new data services?

Let us answer these questions by first describing the main features offered by DAB:

In conjunction with Panasonic, **Roke Manor Research** – a Siemens company – made history in September 2001 with the launch of the Pocket DAB – claimed to be the world's first handheld Digital Audio Broadcasting (DAB) receiver. Despite measuring just 50mm x 76mm x 24mm, the Pocket DAB includes a battery providing five hours of radio listening per charge and can display 10 lines of text or graphics.
http://www.roke.co.uk/

- **Broadcasting**

  DAB is a suitable medium for delivering wide-interest content (streaming audio programmes, as well as data services such as news, traffic reports, weather forecasts and financial information). It is cheaper than the UMTS wireless system – because of its high scalability (one DAB transmitter can serve an infinite number of wireless receivers) – and it is cheaper than the DVB system through its more flexible use of the frequencies allocated for small data-rate services. Multicasting on the Internet can only work for small numbers of users – and with a great abuse of the available resources.

- **Wireless**

  DAB is a wireless system; when used with other wireless systems (GSM, UMTS) to provide so-called *feedback channels*, it can provide tremendous interactive data services to mobile wireless devices: PDAs, in-car units, electronic kiosks etc. Portable devices are becoming attractive and powerful enough to receive specific wireless services. And the mobile capabilities of DAB make it the perfect system for developing location-based data services.

The **Psion** Wavefinder – an interactive DAB receiver for the PC. Currently, the multimedia features only work in the UK but we are told that international versions of the Wavefinder will be available soon.
http://www.wavefinder.com/frHome.asp?page=home.asp

- **Real time / best quality**

  The data transmitted using the DAB system (both high-quality streaming audio and data applications) can be updated in real time; The Internet and mobile phone systems can also deliver real-time audio/data streams – but, again, with a great abuse of the available resources, and they cannot guarantee high performance when there are very large numbers of users.

Along with these specific features relating to the audio distribution, DAB has offered from the very beginning the possibility of delivering supplementary digital data applications: *Broadcast Web Site* [1] and *Slide Show* [2] are two simple and flexible data applications for delivering Internet-like content over the DAB network.

In addition to these two data applications, several recent experiments have been successfully implemented and are currently being evaluated: TPEG [3], IP-Tunnelling [4], DANTE [5], DIAMOND [6] and XML-EPG [7] are all successful examples of how to exploit DAB (alone or in combination with other media) to provide data services.

The DAB community comprises several main interacting constituents:

- **content providers** (BBC, SWR, Swedish Radio, RAI, etc);
- **network providers** (Teracom, Deutsche Telekom, NTL, etc.);
- **receiver manufacturers** (Grundig, Bosch, Psion, Roke Manor Research, etc.);
- and, last but not least, a large body of **users**.

Developing the solutions, and taking into consideration the full needs of all these constituents, is very complex because several conflicting interests are involved.

- On the one hand, the content providers want to have a more flexible way of controlling the applications delivered, so that they can follow the needs of their users. Content providers want to be able to create new data services as fast as possible.
- On the other hand, the receiver manufacturers want to have a clear definition of the minimal requirements for DAB receivers, and they want to provide a flexible execution environment to compete with other digital platforms (PCs, TV set-top boxes, PDAs, mobile phones).
- Network providers want to design and build their DAB networks as quickly as possible for delivering the new service.
- The users want to experience sophisticated and modern interactive DAB applications.

How can we counteract these conflicting interests? The answer is to provide an open, dynamic and standard execution environment, with the capability of offering flexible tools, adapted to the design needs of current digital applications.

Java, as a technology, enables us to realize that answer.

Java under the hood: a brief introduction

Java is a software technology which was introduced to the Internet world by Sun Microsystems [8], as a result of the new forces and constraints being imposed on the production and distribution of digital data content on networked PC systems.

