Support for Internet Information Services on Digital Audio Broadcast Networks

MPhil Thesis
June 2000

Duncan McPherson

University of St Andrews, Scotland
& IBM Research, Zurich

Declaration

I, Duncan McPherson, hereby certify that this thesis, which is approximately 23000 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

date .... signature of candidate ....

I was admitted as a research student in March 1999 and as a candidate for the degree of MPhil in March 1999; the higher study for which this is a record was carried out in the University of St. Andrews between 1999 and 2000.

date .... signature of candidate ....

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of MPhil in the University of St. Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date .... signature of supervisor ....

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any *bona fide* library or research worker.

date .... signature of candidate ....

Abstract

This thesis investigates the use of Digital Audio Broadcast (DAB) radio technology for the efficient distribution of World Wide Web Internet content to mobile wireless clients. Many popular web sites remain static for a day or longer, which makes them particularly suitable for this type of broadcast distribution. The design and development of a complete working reference system - DABWeb - is described.

DABWeb also incorporates a means for service providers to support user profiles and charging regimes. Transmissions are encrypted and customers are supplied with smart cards which decrypt information being received. In contrast to the Public Key style of cryptography, a symmetrical Remote Key algorithm - BEAST RK - is adopted. Bounds on the time needed for decryption operations to complete in order to cope with broadcast transmission data rates are derived.

Finally, DAB is a unidirectional broadcast standard so the usual Internet protocols which facilitate reliable transmission and user navigation on the Web are not available. These requirements are addressed by client-side caching and server-side scheduling schemes. The concept of *pseudo-interactive* Web surfing is introduced.

Preface

The DABWeb project has been developed within the context of the Pervasive Computing group, part of the Applied Computer Science department of the IBM Research Zurich Laboratory. The group is interested in the connectivity of mobile devices using *ad hoc* networking systems (spontaneous networking). This allows devices to meet "in the middle of the desert" and instantly become able to share resources that are unique to each. The group purchase of a DAB Transmitter and several receivers reflects its interest in supplying data to pervasive computing devices in a mobile and low cost manner.

The transmitter system hardware and all supporting software was supplied to IBM by Rhode and Schwarz [45]. Receiver systems have been sourced from both BOSCH [18] and Philips Electronics [41]. Additionally a special purpose Digital Signal Processor card, used to decode the data stream output from a DAB receiver has been supplied by the Fraunhofer Institute in Germany [51].

I would like to thank everyone at St Andrews who made it possible for me to spend time working at IBM, and for considering my request to head to Switzerland at an unusual time in my university career. I am very grateful to Colin Allison - my supervisor at St Andrews - for taking on board this project and carefully reviewing my work. Also to all the members of the DEAPspace project for their help in developing DABWeb and reading my thesis. In particular Dr Dirk Husemann, for his constant enthusiasm to make this project a success, Mike Nidd for his help with all things computer related and Christian Rohner for helping me understand the equipment I have been using.

Duncan McPherson
Zurich, January 1999

Contents

List of Figures

Chapter 1

Introduction

The rapidly growing demand for data-sharing using the Internet has led to the development of many new technologies designed to make efficient use of network infrastructure. Some of these technologies take into consideration the asymmetric bandwidth use by typical Internet users. For example, *web surfing* has a need for a high bandwidth download channel, but only a low bandwidth upload channel. This asymmetry exists because the upload channel is used to make download requests.

Another trend is the demand for a mobile wireless technology allowing Internet content to be browsed from anywhere using a small portable computing device. Wireless point-to-point data links to the Internet over GSM[1] or Satellite Telephones are expensive, and are not efficiently scalable, since many users in close proximity may absorb all available wireless bandwidth.

In this thesis I examine the use of Digital Audio Broadcast (DAB) radio technology to distribute selected World Wide Web Internet content, helping meet the demand for economical wireless web surfing. DAB provides a unidirectional data channel, which can be used to distribute encoded Internet content to many clients. As DAB technology provides no back-channel, web surfing users cannot select what to download and so surfing is no longer a fully interactive experience. However if the Internet content

[1]Global System for Mobile communications - Europe wide standard for mobile telephones

to be broadcast is chosen carefully to encompass many popular web sites, and then cached by each user's wireless client device, *pseudo-interactive* web surfing may still be possible and result in a satisfying user experience.

1.1 Broadcasting Web Content

Many popular web sites are modified only on a daily basis or less frequently. These static web sites are particularly suitable for broadcast channels instead of distribution via the HTTP [54] protocol over multiple point to point TCP [13] links. With HTTP, the network bandwidth requirements increase with the number of simultaneous accesses made to a web site. With DAB network technology a constant bandwidth is needed to transport a selection of popular web sites to an unlimited number of client systems.

The approach taken here is for service providers to broadcast each web site in its entirety for client systems to download. In this situation server-side scheduling of content is important to make sure client devices receive as many web sites of interest to a specific user profile as possible. Also on the client side an appropriate caching scheme is needed so that clients can intelligently cache web sites when they are received. Because of the wireless nature of DAB, continuous access to the network may not be possible, so clients should be able to cache web sites whenever they are received as a continuous background process on the client. When users browse using the wireless client they will be able to view only material contained in the cache of their device.

An encryption system which allows DAB web service providers to charge each client for receiving content is also presented here. With the ability to encrypt all or part of the broadcast data channel, a service provider can charge for the use of the keys used to decrypt each web site and so implement a *pay per view* web browsing system. The secret keys can be distributed on a tamperproof smart card - a service provider would supply appropriately prepared smart cards.

Users of encrypted data services could presumably benefit by the increased quality of content available since they are paying for the service. In the future such a charging system could also be used to distribute software and music.

1.2 Why use DAB?

Digital Audio Broadcast (DAB) has been designed as a replacement for current analogue radio services. In addition to superior *near CD* sound quality, the DAB network standard also makes provision for the broadcast of data services. At the time of writing DAB network deployment is occurring worldwide, and though there are competing digital radio standards - Worldspace in Africa via satellite [65] and Teracom[60] - DAB has emerged as the forerunner in Europe. Canada has adopted the DAB standard, and DAB is currently being evaluated for use in the rest of North America. If DAB services are extended to be Europe wide the price of a receiver unit should drop sharply as has happened in the GSM cellphone market. Currently the DAB standard has strong support from several large consumer electronics companies, who manufacture receivers and other network infrastructure.

DAB receiver devices are less complex and so more portable than equivalent Digital TV [12] devices, making them much more useful for a pervasive computing environment where they may be built into small computing devices, or used as a node in a Personal Area Network (PAN). DAB receivers are also available in many new car radios allowing small in-car clients to be designed which can download continuously. Another possibility would be for the car's client cache to synchronize with a user's palmtop based client cache over a high speed wireless connection, for example using Bluetooth technology [2]. This way the car would always download web information when available, and the user's portable client device would synchronize with the car's cache when within range.

1.3 The Reference Implementation

A complete DAB web content distribution system has been implemented here, named the *DABWeb* system. DABWeb can be used as a reference design for the creation of future commercial DABWeb systems, to hold data service trials, and to test content scheduling and charging strategies.

The concept of a *carousel* is used to indicate a means of scheduling broadcast data parts of a broadcast data stream that may be repeated more than once. For the purposes of DABWeb each carousel should be one web site - a group of web pages collected by Internet URL[2] address. Scheduling the same carousel to be broadcast more than once for example over the course of a day, allows more receiver clients to cache carousels of interest. In this way DABWeb can still cache carousels even if DAB radio reception is not continuous because of radio interference. Further research is needed to determine the optimum scheduling strategies for carousels of different content types.

1.4 The suitability of DAB Networks for carrying Data Services

The DAB standard was developed in the context of the Eureka 147 project [15] of the European Union. The key document describing this standard is published by the European Telecommunications Standards Institute (ETSI) [39] which details the composition of the digital data streams used in DAB and the physical requirements of a Radio Frequency (RF) DAB signal. Several other standards are in place for use by DAB broadcasters, in order that they might build up a complete transmission network where transmitters, audio sources, and intermediate multiplexers are physically separate.

[2]Uniform Resource Locator

The DAB standard provides great flexibility in the use of network bandwidth, including the ability to dynamically create data channels, taken from a fixed total broadcast bandwidth, for short lengths of time. This feature allows the implementation of data services, which can be made to *pop up* when required. Other features such as the ability to encode both audio services and data services at the same time make DAB technology lend itself well to creating a broadcast data network in addition to the audio network it was primarily designed to transport.

Several additional standards have been specified along with the core DAB standard document. Of greatest use to the development of DABWeb is the Multimedia Object Transport [34] protocol standard. The MOT allows for binary files along with associated parameters to be transmitted over DAB. The MOT standard does not provide any way for complete hierarchical directory trees of files to be transmitted. However it is an expandable standard, allowing this DABWeb specific aspect to be introduced.

The goal of my thesis has been to build upon the existing standards and create a new flexible standard allowing Internet content to be broadcast. The reference implementation of the new standard presented here helps show the viability of a commercial DABWeb service. The reference implementation also provides a technology demonstration showing the usefulness of future research which uses DAB as a medium for mobile data services. The next five chapters cover the following topics :

- The Related Work chapter summarises work by others which is closely related to DABWeb.
- Component Technologies is concerned with describing my overall design of the DABWeb system, and then reviewing each of the third party component technologies that form part of this system.
- DABWeb Design then details the functionality that should be achieved by a fully working DABWeb system, and then details my designs for both DABWeb client and server software systems.

- The Implementation chapter follows client and server designs through to a working DABWeb system, and ends with a summary of the current state of the DABWeb reference implementation.
- Finally the concluding chapter describes possible further research work on the existing DABWeb system, and uses for the software in its current state.

All the design and implementation work leading to the finished DABWeb prototype was done by myself between March 1999 and January 2000. Diagrams contained within this thesis are my own work, except where indicated otherwise. The project brief for DABWeb was provided by IBM Research. This brief included the specific choice of encryption algorithm and smart card systems to be used but only the general project description - to make use of DAB equipment for metered WWW content broadcast. IBM also specified that as much of the software as possible should be developed in Java.

Chapter 2

Related Work

2.1 Introduction

The concept of the bandwidth asymmetric DABWeb Internet content distribution system is certainly not new. Related systems may be found in use by theoretical studies attempting to optimize bandwidth allocation within a broadcast data channel, and also in the form of both research and commercial technologies used to disseminate data to a variety of clients including televisions, computer workstations, and mobile phones. Most of the technologies described in this chapter are asymmetric to some extent. In some cases asymmetry is a necessity because of the type of wireless broadcast technology being used, and in others the asymmetry results from attempts to make better use of the bandwidth available in existing network infrastructure.

2.2 The Boston Community Information System

Gifford et al [11] provide an early introduction to the use of dissemination to deliver data. An early application of this approach can be seen in the Boston Community Information System (BCIS) from MIT [11, 10] which was used to deliver information bulletins to client workstations equipped with radio receivers. BCIS was put on trial in the metropolitan Boston area for two years with about 200 clients starting in late 1982. The BCIS system made provision for both broadcast and two way interactive communication, and from the trial it was concluded that users valued both the broadcast and interactive styles of data delivery - showing the usefulness of broadcast technology for data dissemination. In the case of DABWeb web surfing of cached pages from broadcast carousels could be seamlessly integrated with the surfing of pages that must be requested from a server. Users could browse their cached DABWeb pages, which could optionally hyper-link to web pages not broadcast in DABWeb carousels. The user's web browser would then have to make use of a more expensive two way network connection to fetch the uncached pages. This would allow DABWeb users the same complementary mix of broadcast and interactive data delivery that was appreciated by users of the BCIS system.

2.3 The Datacycle Project

Broadcast *push based* technology has also been used to build database systems. The Datacycle project at Bellcore [17, 55] made use of high bandwidth optical fibre data channels to quickly disseminate a database to many clients which would then make use of a combination of hardware and software technology to perform searches and queries on the data stream. Datacycle provided all the usual functionality expected from a DBMS[1] and was able to provide concurrency control between changes made by multiple clients and provide consistency guarantees relating to data cached by each client. Client to server communication was provided by an *upstream network* backchannel.

[1]Data Base Management System

2.4 Broadcast Disks

Research at Brown University [46] describes the use of broadcast disk technology over asymmetric networks by information centred applications. A fully working prototype system is described, which makes the assumption of a broadcast downlink being present. The broadcast disk concept is similar to that of a DABWeb carousel, but is far more flexible having been developed with a wider variety of data dissemination applications in mind. The DABWeb carousel container format has been optimised specifically for web content and to a large extent be a backwards compatible evolution of the MOT protocol. Broadcast disks may be used to form a *non-flat* [25] scheduling model where different pages of content are broadcast at different rates per repetition cycle.

2.5 The problem of scheduling

Recently there has been a large research effort into the problem of generating broadcast data streams scheduled to achieve the best client performance where clients have little or no cache memory available. Jain and Werth [44] show this problem to be NP complete, even in simple cases where each client may only be interested in a small fraction of the whole data stream. Work has been done by Ammer and Wong [30, 31] to put the problem of content scheduling in the same context as bandwidth allocation problems. They describe a system of heuristic rules a broadcast scheduler may follow when choosing the ordering of information pages being broadcast. Several other researchers have also proposed heuristic solutions to broadcast scheduling [36, 6, 56, 47].