Abbreviations

API	Application Programming Interface	**MOT**	Multimedia Object Transfer
BWS	Broadcast Web Site	**NPAD**	Not Programme-Associated Data
DAB	Digital Audio Broadcasting	**OS**	Operating System
DVB	Digital Video Broadcasting	**PAD**	Programme-Associated Data
DVB-J	DVB - Java	**PDA**	Personal Digital Assistant
EPG	Electronic Programme Guide	**TCP/IP**	Transmission Control Protocol / Internet Protocol
ETSI	European Telecommunication Standards Institute	**TPEG**	Transport Protocol Experts Group
GSM	Global System for Mobile communications	**UA**	User Application
GUI	Graphical User Interface	**UMTS**	Universal Mobile Telecommunication System
HTML	Hypertext Markup Language	**XML**	Extensible Markup Language
I/O	Input/Output	**VM**	Virtual Machine

The same technology has been used for other platforms (embedded systems, mobile phones, TV) and nowadays the Java platform is implemented on a wide range of network appliances: from network servers to desktop PCs, from PDAs to mobile phones, from TV set-top boxes to smart cards.

Before describing Java in some detail, we will look at the main reasons for using such a technology.

What, in general, are the forces and constraints imposed on the production and distribution of digital content?

From the point of view of designing data services, a clear separation between the production, distribution and presentation of content is needed: every stage should be independent and should be given the maximum freedom.

This results in some basic requirements:

- **easy programming tools** for producing content in a fast and modular way (object-oriented);
- a **portable format** for distributing flexible and dynamic content (data and code);
- a programming environment **independent of the different platforms** (different OSs, different devices, etc.) in which to execute the applications.

What is Java and why is it a solution to these requirements?

Java technology resolves the problems listed above. The system comprises three main subsystems:

- an abstract machine specification;
- a language specification;
- a standard set of APIs.

The first of these subsystems specifies an abstract software execution environment – an abstract machine – that is independent of the specific OS or hardware support.

For example, Java VM defines [14]:

- a set of "opcodes" (a sort of abstract assembler language);
- a stack-based abstract machine;
- a runtime environment support for multithreading applications;
- a security framework for controlling the execution of downloaded code;
- memory management, etc.

This abstract software platform architecture (so-called **Virtual Machine**) permits the programmers to concentrate on designing the applications, without having to adapt them to different platforms. Linux, Windows 2000, Windows CE, VxWorks are only a few examples of possible platforms on which DAB solutions may be developed: they all support Java *(see Fig. 1)*.

The second subsystem specifies the Java language for developing portable Java applications. The Java language has the easiest learning curve among the most common programming languages, such as C and C++, and it maintains at the same time a very efficient and flexible programming environment, using a clear object-oriented approach.

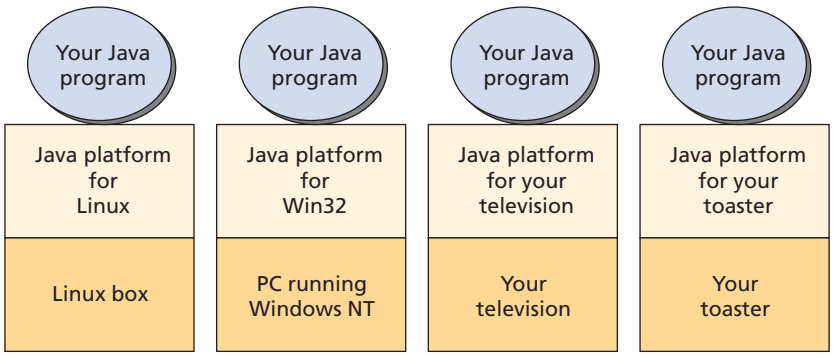


Figure 1
Java portability *(from reference [14])*.

Finally, Java is also a standard set of APIs. The Java platform specifies a standard core set of classes (classes grouped in packages) for controlling the runtime environment: I/O control, GUI access, network, etc.

By using Java as a core technology, different developers have been able to provide more sophisticated software frameworks: Aglets [9], Servlet [8], eCommerce framework EJB [8], JINI [8], etc.

One of these specific solutions is the so-called **applet model**. In brief, an applet is a small application that is downloaded through the network to extend the capabilities of the hosting environment.

An applet is fundamentally different from the classic client-server data protocol solutions, where a specific data format is defined to transport information to the user terminal. For example, the HTTP-HTML data protocol is a data client-server solution, where text formatted with HTML tags is transferred over the Internet, following the HTTP request-response pattern.