2.6 The Teletext System

The teletext system provides a traditional model for distribution of multimedia data over a broadcast channel. Both teletext and the closely related videotext system - which made use of a two way communication channel - have been in widespread use within Europe and Asia since the mid-80's. Theoretical results on client caching and broadcast scheduler design using these systems are described in a paper by Wong [22]. Several other studies have been performed in relation to teletext content scheduling and the latencies that users are willing tolerate. Ammar and Wong [30] have derived a formula to help ensure that content scheduled for broadcast will make optimal use of the broadcast bandwidth available in the absence of a client side cache. Their formula for optimum client side performance states that in teletext systems with no client side cache the bandwidth for each page broadcast should be proportional to the square root of the access frequency of that page. As the DABWeb system relies heavily on a client side caching scheme whereas teletext does not, a direct comparisons between DABWeb and the teletext system is not possible. However Wong and Ammer's formula may still be of use in the context of DABWeb to ensure maximum client side cache availability of a web page instead of access latency as with the cacheless teletext system. The use of a cache by the teletext system is studied in a paper by Ammar [29] describing the performance of a very simplistic caching system.

2.7 An MHEG Carousel Scheduling Simulator

The recent widespread introduction of digital television systems has also brought with it a new and improved teletext system, which features a user interface very similar to the world wide web. Fuhrhop et al [5] look at the use of the new style teletext system in transporting multimedia information pages in MHEG[2] format - a standard similar to HTML[3]. They describe optimisations of content for different geographical

[2]Multimedia and Hypermedia Expert Group
[3]Hyper Text Markup Language

regions to ensure that users have the minimum wait time when accessing a new page. A scheduling algorithm for specific weather report example content is presented from which access time statistics have been compiled. Their system does not make as much use of caching as DABWeb does, but the content dependent approach to scheduling might be useful in designing new carousel scheduling algorithms for DABWeb.

2.8 WebTV

Another system bearing a resemblance to DABWeb is Microsoft's *WebTV* [61] system which allows web pages to be broadcast along with a conventional analog TV signal, and received by a computer with appropriate TV tuner hardware. The WebTV system allows for interactive web browsing when used in conjunction with a conventional telephone line - the telephone line providing a back channel for web page requests. WebTV served as an early example of the convergence between Internet based information services and traditional broadcast based television channels, raising the question of whether television would evolve to become simply another application package on a computer desktop or whether the Internet would become absorbed by domestic television as another channel. Since the initial launch the WebTV[4] service has been expanded to use a satellite broadcast channel making the telephone line upload and satellite download combination highly asymmetric. This large extra download bandwidth allows WebTV subscribers access to a much more interactive and personalised television system where live TV programs may be paused, and rescheduled to each user's personal preference.

[4]The original WebTV product (1995) consisted of a set-top box with no local storage. Companies involved were Sony and Philips (end-system integrators); Excite (WWW search engine), Surfwatch (WWW filter); Concentric (ISP); IDT (electronics); Headspace (ambient music) and Progressive (audio/IP specialists). The company was bought by Microsoft at some point, and is now a brand name for a wider range of services and products.

2.9 "Being Digital"

One important feature of digital media delivery technology is user profiling. Negroponte [35] describes how the utopia of having content availability of 'anything anytime anywhere' may be what users think they would like, but in fact pre-selection on the part of the content provider is a much better scenario. Necroponte envisages a much more ideal system having a detailed profile of each user and the ability to ensure that the content most easily available to each user was especially relevant to them and delivered in a timely fashion. In the context of DABWeb, broadcasts could be made only to localised geographical areas along with local radio stations, and a client side user preference system developed to cache only those parts of the broadcast data stream of interest to that particular user.

2.10 IP Multicasting

Though most of today's Internet traffic is carried via point to point network links, there are protocols which allow multicast transmission of data packets. The experimental Internet Multicast Backbone - Mbone [24] - has been developed to facilitate the transmission of multimedia traffic on the Internet [49]. Multicast traffic is efficiently supported by co-operating routers which dynamically maintain a spanning tree of point-to-point links in response to user-initiated joins and removes. The effect is like that of a spectrum of broadcast channels from which a user can select.

Tools such as RAT [8] and VIC [50] are used for continuous media audio and video. Applications also use the Mbone for non-continuous media distribution. WARP uses multicast for efficient support of distributed atomicity in shared objects [3] and also for resource discovery [4]. WebConf [19] uses multicast for broadcasting HTML.

A key difference between the Mbone and DAB is that the Mbone is multi-directional - any member of a group can send as well as receive. However the Mbone offers no support for a mobile wireless client, as it requires an Internet connection to work.

The use of broadcast strategies for Internet content distribution has received much attention of late since traditional point to point Internet links have failed to provide adequate response latencies when coverage of election events or the Olympic Games is requested by millions of users simultaneously. Imielinski and Badrinath [57] survey data management problems in mobile computing, and argue the use of dissemination technologies as an alternative to point to point links in order to make systems highly scaleable.

2.11 Reliability

Broadcast technologies have also been used to allow reliable file transfer, with several protocols having been developed to achieve this end. These include the Multicast Dissemination Protocol [21] and the Adaptive File Distribution Protocol [43]. Both of these file transfer systems are combined with a back channel in order to allow a form of moderated ARQ [5]. Another system, Fcast [20] makes use of erasure codes instead of ARQ to allow reliable file transfer broadcasts. Erasure codes are a type of FEC[6] scheme which has been specifically designed for one-to-many computer networking protocols such as IP multicast. The basic idea is that k blocks of source data are encoded into n (greater than k) blocks of transmitted data in such a way that any k blocks of encoded data can be used to recover the original data. In this way a receiver can recover from up to n - k losses of good blocks.

FEC schemes are not often employed in the Internet, as the current generation of Internet protocols have evolved primarily for use within duplex networks where ARQ was a natural choice for achieving reliability in point to point communication. FEC systems are perceived as having significantly higher bandwidth overheads than ARQ based schemes, and rely on complex algorithms for data encoding and decoding which are costly to run in software. Traditionally the mainstay use of FECs has been in the

[5] Automatic Repeat reQuest (ARQ) protocols require positive acknowledgements (ACKs) and/or negative acknowledgements (NACKs) from the receiver to ask for retransmission of lost or corrupt packets from the sender.

[6] Forward Error Correction

telecommunications industry where high bandwidth data links make use of dedicated hardware to implement specific FEC schemes.

The use of FECs as software components in IP protocols may become more widespread in the future, with this goal having motivated some of the recent work on erasure codes by Rizzo [26]. Use of erasure codes in conjunction with multicast systems such as the Mbone could form part of a congestion avoidance strategy [27] where servers would be less likely to be swamped by repeat requests if corrupt data packets were received by large numbers of clients at the same time. Using an FEC scheme most of the clients would be able to repair the damaged packets themselves without having to request retransmission.

At present DABWeb does not make use of an FEC scheme, instead only making use of repeat transmission scheduling for reliable delivery, and enhances data availability to the user by downloading whenever possible and holding the results in a large cache. The inclusion of an FEC scheme could be of use to DABWeb in the future, in allowing greater reliability of the system in areas where radio reception was particularly bad, and the likelihood of clients receiving several carousel repeat transmissions containing the same damaged blocks is high.

2.12 Multiple Unicast

Several commercial systems make use of pseudo broadcast such as the Pointcast [42] system. Pointcast claims to make use of push based technology in order to distribute information bulletins, but in fact the client side of the system makes use of a two way network link to periodically poll a server for new information bulletins. The subscribing Pointcast user base can profile themselves to a central server, which will then be able to pseudo broadcast the information bulletins to specific groups who would find them of interest. Other similar systems include Marimba [32] and BackWeb [1]. As multicast Internet technology becomes more widely available Pointcast and similar systems could be made to operate in a truly multicast fashion.

2.13 Other DAB based data service systems

Studies have been made of the possible usage of DAB broadcast networks to support interactive Internet surfing using a different form of wireless technology as a back channel. The MEMO project [33] contemplates using DAB technology in this way where GSM provides a low bandwidth uplink channel which is combined with a high speed DAB download channel. This system then transports web content via the HTTP protocol. Another study by Ljungquist [23] looks at the possible use of Digital TV broadcasts as a downlink data channel, and GSM as an uplink channel in the same way as MEMO. Both MEMO and Ljungquist aim towards using a broadcast channel for distribution of user requested data, whereas DABWeb does not support user requests in its current form.

Rhode and Schwarz [45] currently markets a multimedia data server system, similar in concept to the DABWeb system. Their multimedia server is intended to be used with content tailored for broadcast (as with teletext) using the MOT. DABWeb technology sets itself apart from the multimedia server by the smart card based encryption system, and because it can broadcast standard web content with no special tailoring.

2.14 Alternative wireless technologies

Within the commercial world there is widespread interest in wireless devices which allow consumers access to the Internet at a low price point. However much of this work is only visible in the form of public technology demonstrations attributable to specific commercial institutions, and so are not easily referenced. Currently a large development effort is being made by several companies to allow users of GSM mobile phones to access a subset of the Internet. This is to be achieved using WAP (Wireless Application Protocol) enabled mobile phones which have recently been placed on sale throughout Europe. WAP phones rely on a two way wireless connection to provide for fully interactive surfing.

The mobility group at Rutgers [57, 58, 59] has studied the problem of conserving battery power on mobile wireless client machines. Its system focused on minimising the amount of listening time each mobile receiver client made to the broadcast data stream. Its work in this area [58, 59] shows the use of indexing strategies to determine a period of time in which the client need not listen to the broadcast, and so conserve battery resources. By this strategy clients use the indexes to determine the time distance until the next broadcast is made of a page of interest. A similar mechanism is supported for by the MOT protocol, and if time distance indexes were placed in a DABWeb data stream DABWeb clients could be made to behave in this way.

Chapter 3

Component Technologies

This chapter describes the hardware and software technologies used to build the DABWeb reference implementation. In some cases specific hardware and software solutions have been chosen for reasons of availability, for example smart card systems and the WebFS filesystem, both of which have been developed within IBM.

3.1 A DAB Transmission Network

The DAB Network depicted in figure 3.1 consists of two service providers who may generate audio, data or both types of content, multiple transmitter sites, and all intermediate DAB network hardware. Several services are multiplexed together before transmission at Radio Frequency (RF) as a single broadcast *ensemble*. Since the same RF can be used for an ensemble at each transmitter site, mobile receivers do not have to be re-tuned when they move between areas covered by different transmitters - as they must be with conventional analogue radio stations. This is because all transmitter sites in a DAB network are synchronized.

Central to the entire network is the multiplexer unit, which takes *encoded services* as input from each of the service providers and multiplexes them together to create an ensemble data stream. An ensemble is then output to each transmitter site for

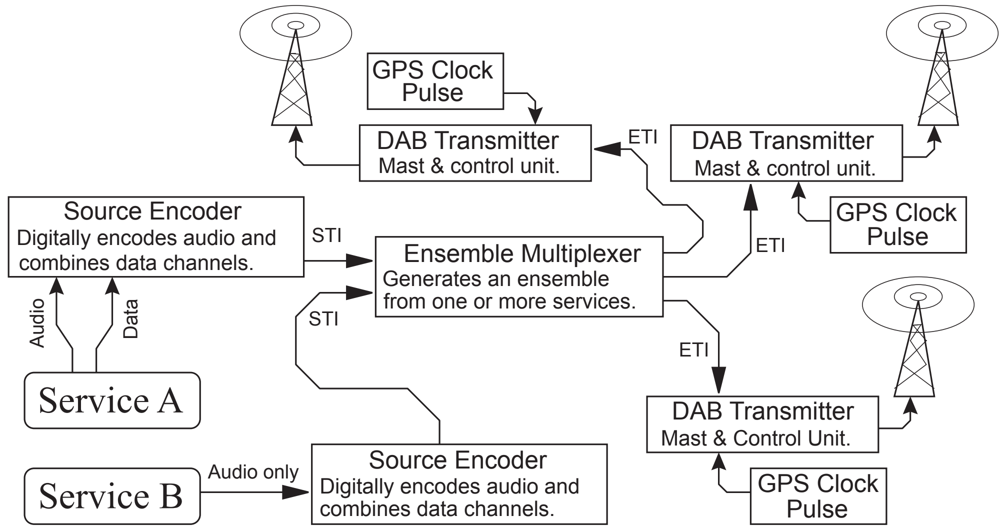


Figure 3.1: Illustration of a typical DAB Network

broadcast. Neither transmitter units nor encoder units need physically be located near the multiplexer. Instead they communicate using two well defined interfaces over satellite, or land based channels. The Service Transport Interface (STI) [53] is designed to handle all communication between a service provider's encoder unit and the ensemble multiplexer, and the Ensemble Transport Interface (ETI) [14] carries an ensemble data stream between the multiplexer and a transmitter control unit.

Because the commercial model for providing DAB broadcast services relies on the content providers and broadcasters being separate entities (a broadcast ensemble can carry up to 64 separate *subchannels* of content) each content provider will own an encoder unit, which is connected over a dedicated third party link to the transmission provider's multiplexer. This is the main reason for the STI and ETI being well defined, allowing each *radio station* to feed encoded service data to a transmission provider's network, making use of any vendor's encoder hardware that uses the STI protocol. Similarly any transmission unit may be used as long as it can receive an ensemble transported using the ETI.

The DAB standard allows a data channel to be encoded to fill an entire subchanel (replacing the audio) or be combined with the digitized audio data in a subchannel. Data streams which are combined with audio in a single subchannel are known as Program Associated Data (PAD). To further complicate the terminology, it should be noted that each subchannel output from an encoder is only an *encoded service* at that stage. Services are not associated with a name and subchannel number until they reach the multiplexer unit. Up to 64 encoded services are each given subchannel numbers by the multiplexer. The subchannels are then multiplexed together to create an ensemble. Subchannels within the ensemble may also be given textual *service names* individually or in groups.

A complete ensemble data stream is then output from the multiplexer to each transmitter unit. A transmitter unit modulates the 1.87Mbps ensemble to an allocated collection of radio frequencies spread across approximately 1.5Mhz using Orthogonal Frequency Division Multiplexing (OFDM). Finally the output RF signal is fed to the transmitter mast for broadcast.

Transmitters based in different locations (three are illustrated in the diagram) are kept synchronized using a highly accurate timing pulse from the GPS[1] satellite network. Broadcast synchronization is necessary for areas of coverage where transmissions overlap from two transmitter sites.

3.2 The DEAPspace project's DAB testbed

The DEAPspace project's DAB transmission system (see fig. 3.2) consists of four functional units, a PC computer used to configure the system, and a small antenna mast. All of the units are rack mounted, the only sensible way to contain the many interconnecting cables needed. The first two units are source encoders, then a multiplexer unit, and finally a transmission control unit which generates an output signal at RF. Currently RF signals from the transmitter unit are not fed into a secondary

[1]Global Positioning System

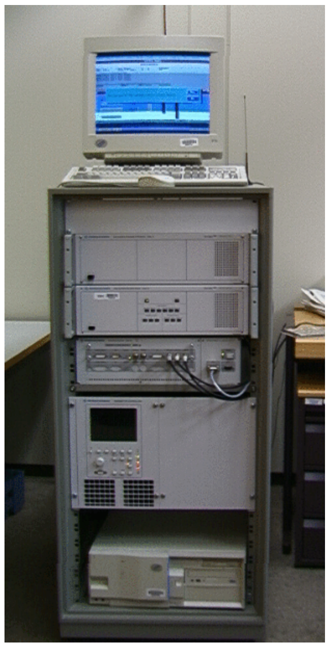


Figure 3.2: The DEAPspace project's DAB transmitter

amplifier, so the range of the transmitter is limited. In the laboratory, the transmission unit has been fitted with a GPS receiver which provides it with a regular clock pulse.

The two source encoder units allow analogue audio sources and digital data inputs to be fed into the transmitter system. One of these encoders functions as a master unit, and the other as a slave. Both contain two separate stereo audio/data encoder circuits each of which executes a software based codec. The codecs are configurable using an external PC which downloads new configurations (or even new codecs) to the encoder via an RS232 data link. Configuration parameters are available for the type of audio compression used, various acoustic options such as gain and equalization on the audio inputs, and the required output bandwidth of the encoded audio data.

Combined, the two units allow up to four subchannels to be encoded, each containing audio, data, or both audio and a PAD service. PAD data is combined with audio data by the codec, as input into an encoder unit using an RS232 data link. The exact number of PAD data bytes per frame of audio data in a subchannel is codec configurable.

The mapping of services to subchannels depends on the current configuration of the multiplexer. New configurations may be downloaded from an external PC, again over an RS232 data link. The application allows services to be mapped into an ensemble *time slice*, each service being allocated some of the total 1.87Mbps bandwidth of the ensemble.

The DEAPspace project also has access to three different types of DAB receiver device, and one PCMCIA card Digital Signal Processor (DSP) device, used to decode Radio Data Interface (RDI) [16] output from a receiver. An RDI output stream contains an entire ensemble, from which components may be extracted (e.g. a subchannel) using the DSP card. One receiver supplied by Philips outputs PAD data directly via RS232.

3.3 Overview of the DABWeb System

Figure 3.3 illustrates the components necessary for a complete DABWeb system, in which web content is harvested from the Internet and broadcast to multiple clients, which then cache the content so that it can be browsed later. The latest version of each web site is retrieved from the Internet by the web harvester module. The harvester caches each downloaded web site, until scheduled for broadcast.

Before broadcast, web sites must be encoded into carousels and (optionally) encrypted. With encryption a secure channel is needed to distribute secret keys between the server, and each client. Encoded carousels are usually scheduled for broadcast more than once, in the hope that they may be received completely by many clients. Each client decodes, decrypts and caches carousels for later retrieval by the local web browser.

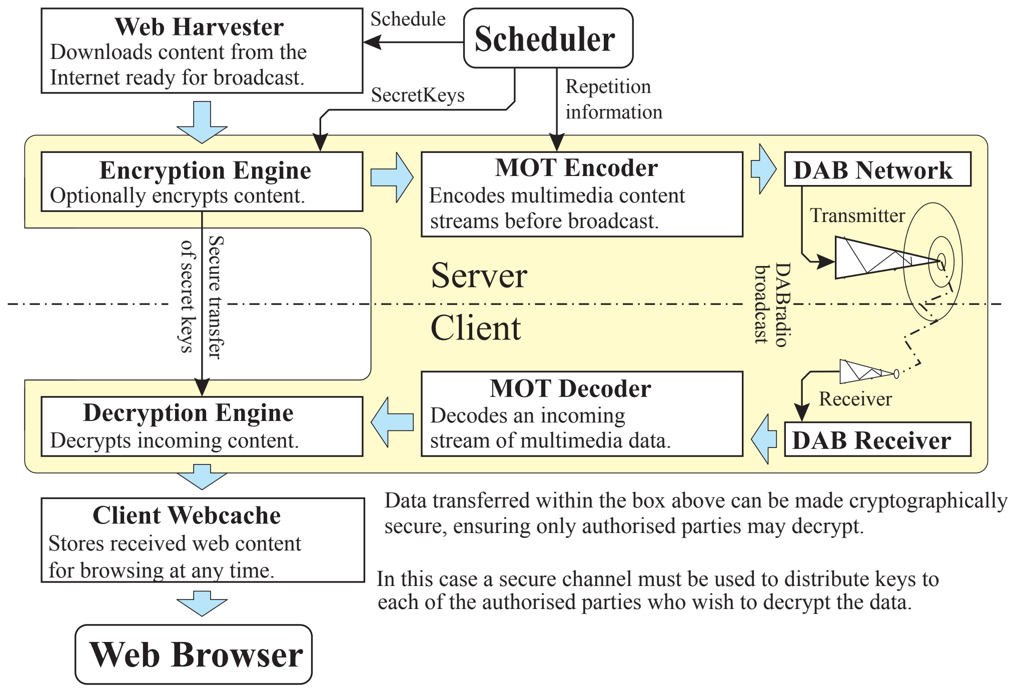

Web Harvester
Downloads content from the Internet ready for broadcast.
Schedule
Scheduler
SecretKeys
Repetition information
Encryption Engine
Optionally encrypts content.
MOT Encoder
Encodes multimedia content streams before broadcast.
DAB Network
Transmitter
Secure transfer of secret keys
Server
Client
DABradio broadcast
MOT Decoder
Decodes an incoming stream of multimedia data.
Receiver
Decryption Engine
Decrypts incoming content.
DAB Receiver
Client Webcache
Stores received web content for browsing at any time.
Data transferred within the box above can be made cryptographically secure, ensuring only authorised parties may decrypt.
In this case a secure channel must be used to distribute keys to each of the authorised parties who wish to decrypt the data.
Web Browser

Figure 3.3: The components required for a DABWeb system

3.4 The MOT Protocol Stack

The Multimedia Object Transport (MOT) Protocol Stack [34] has been developed by ETSI to provide a standard for the broadcast of multimedia data streams as components of a DAB ensemble. It is being used in some commercial data service trials at present [63] and has been developed by a subgroup of the engineers responsible for the original DAB standard [39]. The MOT standard integrates seamlessly with the core DAB broadcast standard, and is extendible for use with web based multimedia content. Because of this it has been chosen as the method of encoding a broadcast DABWeb data stream.

The protocol stack makes provision for repeats in the data stream, so that broadcast content can be scheduled in carousels of data, each carousel being broadcast several times to ensure that receivers have the maximum chance of receiving that carousel. Also catered for by MOT is fragmentation of the data stream into units optimized for receiver side caching. This is necessary so that receivers can hold incomplete snippets of a carousel in a cache until the missing parts of the carousel are broadcast again, and so the carousel can be completed. Repetition overcomes the problem of DAB not providing a reliable physical layer when continuous reception is not possible. If repeats are scheduled carefully, the probability of individual clients receiving more complete carousels increases.

Issues that must be dealt with in caching parts of a data stream are the ability to distinguish between fragments of an incomplete web site (so they may be cached and later retrieved), and time indications for the next repeat. When a cache manager knows how much time will pass between repeats, policies can be implemented to make efficient use of cache space.

The MOT protocol stack allows for the addition of parameters for time delays and names, and also allows new application specific parameter encodings to be added to a data stream. In the case of an encrypted data stream, extra parameters could be defined to specify encryption type and hold a session key if needed.

The MOT standard also specifies that parts of the data stream may be tagged with application specific parameters. This provides a flexible mechanism whereby tree structured web site data may be mapped onto a serial MOT data stream for broadcast. With an appropriate mapping scheme, web site files broadcast as parts of an MOT stream can be linked using special parameters, so that a receiving client can then map the files back into a tree structure in a local file cache.

3.4.1 The MOT Session layer

MOT Objects (see fig. 3.4) form the top *session layer* of the MOT protocol stack. Each object contains a data area and zero or more associated parameters. The MOT allows objects to be grouped together with a single numeric identifier if they are related for application purposes. Objects are identified within these groups by their transport identifier, which must be unique within the group.

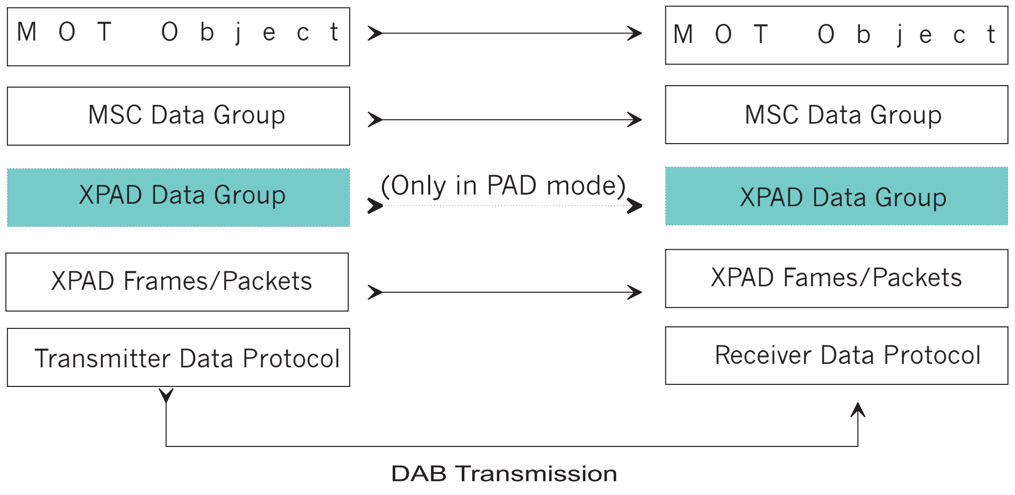


Figure 3.4: The MOT Protocol Stack (as used for both PAD and Packet modes)

Directly below the session layer comes the data group *transport layer*. Each MOT object is split into two or more Main Service Channel (MSC) data groups. Each MSC group bears the object's transport identifier along with a segment number counting up from zero (the first data group of the object). Data groups are optionally checksummed with a CRC, and contain a four bit continuity index allowing a receiver to check whether it missed any data groups in a transmitted sequence.

3.4.2 Transmission and re-assembly of MOT Objects from data groups

Each MSC data group is a uniquely identifiable sub-unit of a specific MOT object. When errors occur during reception of an object, that object can be kept in an incomplete form by the receiver until it is repeated again, and all data groups of the object are successfully received. Each data group contains a counter in its segment header indicating how many times it will be repeated in the future. In addition each object optionally contains a parameter to specify how many seconds in the future it will be repeated, so if some parts of the object are successfully downloaded a receiver will know how long it should wait for a repeat of the remaining parts.

Uniquely identifiable data groups also allow several objects to be transferred concurrently by multiplexing their data groups together in time. This is described in more detail in Chapter 4 along with other details of the DABWeb receiver client design. Concurrent file transfer may have useful implications for scheduling policies.

3.4.3 Ensemble bandwidth allocation for MOT data streams

In general two modes of transport are possible for MOT data streams :

1. Packet Mode where the MOT data stream occupies an entire DAB subchannel
2. PAD Mode where the MOT data stream shares a subchanel with audio data

With PAD mode transport only 54 data bytes may be inserted per audio frame, at the rate of one frame per 24 milliseconds. This provides a total bandwidth of approximately 2.19Kb/s with the available DAB testbed.