With the applet model, on the other hand, we transfer *dynamic* content: both the data and the software logic needed to decode and display the information is delivered to the users, independent of the host platform. Usually the applet extends the capability of a local application: for example if a content provider wants to deliver a video stream with the most recent video format (MPEG XX), he can transmit an applet with the new MPEG XX decoder, thereby allowing the new stream of data to be displayed on all the different platforms he can reach.

In the following section, we will see how we can extend the Java platform to include specific control of DAB networks, and how we can leverage the Java platform to provide new dynamic content.

A Java application framework for DAB

Although DAB is ready for hosting a technology such as Java, Java is not able to deal with the DAB system-specific characteristics.

As a result of the WorldDAB Task Force for Virtual Machine (TF-VM), an extension to the Java platform has been defined to provide the necessary control of the DAB system.

The work was divided into three main subsystems or packages, each dealing with a specific set of requirements *(see Fig. 2)*:

- Java DAB API package;
- Java DAB Runtime;
- Java DAB I/O.

See *Panel 1* (at the end of this article) for a description of the software architecture of a DAB platform.

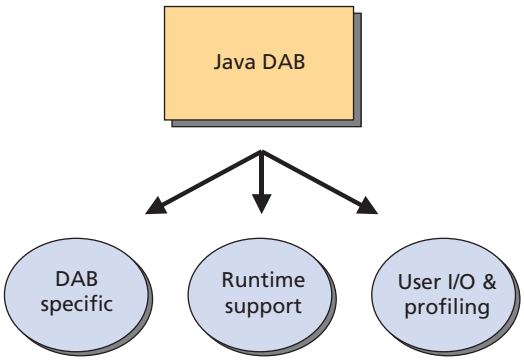


Figure 2
The three main sub-systems of the Java broadcasting extension

Java DAB API package

The first of these packages defines the resources of the DAB network and it specifies a simple transactional-asynchronous access model.

> *"The DAB package provides high-level access to the services of the Digital Audio Broadcasting (DAB) System. The package uses the Event-Listener pattern for the communication between the DAB system and the application. On top on this basic communication pattern a transaction concept is defined (e.g. to deal with ongoing events) [10]"*

In brief, the Java DAB API package is a collection of Java classes that model the DAB system; using this basic framework, the developers can easily produce applications or build up a more sophisticated framework for eBusiness using the DAB resources.

See *Panel 2* (at the end of this article) for a complete example of how the Java DAB API can be used.

Java DAB Runtime

The Java DAB Runtime specifies the runtime supports needed to provide a safe and reliable execution environment [11]. The Java platform partly provides this support, but we also needed specific extensions for DAB.

We divided these extensions into five subgroups:

- **DAB application model**

  This part defines a lifecycle for DAB Java applications, based on the Xlet model of JavaTV. This is the basic model for downloading small Java applications from the DAB network.

- **Control of Java applications**

  Here, we specify how an application is launched and how its state can be controlled.

  The model we used for an application downloaded through the DAB system is the same application model used in the DVB-J system – the so called **Xlet** model – with some additional elements specifically designed for DAB. The DVB-J Xlet framework is more suitable for a wide range of application platforms: alternative models – such as, for example, the applet – are specific to the browser environment. The Xlet model is a convenient minimum framework and it can also be used for controlling more sophisticated application types. For example, in the Java DAB prototype developed by the WorldDAB TF-VM, the Tetris game is an applet adapted to the Xlet framework *(see [11] and the screen-shot shown on the next page)*.

- **Security management**

  This part handles the security issues with regard to DAB Java applications: this is very important, especially in the case of remote downloaded code.

- **Resource management**

  The resource management provides mechanisms for sharing resources between different DAB Java applications.

- **Configuration management**

  This section deals with the handling of the internal profile (i.e. the profile information that is available to the application).

Java DAB I/O

Finally,

> *"The DAB User I/O package specifies the Java platform that should be supported for DABJava. It also defines the profiles for DABJava and the method for signalling the profile using the FIG0/13 User Application Type [13].*
>
> *"**DABJava** is defined as a User Application (UA) type within FIG0/13. The DABJava UA parameters "**Platform**" and "**Version**" are used to signal the DABJava profile or application environment. They are carried in the UA-specific part of FIG0/13 [12][13]."*

With the first one (Platform), we signal the execution environment needed to run the applications, while the second parameter (Version) flags the software compatibility of the DAB Package (with backward compatibility).