When Packet Mode transport is used, the bandwidth of the subchannel in question is dynamically configurable from the total 1.87Mb/s, by the multiplexer responsible for generating the ensemble. It is even possible to dynamically create *pop up* subchannels after reducing the bandwidth allocation of existing subchannels.

For example during news audio broadcasts where the audio quality required is lower, the bandwidth used by the news audio subchannel could be halved, and a new subchannel created to absorb the freed ensemble bandwidth. This new subchannel could carry an MOT data stream while it existed, and then disappear when the news finished.

3.4.4 The MOT in Packet Mode

A transmission frame from a complete DAB ensemble contains three specific components :

1. Synchronization Channel

2. Fast Information Channel (FIC)
3. Main Service Channel (MSC)

Depending on the transmission mode chosen (there are several defined) one or more of each part may comprise a complete transmission frame. Audio / Data information is always contained within the MSC part of a transmission frame with the other parts being used internally by the DAB system. Each MSC group from an ensemble transmission frame corresponds to one DAB subchannel. The format of the MSC part is exactly the same as that used by the MOT system to encode data groups (segments of an MOT object). For this reason transmission of MOT objects in packet mode (occupying a complete subchannel) should be easy in theory, as the same encoded MSC data groups can be placed directly into a subchannel with no further encoding. Within an ensemble transmission frame the MSC groups are further subdivided into packets, which can be reassembled by their sharing of a DAB network packet address.

3.4.5 The MOT in PAD mode

When MSC data groups are transported in PAD mode, as part of an audio subchannel, each data group must first be split into one or more XPAD frames. These frames are then combined with the audio data by a DAB source encoder unit. Each XPAD frame has a content type and frame length indicator. Many different content types are defined, as a PAD data stream is not designed exclusively for use by the MOT (only types 12-15 and 0 contain MOT data).

The first XPAD frame part of an MSC group must be a typed *group length indicator* giving the complete length of the MSC data group to follow. Immediately after the group length frame - and no other frame type can interrupt - is an *MOT group start* XPAD frame. After the start frame, the remainder of frames making up the MSC group are transferred as *MOT continuation* typed XPAD frames.

Since each XPAD frame is typed, it is possible to include other types of content between MOT data, as long as the length indicator frame is immediately followed by a group start frame.

Each frame can be individually checksummed using CRC16 so corrupt frames will be detected. However the only way to detect whether one or more complete frames have been missed is via feedback from the DAB receiver unit. Should several frames go missing, including the length and start frames of the next group, an MOT stream decoder would still notice when the checksum of the received MSC Data Group was incorrect.

3.5 WebFS (IBM Zurich Laboratory)

The WebFS file system has been developed within the DEAPspace project by Patrick Zwahlen [38]. It is used by DABWeb to harvest web sites from the Internet and provide server side caching of web sites before they are encoded into carousels for broadcast.

The WebFS system allows a non-volatile cache of data scheduled for transmission to be held with the minimum of active cache management by a DABWeb server. Some knowledge of how WebFS downloads web sites from the Internet is required by the main DABWeb server though, to ensure that web sites are cached in their entirety by WebFS. Other than this WebFS provides an excellent solution to the problem of keeping a DABWeb server continually running when remote Internet sites become inaccessible due to Internet network problems. In addition the caching of entire web sites on a local disk allows the DABWeb server to start up quickly in the case of software failure, as there is no need for time consuming data downloading from the Internet on restart.

Prompting WebFS to begin downloading new web content into its cache can be managed by a low priority server thread. The WebFS process itself could even run on a separate machine in a local network with a shared filesystem. This would reduce the processing load on the main DABWeb server machine(s).

3.5.1 WebFS Functionality

The WebFS system creates its own *virtual web* Network File System (NFS), allowing the web to be mounted just like any other NFS file system under UNIX, for example:

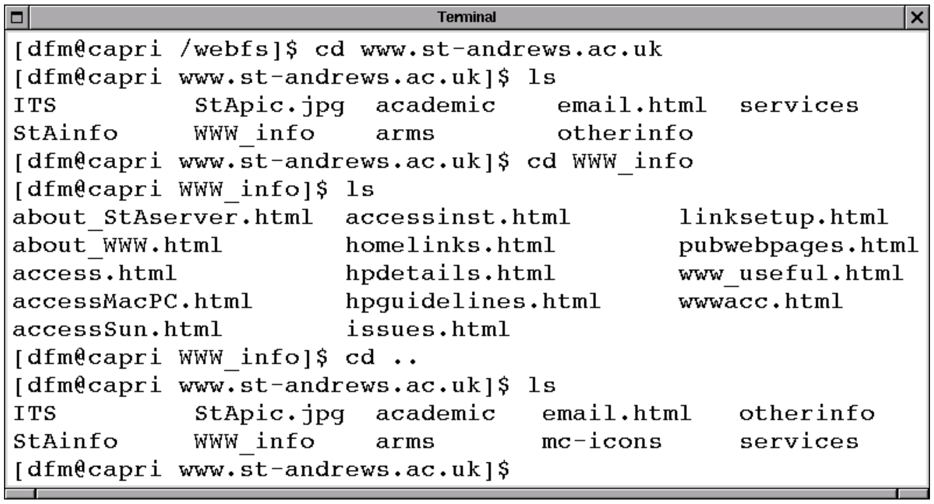
```
cd /webfs/www.st-andrews.ac.uk/
```

After this command has been executed WebFS would then allow the user to move around the directory structure of the web site `www.st-andrews.ac.uk` as if it was held on a local disk. Files may be loaded and copied to a different local filesystem, but as WebFS is read only they cannot be altered.

There are some technical considerations with WebFS, stemming from the fact that a hierarchical file system is only able to represent a subset of the web. Many web sites on the Internet are not presented in tree form, notably those dependent on CGI scripts to generate content. At present WebFS simply ignores these. In relation to DABWeb this should matter very little, as those web sites that may be encoded into a tree based data structure are also particularly suitable for easy broadcast, and caching in a DABWeb client's local hierarchical filesystem.

One consideration when downloading web trees using WebFS is that on mounting a particular web site, WebFS is only able to find the contents of a directory by intelligently parsing the HTML files it knows about, starting from files in standardized places, for example `index.html` is usually the entry point in the root directory of a web site. Because it is not possible to estimate how long this will take for a whole web site, WebFS performs a depth limited search of the complete web tree. This approach allows WebFS to access the Internet in real-time with further parsing being performed *on the fly* as new directories are accessed. As soon as WebFS has discovered the location of a file in a remote web site, it will download that file and cache it locally. If WebFS was to try and discover an entire web site directory tree structure and download all of that site's files in one operation - as a traditional web site downloading robot would - the WebFS user could experience a long delay when initially changing directory to a web site using WebFS. The use of a depth limited search allows WebFS to be used interactively as a command line tool, and as such allows standard UNIX file search tools to be executed on a web site.

A side effect of the depth limited search approach is that some directories and files in a web site tree may not become visible until all files in the site have been parsed. All parts of a site that have been visited are cached locally by WebFS (on the machine hosting the WebFS NFS server) and only when every HTML file has been found and parsed can a WebFS client be sure that it is looking at the most complete web site tree available. Even when all HTML files have been parsed, only files that are linked together in the web site will become part of the tree. Files that have no link to any part of the web site, but are available on the remote web server will remain hidden.



```
[dfm@capri /webfs]$ cd www.st-andrews.ac.uk
[dfm@capri www.st-andrews.ac.uk]$ ls
ITS          StApic.jpg  academic    email.html  services
StAinfo      WWW_info    arms        otherinfo
[dfm@capri www.st-andrews.ac.uk]$ cd WWW_info
[dfm@capri WWW_info]$ ls
about_StAserver.html  accessinst.html      linksetup.html
about_WWW.html        homelinks.html       pubwebpages.html
access.html           hpdetails.html       www_useful.html
accessMacPC.html      hpguidelines.html    wwwacc.html
accessSun.html        issues.html
[dfm@capri WWW_info]$ cd ..
[dfm@capri www.st-andrews.ac.uk]$ ls
ITS          StApic.jpg  academic   email.html  otherinfo
StAinfo      WWW_info    arms       mc-icons    services
[dfm@capri www.st-andrews.ac.uk]$
```

Figure 3.5: Viewing `www.st-andrews.ac.uk` with WebFS

Figure 3.5 shows WebFS running on the DEAPspace project Linux server *capri*. The user is accessing WebFS from a terminal window, and the first time the contents of the web site are viewed only nine directory entries are visible. After viewing one of the sub directories and then the root directory again, more files have become visible. Here the parser managed to find more root directory links when it parsed HTML objects in the sub directory. Taking such limitations into consideration WebFS still provides an excellent method of collating web content for DABWeb broadcast, with the WebFS server fully handling the task of collating web sites into web trees suitable for transmission.

3.6 Security Considerations for DABWeb

The BEAST RK [48] encryption protocol is used to encrypt DABWeb content, with each user requiring a smart card in order to decrypt BEAST encrypted content. The three main components of this encryption system are as follows :

1. The host computer based encryption and decryption software with secret key management functions.
2. The JavaCard smart card system, and on-card *applet* software. Smart card applets are not the same as web page applets, being designed specifically for smart card devices.
3. OpenCard [62], the smart card - host communications layer which provides a high level interface for software running on a host computer to communicate with software running on a smart card.

The IBM JavaCard smart card system [52] contains a very efficient on-card implementation of a Java Virtual Machine (JVM) and a *feature rich* on-card Operation System (OS). In particular the JavaCard OS contains many cryptographic functions, including the SHA-1 secure hash function which is one of the core components in the BEAST RK algorithm. Each JavaCard has the ability to hold several executable Java applets, each of which can support one or more application specific functions. Several different commercial implementations of JavaCard are available from IBM and other vendors, with differing specifications. Today's typical JavaCard contains between 8 and 32KB of EEPROM, 256bytes - 4KB of RAM, an 8 or 16 bit CPU and 32KB of ROM memory which holds the JavaCard OS and the on-card JVM.

3.6.1 Encryption of DABWeb content using BEAST RK

BEAST RK is a block cipher symmetric encryption algorithm which can be used to encrypt arbitrary sized blocks of data. The cryptographic security of BEAST is provable - as discussed in the next subsection - and it has been designed to allow the decryption of multimedia data streams in real-time.

The BEAST RK algorithm specifies the use of three secret keys to encrypt multimedia content blocks, along with a session key, randomly generated for each block. These secret keys are shared with untrusted DABWeb clients who would like to able to decrypt content. The session key is transported along with each MOT Object, with the content of each encrypted object having a unique session key.

Since BEAST RK is symmetric, where the same secret keys must be used by both parties, and DABWeb clients are untrusted, the secret keys must be provided on a tamperproof smart card device. The smart card solves the key distribution problem using its own on-board computing capabilities which allow cryptographic algorithms to be executed. The smart card handles all computations directly associated with the secret keys, never divulging the keys to the untrusted client. Keys are uploaded to each client's smart card by the DABWeb server, and then smart cards are sent, for example by post, to each client.

3.6.2 The cryptographic security of the BEAST algorithm

Because BEAST RK uses a unique random session key to encrypt each block, the untrusted client should be unable to perform any chosen ciphertext attack on the smart card in order to discover the secret keys. Analyzing BEAST RK, Lucks [48] holds that each block's unique session key makes the BEAST RK algorithm provably secure against such attacks when the attacker has many examples of both the cipher text and the corresponding plain text. In his claim about the cryptographic security of BEAST, Lucks makes the proviso that the building blocks used to make up BEAST must be provably secure.

In the case of DABWeb, chosen cipher text attacks can easily be performed on the smart card *applet* software, giving the attacker an unlimited amount of cipher texts to choose from.

3.6.3 The IBM JavaCard System

A computer host accesses the JavaCard by means of a smart card card terminal. Two possible types of terminal are available, one which handles contactless communication with a smart card by means of a short range RF link and powers the card by emitting an electromagnetic wave. The second type must make electrical contact with gold plated connectors on the surface of the card, and as such requires the smart card to be inserted into a slot. Contact based smart card terminals are advantageous as they can supply the card with more power and so increase the card's computing capabilities.

A card terminal will inform the host that a smart card is present, so that card-host communication can begin. When a smart card is first powered up by the card terminal (smart cards contain no on-board power supply) it will send an Answer To Reset (ATR) signal to the terminal, which informs the host what type of card is present. After this, subsequent card-host communications follow a set of discrete steps :

1. The host sends a command with optional parameters to the card.
2. The card receives the command completely, and then takes the appropriate action.
3. Once the card has finished processing, a result is returned in its entirety. Results may include one or more parameters, or be a simple boolean indication that the requested on card function has succeeded or failed.

The three steps described above constitute a complete card-host communication session. However data may be made persistent between communications to a card applet stored in the card EEPROM memory even when the card is powered down. By default all Java objects which are created as class member variables persist in this way, as do those created on the JavaCard applet's heap. Other objects which are created on the applet's stack are transient, and are lost between different host calls to applet functions. In the event of a power down (the smart card is removed from the card terminal before returning a result) the transient variables will also be lost, and the card cannot continue its current execution path.

The IBM ZRL JavaCard system follows the standard ISO 7816 [37] host to smart card communication protocol.

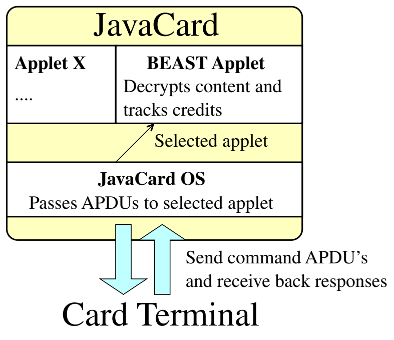

JavaCard
Applet X
....
BEAST Applet
Decrypts content and tracks credits
Selected applet
JavaCard OS
Passes APDUs to selected applet
Send command APDU's and receive back responses
Card Terminal

Figure 3.6: Overview of a JavaCard with two applets installed

All data exchanged between the host and card takes the form of Application Protocol Data Units (APDUs). Each APDU is simply a byte array, with formats being specified for both command and result APDU's. The ISO specification also defines simple APDU responses to indicate whether the last command was successfully executed by the card. For more complex responses, the specification defines how different data types should be arranged in the response APDU byte array.

The first communication between the host (running OpenCard) and a newly powered up JavaCard is the ATR which the card transmits. When the host receives the ATR byte array the lowest software layer of OpenCard is able to recognize the card type as being JavaCard. Next OpenCard must select which JavaCard applet it would like to communicate with (see figure 3.6). This is done using a JavaCard OS *select* APDU.

Each applet on the card has a unique identifier number known as the Application Identifier (AID). The AID is encoded in two parts, firstly a universally unique company identifier, assigned to the applet's developer, and also a number which uniquely identifies the applet, assigned by the developer. This allows any combination of applets that the card can accommodate to coexist.

A significant advantage of this type of multi applet card system is that a BEAST applet can coexist with a user's credit card applet, or along with other multimedia and e-commerce applets. With such a system, users need only make use of one smart card for all their e-commerce, DABWeb and other smart card needs, making swapping smart cards a thing of the past.

3.6.4 Security of data held on a JavaCard

JavaCards are considered physically secure computing devices providing only software access to card functions, and data held within the card. Assuming that the software running on the card will not allow unauthorised access to card data (e.g. only allowing data to be written to by off-card software, never read) keys can be securely held on the JavaCard without fear of them becoming known to untrusted parties.

Another benefit of the JavaCard is that in order to download a new applet to the card, or remove an existing applet from the card, a host must have a cryptographically signed copy of the applet. Only applets signed using keys known to the JavaCard OS can be downloaded or removed.

Since several applets can coexist on one JavaCard (though only one can be active at a time) the integrity of the BEAST applet is ensured as only parties with the ability to cryptographically sign the applet may alter it. However the main security consideration is that once secret keys have been uploaded to the BEAST applet on the card, there is no way to recover them. The JavaCard OS ensures that while several Java applets exist on the card, none may access memory used by another applet. The key assumptions here are that both the BEAST applet and the JavaCard OS are bug free.

Work is ongoing within IBM Zurich to mathematically prove the security of the JVM running on the ZRL JavaCard [52].

3.6.5 Overview of OpenCard

The Open Card Framework (OCF) [62] is designed to provide a high level service oriented software interface so that applications wishing to communicate with a smart card need not know about the low level APDU's (or other communication protocols) involved. OpenCard handles all card-host communication and specially written OpenCard *card services* must be written for each specific on-card application. Card services are modular in that each is a Java object, and can be instantiated and removed dynamically by OpenCard when an OpenCard client application needs them.

OpenCard has been chosen for this purpose partly because of the development experience with the OCF that the Zurich laboratory has (much of the OCF has been designed here), and because in contrast with other similar systems it is a true modular and open multiplatform standard. A competing standard PS/SC [40] is tied to the Win32 family of operating systems, and is not written in Java. OpenCard was the better choice, as in addition to directly supporting most of the smart card terminals that PC/SC does, OpenCard is also able to make use of PC/SC itself to support yet more terminal device types.

3.6.6 OpenCard Card Services

For each card service there must also be a corresponding *card service factory.* The factory module recognizes the type of smart card inserted into a terminal (typically by the card's ATR), and only allows its CardService to be instantiated if the correct type of smart card is present. The card service handles all the required card-host communication sessions needed for a particular smart card function. Since card services are implemented as Java methods, the results computed by the card can be decoded from response APDU's into standard Java data types such as integers and strings.

Chapter 4

DABWeb Design

4.1 Design Goals

The reference implementation of DABWeb is designed to make maximum use of existing standards and portable software technologies. Since the implementation of DAB data services requires the use of a large amount of specialized hardware both for transmission and reception it is important that a reference implementation should be dependent on specific devices as little as possible.

In all cases the goal has been to create software interfaces which map the functionality detailed in standards documents - for example MOT and DAB - onto the subset of functionality offered by specific devices. That way new devices designed around the same standards should be easily accommodated by the current software. Since parts of the DABWeb system may be reused separately in future, such as the encryption system, a generic interface design will also help to separate reusable components. The following broad design goals were considered for the DABWeb client and server.

Design issues specific to the client side were :

- The ability to cache partially downloaded carousels so that when the same carousel is repeated later, downloading can continue from as near to the interupted point as possible.

- Management of client-side caching to allow encrypted data to be cached and recognized as such for later decryption. Every cached file should have an expiry date and other associated parameters so that the cache can be weeded.
- Generation of a DABWeb home page by the client which allows a DABWeb user to immediately access any web pages currently held in the cache. This home page should be updated ‘on the fly’ so that users of the DABWeb system will continuously have the most up to date web resources held in the cache available to them.
- Design of a software interface that allows the MOT to be used in combination with several different types of DAB device transparently (virtual DAB data channel). This software interface should be used by both the DABWeb client and server.

Issues considered as part of the design of the server were :

- The ability to split content into small parts. This allows receivers to quickly resynchronize to a lost transmission signal, and continue caching the current carousel.
- Internet downloading capability of certain standard Internet content types, which are to be cached locally until ready for transmission. Content types such as Java applets, graphic and HTML files pose no problems, however script files that form part of some web pages are not suitable for DABWeb broadcast, as they require two way communication with a remote server in order to function. All types of content supported by WebFS are suitable for broadcast however, allowing WebFS to act as a content filter.
- Ability to be able to encrypt data at the server side in real time, and allow the same data to be decrypted in real time by receiving DABWeb clients.

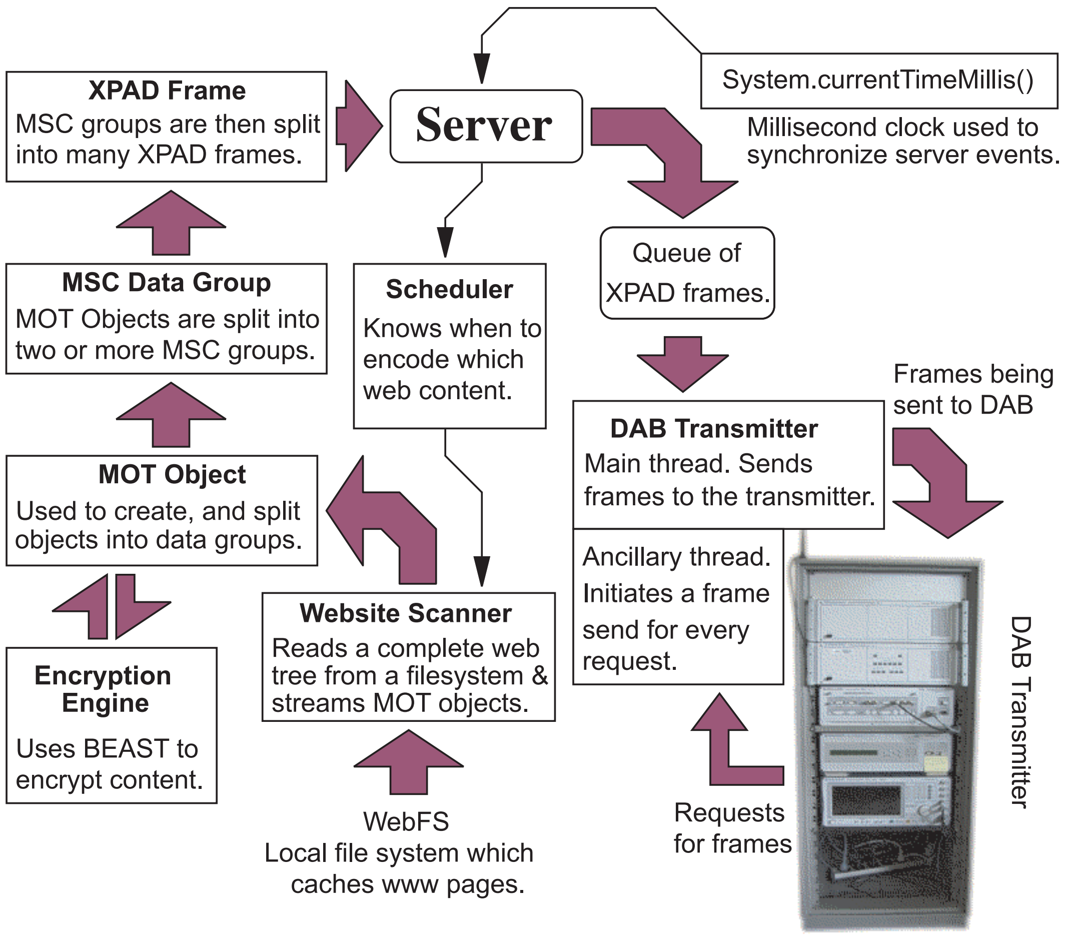

XPAD Frame
MSC groups are then split into many XPAD frames.
Server
System.currentTimeMillis()
Millisecond clock used to synchronize server events.
MSC Data Group
MOT Objects are split into two or more MSC groups.
Scheduler
Knows when to encode which web content.
Queue of XPAD frames.
Frames being sent to DAB
DAB Transmitter
Main thread. Sends frames to the transmitter.
MOT Object
Used to create, and split objects into data groups.
Ancillary thread. Initiates a frame send for every request.
Website Scanner
Reads a complete web tree from a filesystem & streams MOT objects.
Encryption Engine
Uses BEAST to encrypt content.
DAB Transmitter
WebFS
Local file system which caches www pages.
Requests for frames

Figure 4.1: Architectural overview of a DABWeb server system

4.1.1 Overview of the Server system

The DABWeb server system is designed to harvest content from the Internet using WebFS, map individual recovered web sites into carousels (each carousel being a group of MOT objects) and then transmit each carousel according to a schedule. Figure 4.1 shows all the functional units involved in this process, starting with web sites cached by WebFS being mapped into carousels by the web site scanner module (where they can optionally be encrypted also).

Carousels are cached in main memory by the server, until the scheduler requires that they be broadcast. When this occurs, each MOT object is split into MSC data groups, then further split in the case of PAD mode transmission into XPAD frames. This stream of data is then sent to a transmitter, via an appropriate device driver software layer. Between the device driver and the lower levels of the protocol stack lies a queue which allows the server to reliably deliver one frame to the transmitter as quickly as possible. Figure 4.1 illustrates this for the case of PAD transmission.

Laboratory experiments have shown that with certain receiver/transmitter device combinations, data frames arriving a few milliseconds later than the transmitter expected will not be transmitted. A receiver device may then output a repeat copy of the last frame received to a DABWeb client. This situation is avoided by most receiver devices, which will not output a data frame when none was transmitted. For those receivers that will output a data frame whether one was transmitted or not, a sequence number system must be used to label consecutive frames.

At the lowest level, integration with WebFS is provided using standard file system operations. To ensure that a web site is cached, the server must have previously have retrieved the directory content of that web site from WebFS. The scheduler module is responsible for this. Assuming that a web site is entirely cached, the web site scanner module can then map the site into a carousel of MOT objects.

Since carousels of scanned web sites are held in memory, the same MOT objects can be used each time a carousel is repeated. As information about the time period until the next carousel repeat is built into the data stream, cached carousels must be altered before retransmission.

4.1.2 Overview of the Client system

The client architecture (see fig. 4.2) illustrates the processes required to receive and decode an incoming DABWeb data stream. At the lowest level, a driver device for a DAB receiver device manages the receiver (tuning functions etc.) and feeds the frame decoder with a stream of incoming bytes. Individual frames are recovered (the architecture illustrates PAD mode) and a stream of XPAD frames are then grouped together into a stream of `MSCDataGroup` objects by the group decoder.

If there are errors, or missing parts at any stage up to the level of `MSCDataGroup`, the client simply discards the current group of frames, and continues decoding the next group. Once a complete `MSCDataGroup` is received it is cached in memory, and will not be dealt with further until all the data groups needed to complete that MOT object are cached. The MOT Decoder attempts to assemble several MOT objects concurrently, allowing the parts of each object to be collected over several repeat transmissions if needed.

The MOT decoder outputs a stream of complete MOT objects which can then be decrypted, and sent on to the client's persistent cache. The cache manager generates a list of currently cached carousels as a HTML *home page*. This home page is updated as new carousels are cached, giving DABWeb users a entry point into the subset of the web they have cached.

A user profile system is also illustrated as part of the client architecture, allowing users to specify which content they would like to decrypt, and cache. This would give users the freedom to only pay for what they are likely to read.