Two platforms have already been defined: **Standard Personal Java Profile** and **Network-enabled Personal Java Profile**.

Currently these two platforms are the most suitable for embedded devices, both in terms of the resources used, and in terms of their flexible programming environment: several implementations (TV set-top boxes, PDAs) are already available on the market *(see [12])*.

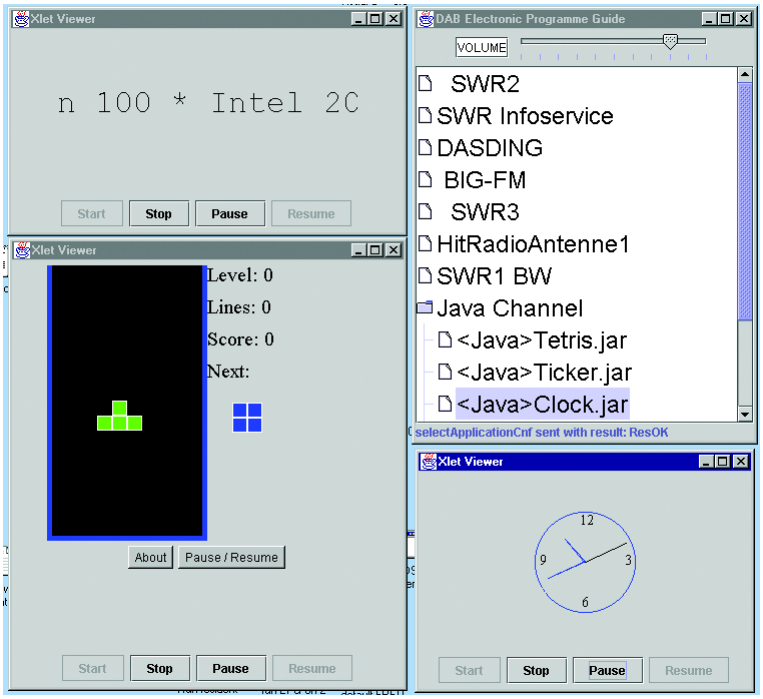
The Sony evaluation implementation of the Java DAB API, for example, uses a Personal Java profile which is available free from Sun Microsystems (about 4 MB), plus the specific Java DAB package (68 KB). Personal Java imple-

mentations are readily available on the market from various software houses, for different embedded platforms. (The Kada System implementaion, for example, is about 330 KB – *see* http://.www.kadasystems.com.)

In order to calculate the memory resources needed for implementing the Java DAB Runtime, we have to add the native OS memory footprint (including the DAB native protocol stack), plus the memory needed for the Personal Java, plus the DAB-specific Java class.

DAB Java applications

For validating the Java DAB API, three types of application were considered: an Electronic Programme Guide (EPG), a Share Price Ticker and a Game.



Example DAB applications developed during the Task Force VM trial.

These three types of application cover a wide range of possible application models and they can be delivered natively or downloaded via DAB. However, different security and resource issues must be considered and specified.

The **EPG** is an application that has full access to the DAB system, and controls the DAB terminal. The **Ticker** is a classic business data application, where three main functions are implemented: access to the data stream, decoding of the information and displaying it to the user. The **Game** is a fully-interactive application using the hosting I/O resources.

EPG

An EPG can be developed either as a native Java application or as a downloaded Xlet module. Content providers can deliver their own flavour of EPG and dynamically update its "look and feel".

The main functions of an EPG are, firstly, to access and control the audio and data stream and, secondly to provide a user-friendly interactive View to the user *(see Panel 2)*.

The EPG is also the most critical component in terms of security access. A downloaded EPG has access to all the DAB resources of the user's terminal: selection of audio, tuning, volume level, etc. This is an entirely new concept, brought about by the use of mobile code. However, the Java DAB API will also provide user access to these resources. As a possible example of the Java DAB API in use, let us imagine that the BBC sends a new EPG to a DAB receiver and that, whenever a user selects a BBC radio station, the sound level is automatically

Antonio Barletta obtained a degree in Electronic Engineering from Turin Polytechnic in northern Italy. This was followed by a Masters degree (MSc) in Telematics and Multimedia (his thesis was on Mobile Code technology and the broadcasting medium, DAB).