The client's persistent cache may also support functions such as weeding, which would ensure out of date and read carousels are removed from the cache to make space for

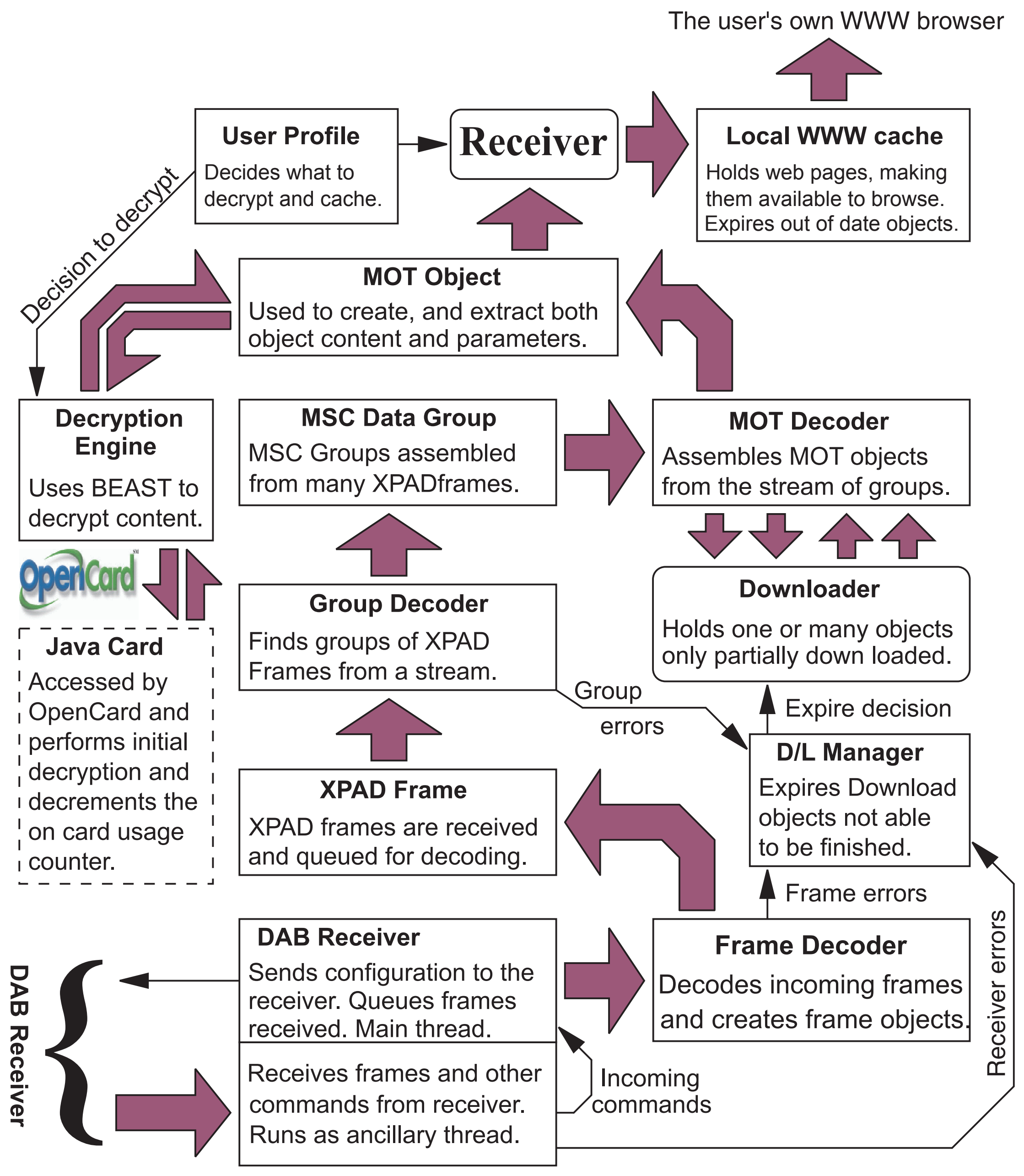

The user's own WWW browser
User Profile
Decides what to decrypt and cache.
Receiver
Local WWW cache
Holds web pages, making them available to browse. Expires out of date objects.
Decision to decrypt
MOT Object
Used to create, and extract both object content and parameters.
Decryption Engine
Uses BEAST to decrypt content.
MSC Data Group
MSC Groups assembled from many XPADframes.
MOT Decoder
Assembles MOT objects from the stream of groups.
OpenCard
Java Card
Accessed by OpenCard and performs initial decryption and decrements the on card usage counter.
Group Decoder
Finds groups of XPAD Frames from a stream.
Downloader
Holds one or many objects only partially down loaded.
Group errors
Expire decision
D/L Manager
Expires Download objects not able to be finished.
XPAD Frame
XPAD frames are received and queued for decoding.
Frame errors
DAB Receiver
DAB Receiver
Sends configuration to the receiver. Queues frames received. Main thread.
Frame Decoder
Decodes incoming frames and creates frame objects.
Receiver errors
Receives frames and other commands from receiver. Runs as ancillary thread.
Incoming commands

Figure 4.2: Architectural overview of a DABWeb client system

new carousels. Parameters encoded in each MOT object can be used to specify the useful life of a carousel.

4.2 Mapping a web tree structure onto a stream of MOT objects

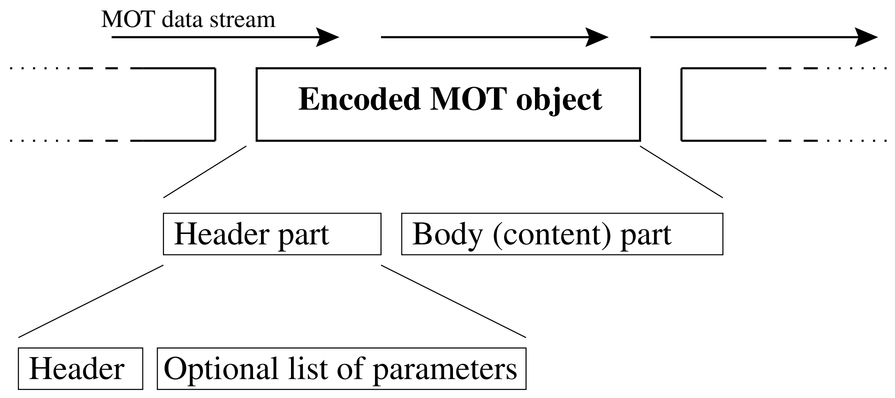


Figure 4.3: Encoding the components that make up an MOT object

An MOT object consists of two data areas (fig 4.3), the first part contains header information and parameters associated with that object, and the second the object's content. In terms of a web site, each content area can contain one or more of the files comprising a web site. Any number of application specific parameters may be included in the header part which can be a maximum of 8192 bytes long. A flexible header parameter system allows each parameter to vary in size from one byte to 32,768 bytes long - though in practice a parameter must be a maximum of 8192 bytes long in order to fit into the header area.

All parameters contain a six bit type indicator, and ETSI specify many useful parameter encoding schemes including time of day, and textual information given in a specific international character set. For DABWeb new parameter types have been introduced to indicate that the content part is encrypted and to map each of the files in the content part back to a hierarchical web site tree data structure.

A maximum of 2^{32} different groups of MOT objects can be distinguished between numerically. Each group is considered to be one web site by DABWeb. For this reason the DABWeb server must ensure that each web site broadcast has its own unique group identifier, and rollover of the group identifier must be considered. It is possible that an encryption system could be implemented where a smart card would know which group identifiers it is able to decrypt (and maybe which keys to use).

The MOT parameter types `ContentName` (type 12) and `ContentDescription` (type 15) are used by the DABWeb system to map hierarchical web trees retrieved from the Internet into a serialized stream of MOT objects. Access to files over the Internet via hypertext transfer protocol (HTTP) requires both the file name and address to be specified using a uniform resource locator (URL). The URL uniquely identifies a file's position anywhere on the Internet. In DABWeb the `ContentName` parameter is used to contain the filename part of the URL (e.g. `/dfm/home.html`), with the `ContentDescription` parameter containing the Internet address of that file (eg. `www.st-andrews.ac.uk`).

A new parameter (type 16) is used to indicate the length of each individual file comprising the content part. With more than one file stored, the parameters associated with each must appear in the parameter area in the same order that the files appear in the content area. It is assumed that each subsequent file is from the same Internet URL as the previous (the same web site), unless a new URL parameter is supplied along with that file's name and length parameter. With these three parameters a DABWeb receiver can cache complete web trees given an input stream of MOT objects.

If DABWeb was to be used as a high bandwidth system with many thousands of web sites available for download, more care might be needed to ensure that different web trees were not broadcast with the same group number so as not to confuse receiver client cache managers. After an MOT object used to transport all or part of a web tree has been decoded, the DABWeb receiver client makes use of each file's Internet address in order to cache the files the object contains. The client cache creates a single directory folder for each web tree, and names that folder the URL of the web tree. Files received belonging to a web tree which already has a folder in the cache are stored in that folder. Within cache folders sub-folders are created when needed, so that the full hierarchical structure of a web site is cached.

The size of the content part of an object is a maximum of 2^{28} bytes long. In practice this should not provide a limitation for DABWeb, and an optional *unknown size* flag is provided if streamed content was ever used. An analysis (later in this chapter) of the BEAST encryption system links the maximum size of an MOT object to the maximum data rate theoretically possible using a smart card based decryption system.

4.3 Creation of a Virtual DAB data channel device driver

The creation of the virtual broadcast data channel software layer allows clean separation between higher layers in the MOT protocol stack, and lower layers which depend on the type of data transport (XPAD or Packet mode) being used, and device specifics. The goal has been to allow the creation of a single Java object representing a single broadcast data channel - either a full subchannel or an XPAD stream.

Each data channel object would provide the same software interface whichever transport method was selected when the object was created. The only device specificity to be considered is that data channels connected to DAB receiver equipment will be read only and transmitter devices will only allow a write only channel.

With this software layer in place, along with a standard framework into which drivers for specific DAB devices can be written, access to information services provided by DAB networks should be as easy as streaming data to or from a local file.

4.4 Using JavaCard to decrypt a multimedia data stream

In an ideal secure system the untrusted host computer would pipe an entire multimedia data stream to a smart card, and have the card return the decrypted stream to the host. The card would charge for each item of content decrypted, for example by decrementing a counter, and the untrusted host would never be able to know the secret keys held on the card, and so circumvent the charging mechanism.

However the round trip time (table 4.1) for sending encrypted blocks of data to a smart card and the card then returning the resulting decrypted block is many times higher than if the same blocks were decrypted entirely by the host computer.

```
Rtt = Time to send data to smart card including command +
      Time taken for smart card to decrypt data +
      Time taken to return decrypted result
```

Table 4.1: The Round Trip Time (Rtt) from host to smart card

This is due to the card's computing speed being relatively slow (compared with the host computer), and the card-host communication channel adding significantly to the time delay. In addition there is an upper limit to the amount of data the smart card can decrypt at once due to limited memory resources on the card. It is the case that today's JavaCard technology accessed through OpenCard is not able to decrypt a DABWeb data stream in real time (in the order of about 2Kb/second for a single PAD channel).

The BEAST RK algorithm takes a different approach, using the smart card device to decrypt only the first 160 bits from each block of content. The first 160 bits along with the block's 160 bit session key are sent to the smart card. The card then decrypts these using the secret keys and returns only the resulting 160 bits of decrypted content. The use of only the first 160 bits in this way is specified by Lucks [48] who holds that this allows BEAST RK to be secure as discussed in the Component Technologies chapter.

The remainder of the n-byte block (n-20 bytes) is then decrypted using a stream cipher (generated using the SEAL stream cipher algorithm) on the untrusted host. Since the host can generate the stream cipher very quickly the remainder of the decryption operation takes very little extra time. If round trip time for the card to decrypt 20 bytes combined with the time for the host to decrypt n-20 bytes is less than or equal to the time taken to receive n bytes (at data rate n bytes/second) then real time decryption is possible.

Normal network buffering can be used to smooth over variance in arrival rate of continuous data streams, so in fact only the average data rate would have to satisfy the condition. Experimentation with different sized client-side buffers would be needed, also taking into account that many MOT objects are repeat broadcasts, and so do not need to be decrypted a second time.

4.5 Creation of OpenCard services for JavaCard

OpenCard provides built in *services* for many common smart card uses such as card holder verification and services allowing a smart card to be used as a low capacity file store. Since BEAST is a custom application, a new set of card services has been written. Card services also require associated *card service factories* which are linked into the OpenCard framework, and allow the framework to recognize a specific type of smart card, and create the appropriate type of card service.

The OpenCard service `BEASTCardService` to be written should operate as follows in conjunction with a client requiring BEAST decryption services. On startup, a client application registers itself with OpenCard. The OpenCard software initializes all connected card terminals (if it has not already done so for another application) and then goes back to sleep. Card terminals are usually registered to OpenCard by means of a `.properties` file which is standard practice for Java extensions and will be parsed by OpenCard on startup. The `.properties` file also specifies the location of `CardService` and `CardServiceFactory` classes.

After a client has registered itself with OpenCard it can choose how it can interact further using three different software models :

- Java events are generated whenever a new smart card is inserted or removed, and an OpenCard client program can capture these events. The event driven model does not require client applications to block while awaiting a smart card,
- The client can poll OpenCard, continuously waiting for a smart card to become available,
- The client can call a blocking function in OpenCard, `WaitForCard` which does not return until a smart card becomes available.

When a smart card is inserted into an OpenCard card terminal the client can decide whether it is interested in that card, and if so ask OpenCard to instantiate appropriate card service. For DABWeb, this is a collection of services as written in the class `BEASTCardService`.

In the case of `BEASTCardService`, a `BEASTCardServiceFactory` must first recognize that a BEAST JavaCard has been inserted into an OpenCard card terminal. It then creates an instance of a `BEASTCardService` class which it passes to the client application. Client applications use a set of high level OpenCard calls to learn whether a smart card is in the card slot, and then ask to be given the card service class they would like to use - in this case the `BEASTCardService`.

If the JavaCard in the card terminal slot identifies itself as a BEAST JavaCard then OpenCard hands the application an `BEASTCardService` object, which the DABWeb client can then use to perform decryption and check to see how many credits are left on the card.

Once the client application has been passed the `BEASTCardService` object, three high level functions are available to it as long as the JavaCard remains in the card terminal:

1. Upload keys and credits to the smart card applet. This is done only at the server-side and the cards are then sold to DABWeb users,

2. Decrypt a content byte array with a given session key. This operation is done each time a client decrypts received content,
3. Read the remaining number of credits on the smart card.

Making the upload keys function available to untrusted hosts does not compromise the cryptographic security of the BEAST RK algorithm. Since only valid keys will be of any use in decryption, as long as only trusted parties know the secret keys, it does not matter that others may try to upload incorrect keys to the card.

When a charging system requires that a credit value be associated with a set of secret keys so the keys can only be used a set number of times, then that value should be uploaded only after the keys have been placed in non volatile memory in the card. The `BEASTCardService` function to upload keys, first uploads all the keys, and then the credit value associated with them only after the keys have been uploaded. This ensures that new credits cannot be uploaded for old keys by powering down a card prematurely - something that might be possible should the credits be uploaded first.

The OpenCard `CardChannel` software layer is a virtual channel through which a terminal can communicate with a smart card. For example when the `BEASTCardService` would like to talk to a JavaCard a `CardChannel` may be requested for the card terminal that the JavaCard is currently connected to. APDU's may be sent and received through a `CardChannel`, and the card service object concerned need not worry about how many card terminals are connected, or what type of device they are.

When OpenCard performs an applet select, the previously selected applet on the JavaCard loses any transient (volatile) state that it currently holds. Since several client OpenCard applications may be assuming that they have access to *their* card applet, OpenCard should attempt to restore the state of an application's JavaCard applet, when the original applet is re-selected. Alternatively the OpenCard service can request *mutexed* access to the JavaCard, and if successful, OpenCard will only allow that card service to use the JavaCard. A card service should either be able to restore context on exit or else request mutexed access to the card.

Since an applet select need not be performed if the correct applet is currently selected, OpenCard services which cooperate and remember which applet is selected can avoid the overhead of performing an applet select before each JavaCard operation.

4.6 Functional units of the BEAST RK algorithm

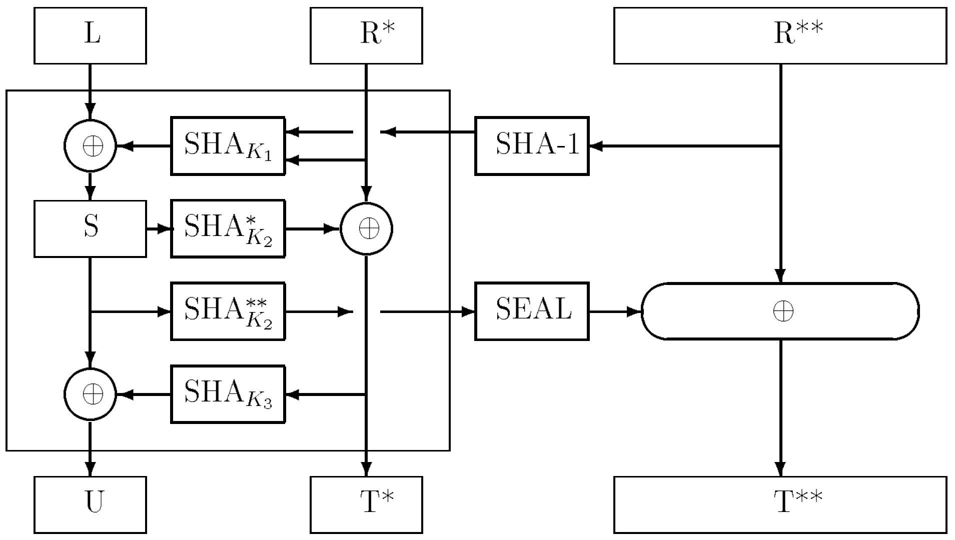


Figure 4 The remote-key variant BEAST-RK of BEAST.

Figure 4.4: Lucks's original diagram [48] of the BEAST RK encryption system

Figure 4.4 indicates each functional block involved in the BEAST RK encryption process. Decryption is performed by applying the inverse of these steps to the encrypted data. The three functional components of the BEAST RK algorithm are :

1. The SHA-1 secure hashing function which can take an arbitrary sized byte array, and generate a 20 byte hash value unique to that byte array.
2. The SEAL stream cipher function which can expand a given 160-bit hash sequence into an n-byte stream cipher where n is the required number of bytes. The process generates the same n-byte sequence for the same 160 bits every time.

The process cannot be reversed, and so allows cryptographically secure ciphers to be generated.

3. A BEAST RK specific variant of the SHA-1 algorithm SHA$_{K_X}$ which uses the SHA function to generate a hash of one of the BEAST RK secret keys - K_X - exor'ed with a bit string of the same length.

The same three secret keys used by BEAST RK for the encryption of content must again be used to decrypt that content. Three keys are used by BEAST RK to provide stronger encryption than is achievable with two keys. See Lucks [48] for more details. Key 1 is 320 bits in length and keys 2 and 3 are both 160 bits in length. Each of the three keys is randomly generated, and a keyset of all three keys is known to both the users smart card (for decryption operations) and the DABWeb server (for encryption operations).

Content is encrypted in two parts along with a 160 bit random value to be used in generating a unique session key, and cryptographic cipher for the content. The first 160 bits of content are labeled R*, with the remainder being labeled R**. Because the content is encrypted on a trusted host before it is broadcast you do not need a smart card for encryption. A session key L (size 160 bits) is also introduced into the process, which must be generated using a secure random function (one is supplied as part of the Java SDK).

From figure 4.4 the inverse of the encryption process can be determined, and so the formula for the SEAL cipher that the JavaCard must compute can be derived. This acts on the outputs from the encryption process, T*, U and T**, and is shown in table 4.2.

The JavaCard is passed the U and T* values, and returns the result of the above function. From the JavaCard's output, the SEAL *expanding hash* (stream cipher) function generates a cipher of the same length as the byte array T**. The SEAL function executes on the host, after using OpenCard to retrieve the JavaCard part of the decryption operation. After the SEAL cipher has been generated, it is exor'ed

```
shakey( 2, exor( shakey( 3, T* ), U ) )
```

With the following defined functions :

`shakey( key number, 160 bit value input )` yields the BEAST RK specific SHA_{K_X} function using secret keys 1, 2 or 3.

`exor( array1, array2 )` returns the value of the two byte arrays bitwise exor'ed together.

Table 4.2: The on smart card part of the BEAST RK decryption process

with the T** array, and then the decrypted T* and T** are concatenated together to yield the decrypted plain text.

The original session key, L is never recovered, however continuing the inverse of the encryption process the JavaCard could find this value. It is of no direct use if it is computed randomly. Implementation of BEAST is helped by Java's built in SHA Secure Hash function. This function is also present in the JavaCard's on board *vault* of cryptographic functions.

4.6.1 BEAST encryption of MOT objects

BEAST RK encryption is used to encrypt an MOT data stream with each block encrypted being the content part of one MOT object. An extra parameter known as `BEAST_U` is present in the header of each encrypted MOT object containing a session key (the parameter's presence is the indicator that the block is encrypted). Since an MOT object is a container for one or more physical files making up a web site, extraction of individual files is only possible after decryption of the whole block has taken place.

After the entire content block is decrypted individual files can be extracted and cached by a DABWeb client. Multi-file MOT objects are important to BEAST because the data rate of the incoming stream imposes a minimum size restriction on each block decrypted in order to achieve real time decryption.

4.6.2 Charging strategies for decryption using the JavaCard

Since the `BEAST_U` session key was generated in conjunction with the entire content of each file using a contracting hash function (SHA), use of a non-random value *might not* compromise the cryptographic security of the BEAST algorithm to an unacceptable standard. For example selecting four ranges within which a random value could be chosen, and using each range as an indication of charge value or that a new charging *time period* had begun.

If the use of the session key was not possible, then the first few bytes of any BEAST encrypted content could be used for the same purpose with no danger of loss of cryptographic security. Once the card had decrypted the appropriate bytes of content it could then compare them to the same bytes recovered from the last decryption operation. If they were different then the card could assume that a new time period was in effect and that the user should have a credit debited. Optimally this charging information would be held in the first 20 bytes of content, so that they would be known to the JavaCard BEAST applet after decryption. In this way the DABWeb system could charge for content by the day, and the same content being received twice on one day would not cause the user to be charged more.

A more advanced variant of this system would be needed if some content was likely to be the same over several days as it would seem unfair to charge users twice for the same thing (unless of course the DABWeb service provider was charging on the basis of service availability). The modification would require the smart card to keep track of the values associated with the last four or five decryption periods. Only when the first few bytes did not match any of the last few sets decrypted would a DABWeb user's smart card be charged one credit.

Such a system would require more intelligence on the part of the server, but would allow fair charging for services that were updated daily, and those that were updated less frequently e.g. weekly. The nature of DABWeb would make its use for services that are updated less frequently than this limited.

Chapter 5

DABWeb Implementation

Java was chosen for most of the software implementation, allowing a modular object oriented approach. The Java Software Developers Kit (SDK) with an associated extension package `java.x.comm` allows communication with peripheral devices, for example by RS232 interface, however some device interface types are not supported. For this reason the DABWeb system has integrated low level C code using the Java Native Interface (JNI).

The use of Java has allowed most of the system to be portable to any architecture which has a Java Runtime Environment (JRE). Implementation work is divided between the DABWeb client and server programs, and then further subdivided into the following components :

- The web server system which schedules web content carousels appropriately and manages encryption and key generation when required,
- A network session layer that complies with and extends the MOT standard for DABWeb specifics. This has also been developed entirely in Java and provides a uniform set of multimedia object encoding and decoding functions for use with encrypted web content,

- A network transport layer for DAB based network devices, including drivers for a selection of development DAB receiver devices, and one transmitter device used to test the system. This has been written in C, making use of Java Native Interface technology to communicate with higher layers of the protocol stack,
- A smart card based decryption *applet* program that performs the on card part of the BEAST decryption process,
- A selection of OpenCard card services that allow access to the BEAST card applet, for uploading of secret keys and credits (server-side), and decryption operations (client-side). Also a method of charging for each decryption operation that will inform the user how many credits are left on a BEAST card,
- The client-side cache manager and GUI which allows web pages to be received and stored in local storage, removed when they become out of date, and decrypted on demand,
- The `java.x.comm` extension has been used to create a DAB network simulator allowing the client-side software to connect directly to the server, over an RS232 serial data link in the laboratory.

As much of the code as possible has been made common to both the client and server.