Three years ago, Mr Barletta joined the Sony research centre in Stuttgart, Germany, to work in the mobile multimedia group. Much of his earlier work with Sony was on the DANTE telematic project, studying the use of XML, GSM and DAB to provide in-car data solutions (the project was presented at the Berlin IFA in 1999).

Over the past two years, Antonio Barletta has worked in the WorldDAB "task force for Virtual Machine" and has developed some telematic applications using DAB on different platforms.

turned up (something more commonly associated with TV sets during commercial breaks). The Java DAB API provides a way of controlling this, thus enabling the user to avoid having to accept volume changes when switching between radio stations.

Ticker

The Ticker application normally controls a data stream provided by a DAB network, and decodes it in a proper way. The data decoder logic and the user interactive View are downloaded dynamically. Security permissions are limited strictly to a few DAB resources (for example, access to a specific data stream). Other examples of ticker-like applications are map viewers, TPEG decoders, etc.

Game

The Game is a fully interactive application. Usually such applications should access only limited resources (GUI, User I/O) in a protected manner. There are thousands of such applications on the Internet that can readily be adapted for use with the DAB system. Educational applets, games, animations and banners are only a few examples of the possibilities offered by an open platform such as Java (remember that Java is a portable technology!).

Java DAB prototype : the applications can run on both a laptop PC and a PDA platform.

What next?

The results obtained by the WorldDAB Task Force for Virtual Machine [1] will be forwarded to ETSI for standardization in the next few months. In the meantime, several organizations have been testing and adopting the Java DAB API for validating and prototyping new applications and business models, using the DAB network *(see photo)*.

Currently, a new Task Force within the WorldDAB forum is involved in specifying a mobile DAB terminal with extended capabilities – both in terms of integration with others networks (GSM, UMTS) and in terms of eCommerce solutions [2]: this task force is willing to use the Java DAB API for showing off the potential of the DAB system.

1. **The Task Force for Virtual Machine** was a technical group formed by technical members of the WorldDAB Consortium. For more details about the companies involved, please contact the WorldDAB manager, Mrs Julie Ackerman.
   ackerman@worlddab.org
2. For more details of the work of the Task Force for DAB/Mobile, please visit the WorldDAB website.
   http://www.worldDAB.org

Bibliography

[1] TS 101 498-2 v 1.1.1 (August/September 2000): **Digital Audio Broadcasting (DAB); Broadcast website; Part 1: User application specification** and **Part 2: Basic profile specification**.
ETSI: http://pda.etsi.org/pda/queryform.asp

[2] TS 101 499 v 1.1.1 (July 2001): **Digital Audio Broadcasting (DAB); MOT Slide Show; User Application Specification**.
ETSI: http://pda.etsi.org/pda/queryform.asp

[3] TPEG – http://www.tpeg.org/

[4] IP-Tunnelling – http://www.teracom.se/

[5] DANTE – http://www.worlddab.org/events/proceedings/zumkeller.pdf

[6] DIAMOND – http://www.ertico.com/links/5thfp/diamond/home.htm

[7] XML-EPG – http://www.worldDAB.org

[8] "The source for Java technology"
http://java.sun.com/

[9] "The architecture of aglets"
http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html

[10] Gil Muller: **DAB Package Specification, Version 1.7** (13.11.2000)
WorldDAB Task Force for Virtual Machine.
ackerman@worlddab.org

[11] Antonio Barletta: **DAB Java, The Runtime Package, Version 1.1** (15.11.2000)
WorldDAB Task Force for Virtual Machine.
ackerman@worlddab.org

[12] Tristan Ferne: **DAB Java, The User I/O Package, Version 0.9** (8.3.2001)
WorldDAB Task Force for Virtual Machine.
ackerman@worlddab.org

[13] ETS 300 401 v 1.3.3 (May 2001): **Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers**.
ETSI: http://pda.etsi.org/pda/queryform.asp

[14] Bill Venners: **Inside the Java 2 Virtual Machine**
McGraw-Hill / Osborne. Ref.: ISBN 0-07-135093-4
http://www.osborne.com/programming_webdev/0071350934/0071350934.shtml

Note from the Editor

In addition to http://www.worldDAB.org, another useful website for anyone involved in DAB is http://www.wohnort.demon.co.uk/DAB/ which gives details of DAB ensembles around the globe, and also lists the DAB receiving equipment that is currently known to be available. Mr Carey Taylor, who oversees this independent website in the UK, would welcome updates and corrections from anyone involved in DAB, worldwide.

Acknowledgement

The author would like to thank all the members of the WorldDAB *Task Force for Virtual Machine* for the work they have done on this project, and for the results obtained.

Panel 1
A software architecture view of a DAB platform

The introduction of Java technology will provoke basic changes to the software architecture of the DAB terminals. These changes will allow more flexible data solutions, following the changing requirements of the users' expectations.

In this brief section, we will illustrate the main architectural blocks added to the traditional software architecture for DAB data applications *(see the diagram below)*. Emphasis will also be put on the software components under control of the content providers.

The current DAB receivers have a very simple software architecture: a native operating system (Native OS on the diagram), or an embedded controller, which provides the basic control of the DAB transport layer (audio/data stream) to a set of data decoders (MPEG audio decoder, MOT decoder for PAD and NPAD data). Very basic applications provide access and control for, usually, the Electronic Programme Guide and data-decoding applications (Broadcast Web Site and Slide Show).

Nowadays, DAB platforms are being integrated in PDA OSs (such as Windows CE or Linux devices) or directly on desktop PC platforms. Despite the increased capabilities of the hosting platforms, the basic software structure remains the same: native OS and native DAB applications. In the current scenario, content providers can deliver only a specific subset of data applications, and they are strictly dependent on the receiver capabilities.

Adding support for Java on the receiver side means adding a software engine that abstracts all the differences between the specific platforms (embedded OS, desktop PC, PDAs). Such a universal software platform frees the content providers from delivering only specific content, and it frees the receiver manufacturers from updating their receivers when following the changing needs of the content providers.

Another positive advantage is the possibility to reuse native Java applications (see the diagram) on multiple platforms, reducing the risks of developing the same solutions each time, and increasing the reliability of the code used.

Let's go one step farther. In the Java DAB API framework, there is the possibility of delivering dynamic smart components through the DAB network to different receivers, independent of the receiver's capabilities.

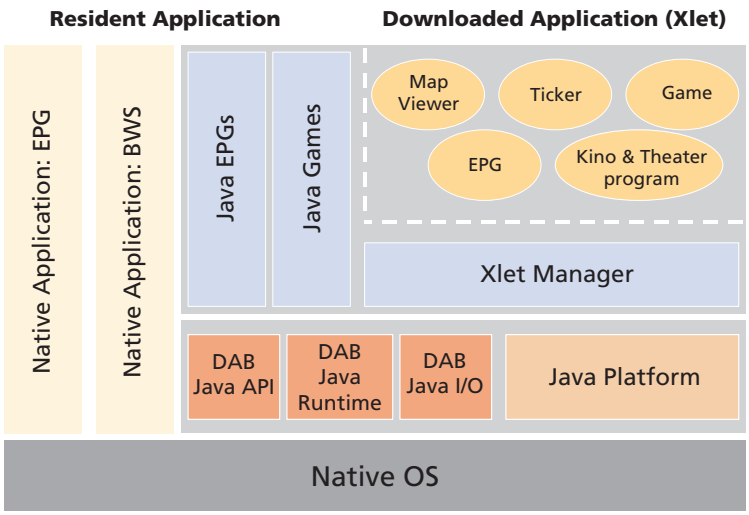
One application, the Xlet Manager, provides the support for downloading, hosting and executing software components. What are the real advantages of such new capabilities in the receiver? We'll explain it through the use of some examples.

Let's assume that a content provider would like to provide a new data service using a proprietary data format (XML, TPEG or MPEG-7). Using the Java framework, the content provider can safely download a new application directly onto the host platform, where the particular data format can be accessed, decoded and displayed. After one month, following the tremendous success of his data application, the content provider decides to compress his data, in order to increase the quantity of data delivered: the development team – by being free of licence fees – develops its own compression encoder.

What happens now? Quite simple – a new application, which includes support for the compression algorithm, is developed, tested and transmitted to the receivers: the receiver software capabilities are updated automatically.

(Using the traditional approach, for every new protocol format developed, the content providers would have to approach a standards committee for validation of each new protocol. They would then have to wait for the receiver manufacturers to provide support for the new services, and so on.)

While adding Java support to a DAB platform can relatively increase the architecture of the system, it can provide the users with a flexible modern environment in which to deploy dynamic new services.



Software architecture for a DAB platform.

Panel 2
An example of a DAB Java EPG application *(from [10])*

In the following we will show how to use the Java DAB interface for an EPG application. We will focus on the main steps for initializing and controlling the DAB system. Basically, we need only two main classes: a `DABListener` (usually on the application side, in our case the EPG class) and a `DABClient` (the main entry for controlling the DAB receiver – see the accompanying diagram.

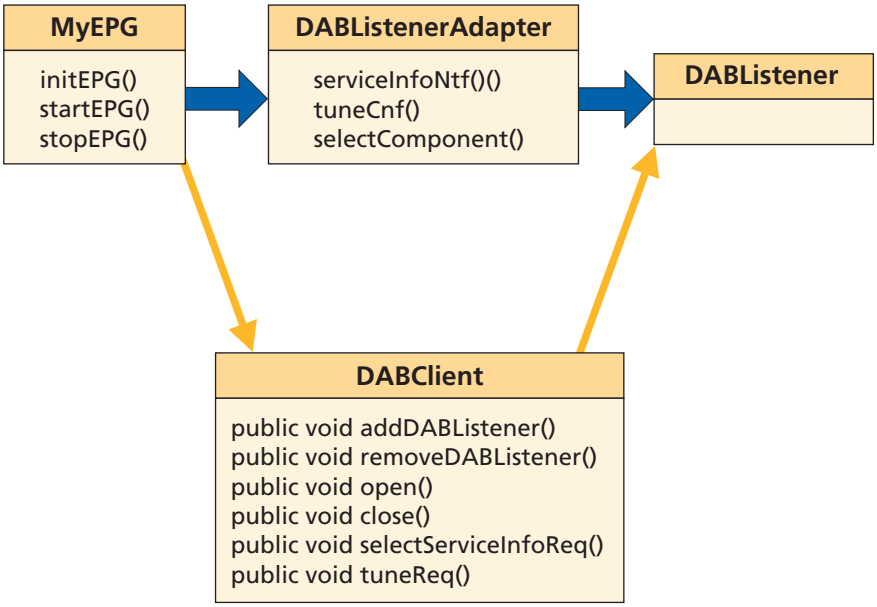


Figure A1
The classes of the EPG.

Before making any actions to the DAB receiver, we have to initialize it, and register a `DABListener` (or an adapter class) for the incoming messages: in our case, the EPG application class implements the listener interface directly.

This leads to the following initialization code:

```
public class MyEPG implements DABListenerAdapter {
     static public void main(String[] args){
          ...
          initEPG();
          startEPG(frequency);
          ...
          pause EPG();
     }
}
```

The implementation of the main `MyEPG` methods are:

```
public void initEPG(){
     DABClient dab = new DABClient();
     dab.addDABListener(this);

     try {
          System.out.println("Sending Open Request....");
          dab.open();
          System.out.println("...  Open Request sent");
     }catch(DABException e){
          System.out.println("Caught exception during open
          request:"+e.toString());
          }
} // end initEPG() method

public void stopEPG(){
     try {
          dab.close();
          dab.removeDABListener(this);
```

Panel 2 *(continued)*
An example of a DAB Java EPG application

```
        }catch(DABException e){
             System.out.println("Caught exception during open
             request:"+e.toString());
             }
        finally{
             System.exit(0);
        }
   } // end stopEPG() method
```

Notes:

- The `open()` method is synchronous, i.e. the application is blocked until the method returns;
- The registration procedure (`dab.addDABListener(…)`) should be done in relation to an `open()`, but absolutely before using any asynchronous methods (see the EPG example);
- The closing procedure is symmetric to the opening: first we close the connection to the DAB system and then we annul the registration as a `DABListener`.

After the initialization steps, we implement the basic exchange of messages for controlling the audio services in the DAB ensemble.

We will focus our attention on the following set of asynchronous methods:

On the DAB side:

- `dab.tuneReq(frequency, mode)`;
- `dab.selectServiceInfoReq(true,true,true,true)`;

On the `DABListener` *side:*

- `public void selectServiceInfoCnf(SelectServiceInfoCnfEvent e)`;
- `public void serviceInfoNtf(ServiceInfoNtfEvent e)`;
- `public void tuneCnf(TuneCnfEvent e)`.

With the events:

- `TuneCnfEvent`;
- `SelectServiceInfoCnfEvent`;
- `ServiceInfoNtfEvent`.

After tuning to a specific frequency, we register `MyEPG` for receiving notification messages about the available services on the DAB ensemble (`EnsembleInfo, ServiceInfo, ComponentInfo`). The usage of the information received by the `DABClient` is a task that is specific to the application. Here, we demonstrate it by selecting a particular audio component from a DAB ensemble.

```
public void startEPG(int frequency){
     try {
          dab.tuneReq(
               frequency, DABConstants.transmissionModeAutomatic);
     }catch(DABException e){
     }
     // to be continued
}

// DABListener interface
public void tuneCnf(TuneCnfEvent e){
```

Panel 2 *(continued)*
An example of a DAB Java EPG application

```
        int result = e.getResult();
        int tunedFrequency = e.getTuneFrequency();
        System.out.println(
             “Tune cnf received,
             result = “+Integer.toString(result)+”+
             Frequency = "+Integer.toString(tunedFrequency));
        return;
    }
```

A tune request message is sent for tuning to a specific frequency (in Hz), using a specific mode [1]. If it is successful, a tune confirmation message is delivered and the receiver is tuned to the requested ensemble [2].

```
    public void startEPG(){
        // continued
        try {
             dab.selectServiceInfoReq(true,true,true,true);
        }catch(DABException e){
        }
    }

    // DABListener interface
    public void serviceInfoNtf(ServiceInfoNtfEvent e){
        int notificationType;
        EnsembleInfo ensembleInfo;
        ServiceInfo serviceInfo;
        ComponentInfo componentInfo;

        notificationType = e.getNotification();

        switch(notificationType){
        case DABConstants.notificationEnsembleAdded:
        case DABConstants.notificationEnsembleRemoved:
        case DABConstants.notificationEnsembleChanged:
             ensembleInfo = e.getEnsembleInfo();
             // notify the Application of a Ensemble info
        break;

        case DABConstants.notificationServiceAdded:
        case DABConstants.notificationServiceRemoved:
        case DABConstants.notificationServiceChanged:
             serviceInfo = e.getServiceInfo();
             // notify the Application of a Service info
        break;

        case DABConstants.notificationComponentAdded:
        case DABConstants.notificationComponentRemoved:
        case DABConstants.notificationComponentChanged:
             componentInfo = e.getComponentInfo();
```

1. In the DAB specification, several modes are specified for the transmission of a DAB ensemble: some DAB receivers can automatically detect the specific transmission mode of an ensemble. In other receivers, such a parameter has to be done explicitly (cf. DAB specification).
2. Other more sophisticated tuning actions can be done using the `scanReq()` method (cf. DAB Java API).

Panel 2 *(continued)*
An example of a DAB Java EPG application

```
            // notify the Application of a Component info
        break;

        }
        return;
    }
```

The application uses `selectServiceInfoReq` for receiving information about all available DAB components (ensemble, services, components). After receiving a confirmation of the request, the `DABClient` will notify every change in the DAB signal information (additions to, changes to, and removal of an ensemble, services, and components). Specifically in this case, we ask to receive with the notification the information related to the particular info object (see the DAB Java specification for details).

The information is delivered to the application using a special event; the usage of the carried information depends on the application strategy.

The final step for our simple EPG is to select a particular audio component, assuming that we have collected all the information about the services and service components available for the selected ensemble. We assume that the user has selected a service component somehow and that the EPG has identified the selected component.

```
public void selectAudio(ComponentInfo componentInfo){
     if(componentInfo.getType() == DABConstants.componentTypeForegroundSound)
     {
          try {
               dab.selectComponentReq(
                    componentInfo.getId(),
                    DABConstants.selectionModeReplace);
          }catch(Exception _e){
          }
     }
}
// DABListener interface
public void selectComponentCnf(SelectComponentCnfEvent e){
     System.out.println(
     "Result"+Integer.toString(e.getResult()));

     return;
}
```

One final example of a Java DAB application running on a PDA platform, this time from **Etheraction**.
http://www.etheraction.com/