5.1 Client Implementation details

Figure 5.1 shows an overview of the implemented client system. Within the client-side MOT protocol stack no attempt is made to cache incomplete MSC data groups, and though it would be possible to do so if only a few frames from the group were damaged, it is unclear whether this is worthwhile. Longer trials of DABWeb in different reception conditions should resolve this and help decide whether any extra effort should be made by a receiver client to keep damaged data groups in a cache

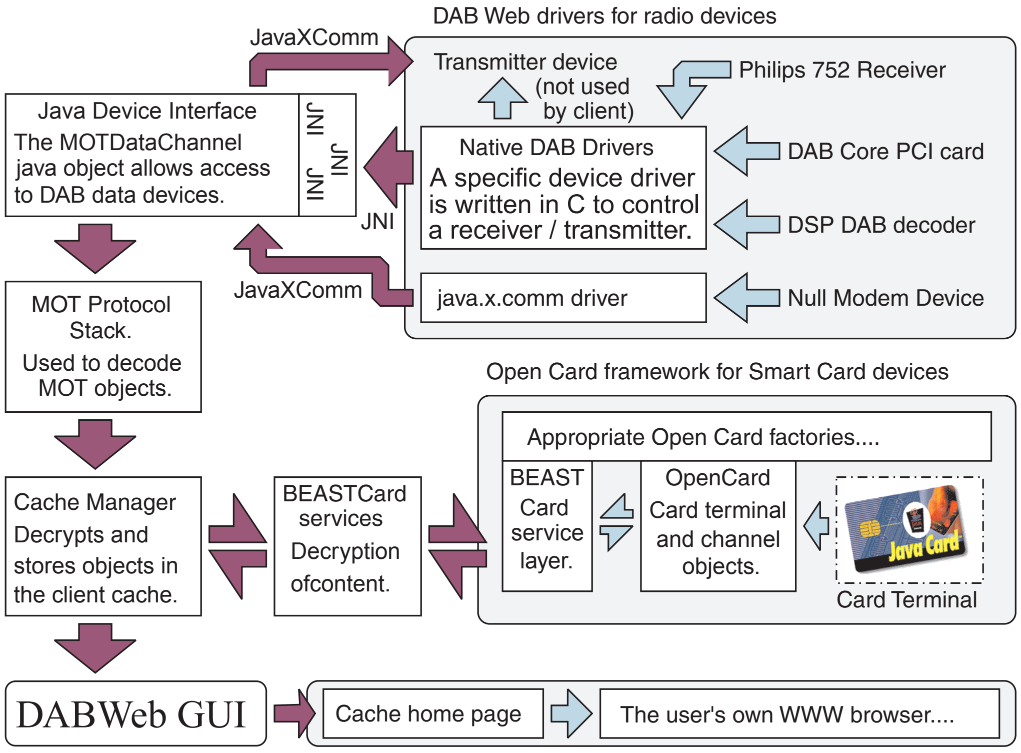
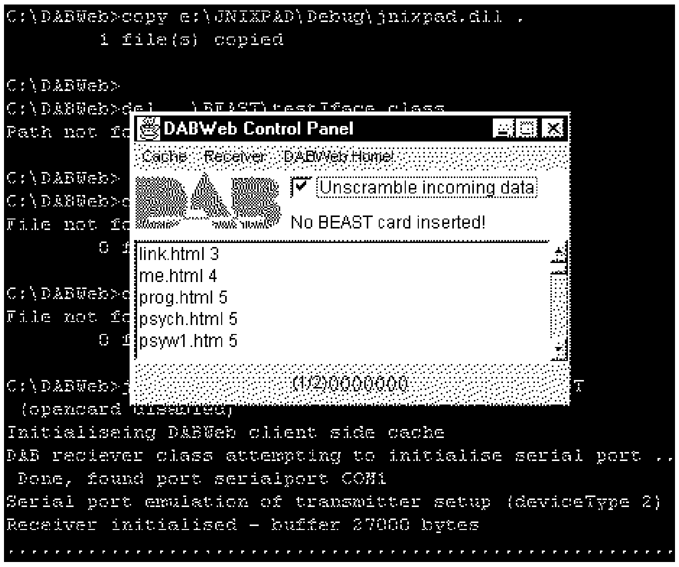

DAB Web drivers for radio devices
JavaXComm
Transmitter device
(not used by client)
Philips 752 Receiver
Java Device Interface
The MOTDataChannel java object allows access to DAB data devices.
JNI
JNI
JNI
JNI
Native DAB Drivers
A specific device driver is written in C to control a receiver / transmitter.
DAB Core PCI card
DSP DAB decoder
JavaXComm
java.x.comm driver
Null Modem Device
MOT Protocol Stack.
Used to decode MOT objects.
Open Card framework for Smart Card devices
Appropriate Open Card factories....
BEAST Card service layer.
OpenCard Card terminal and channel objects.
Java Card
Card Terminal
Cache Manager
Decrypts and stores objects in the client cache.
BEASTCard services
Decryption ofcontent.
DABWeb GUI
Cache home page
The user's own WWW browser....

Figure 5.1: Overview of a DABWeb client, showing data flow between the different components


```
C:\DABWeb>copy e:\JNIXPAD\Debug\jnixpad.dll .
        1 file(s) copied

C:\DABWeb>
C:\DABWeb>del    \BEAST\testIface.class
Path not fo

C:\DABWeb>
C:\DABWeb>c
File not fo
        0 f

C:\DABWeb>c
File not fo
        0 f

C:\DABWeb>j                                          T
 (opencard
Initialiseing DABWeb client side cache
DAB reciever class attempting to initialise serial port ..
 Done, found port serialport COM1
Serial port emulation of transmitter setup (deviceType 2)
Receiver initialised - buffer 27000 bytes
..........................................................
```

DABWeb Control Panel

[✓] Unscramble incoming data

No BEAST card inserted!

```
link.html 3
me.html 4
prog.html 5
psych.html 5
psyw1.htm 5
```

(1/2)0000000

Figure 5.2: The client side DABWeb GUI

until that specific data group is repeated. Incomplete MOT objects are carefully cached though, and completed when they are repeat broadcast.

A Graphical User Interface (GUI) has been created using the Java Abstract Window Toolkit (AWT) software, to allow users to follow the progress of DABWeb in downloading web trees. The GUI (fig. 5.2) lists each complete file downloaded and decrypted, and also allows the user's chosen web browser to be launched viewing the DABWeb home page. In the future such a GUI might be expanded to allow configuration of a receiver device, including the audio output functions of the receiver.

5.2 Server Implementation details

The DABWeb server has been implemented to integrate with WebFS, and closely resembles the server architectural overview in the design chapter. No GUI has been implemented, as console based I/O allowed adequate progress monitoring for the initial prototype. For this reference implementation BEAST secret keys are built into the Java source of the server. These may be uploaded to a decryption smart card using a separate command line program which includes the same secret key source file, and interacts with the appropriate BEAST OpenCard services.

5.3 Object Serialization for Network Protocols

One key concept when implementing a communications protocol stack in Java is the idea of *object serialization*. Java provides an internal mechanism by which objects held in memory may be sent through a communications channel - either between processes on the same host, or to a remote computer which also has a JRE.

The Java serialization mechanisms are not used for DABWeb because once serialized the byte arrays created for Java objects bear no resemblance to the MOT standard, and the use of serialized Java objects at any level in the MOT protocol stack would require that all clients and servers be tied to using a JVM. It is important that the MOT standard be followed because it has been optimized for broadcast multimedia data over DAB. While it may be possible to use Java's own serialization mechanisms they would be likely to be less efficient, and make no provision for repeats in the data stream and caching of partially received MOT objects.

Instead DABWeb object serialization makes use of specific methods (see fig. 5.3) in the classes representing different APDUs of the MOT protocol stack. For example the `MSCDataGroup` class contains two constructor methods one of which accepts a serialized byte array of an `MSCDataGroup` and the other constructs the object from a group of Java parameters specifying the composition of an `MSCDataGroup`. Once constructed, *encode methods* allow the object to be serialized, and passed down the protocol stack, for example by splitting the `MSCDataGroup` into one or more `XPADFrame` APDU objects.

Alternatively an APDU object can be decoded by methods specific to each APDU class, which allow any part of an APDU to be read. These decoded parts can be used to instantiate APDU objects of the next level up in the protocol stack, for example an `MOTObject` APDU can be created by decoding the content parts of one or more `MSCDataGroup` APDUs.

In this way individual APDU's are implemented as *self serializing* Java objects. Since each serializes according to the byte encoding specified in the MOT standard, there is no mandatory requirement for a JVM to be able to deserialize that object on a remote machine.

Since each APDU used by the DABWeb server (`MOTObject`, `MSCDataGroup` and `XPADFrame`) has been written in this way, the implemented server system very closely follows the server architecture illustrated in the design chapter. Also the same APDU classes can be used for both the server and client programs, since the APDU classes are written to include both encode and decode functions. This type of code re-use should help the DABWeb implementation be more free from bugs, as only one class

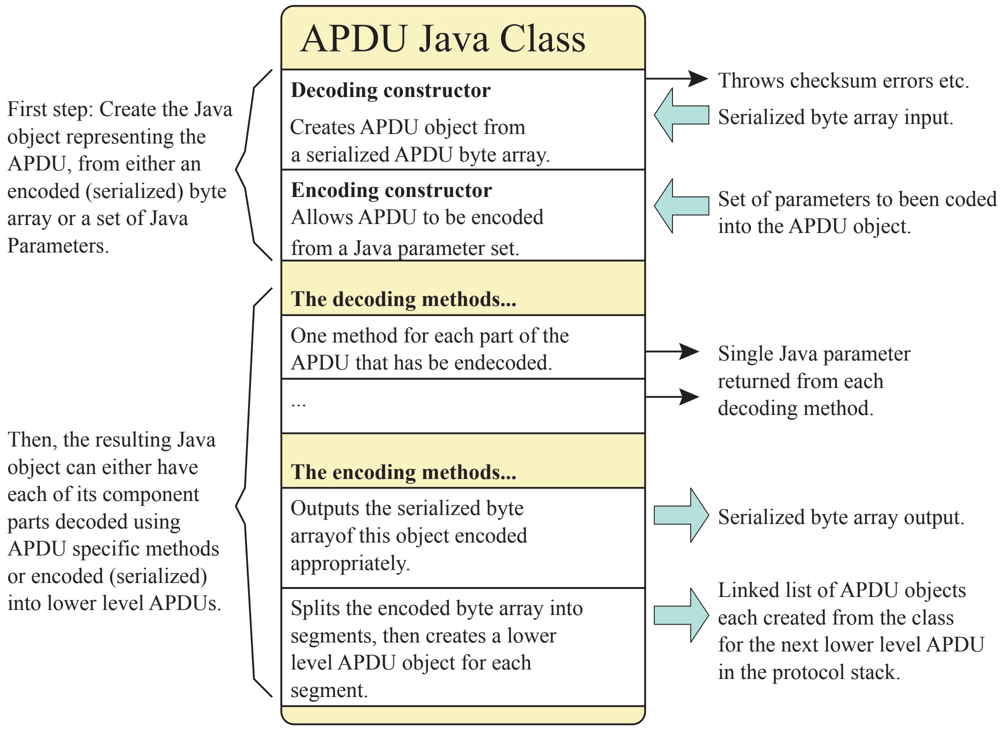

APDU Java Class
Decoding constructor
Creates APDU object from a serialized APDU byte array.
Encoding constructor
Allows APDU to be encoded from a Java parameter set.
The decoding methods...
One method for each part of the APDU that has be endecoded.
...
The encoding methods...
Outputs the serialized byte arrayof this object encoded appropriately.
Splits the encoded byte array into segments, then creates a lower level APDU object for each segment.
First step: Create the Java object representing the APDU, from either an encoded (serialized) byte array or a set of Java Parameters.
Then, the resulting Java object can either have each of its component parts decoded using APDU specific methods or encoded (serialized) into lower level APDUs.
Throws checksum errors etc.
Serialized byte array input.
Set of parameters to been coded into the APDU object.
Single Java parameter returned from each decoding method.
Serialized byte array output.
Linked list of APDU objects each created from the class for the next lower level APDU in the protocol stack.

Figure 5.3: Structure details of a *self-serializing* APDU class written in Java

need be tested for each APDU.

The DABWeb client requires extra decoding classes between the APDU classes. The decoding classes collect together lower layer APDU units, and then when the complete serialization of a higher layer APDU unit has been received, the appropriate APDU object is instantiated. If at construct time the checksum of the higher APDU fails, a Java exception will be thrown, and that APDU can be discarded.

At the `MSCDataGroup` APDU layer in the protocol stack, a client-side decoder has been written which is able to cache several `MSCDataGroup` objects until a complete `MOTObject` is cached. This allows an `MOTObject` to be received in parts over several repeat broadcasts. The decoder has an appropriate caching strategy to ensure that it will not keep `MSCDataGroup` objects that will not be used in the future to assemble complete MOT objects, for example because that MOT object is unlikely to be broadcast again.

5.3.1 Java based Serial Communications for PAD services

RS232 communication in Java is provided for by the `java.x.comm` package. This is an extension to the core Java SDK, and is available from Sun in a stable release version for Windows and Solaris. There is a version of `java.x.comm` for Linux [28], but unfortunately this is less stable, and the ready to build version of its source tree requires debugging traces to be manually disabled at build time.

In order to receive serial data with Java, the communications handler must be written as a separate thread to the main program. Threading semantics were found to differ between Win32 and Linux versions of the JDK, adding to the design considerations of the client and server which are both intended to be portable between Win32 and Linux JVMs.

PAD services may be input to the transmitter system through a `java.x.comm` RS232 link into one of the encoder units. Data must be appropriately framed (according to a protocol supplied by the transmitter manufacturer [45]) for the encoder unit's current configuration.

5.3.2 Transmission of MOT Streams in Packet Mode

In order to transmit MOT data streams in Packet Mode (over a devoted subchannel) a different data interface must be used. As the MSC groups generated by the MOT are the same as those actually sent in DAB transmission frames, two approaches are possible to transport MOT streams from the DABWeb server to the Transmitter system :

1. `MSCDataGroup` APDUs could be sent in packet form directly to a source encoder via one of the higher speed interfaces built into the encoder unit - either RS449 or RS485. Communication with a source encoder would be via a proprietary system that would have to be found from the manufacturer of the transmitter system [45]. A native driver could then be written for DABWeb.

2. `MSCDataGroup` APDUs could be encoded into the format specified by the Service Transport Interface (STI) [53]. The STI is fully specified by ETSI, and describes all aspects of the physical connection and protocol used to send data from a source encoder to a DAB multiplexer unit. This approach would bypass the source encoders, and allow the use of a standard communication protocol. To do this though, hardware (and an appropriate software interface) would have to be developed to output data from the DABWeb server in STI format.

Once received, MSC groups are made available to a DABWeb client via the Radio Data Interface receiver output. This interface is specified by the Eureka RDI taskforce [16], and provides a well defined way to access a complete DAB ensemble as it is output electrically or optically from a DAB radio receiver. In its current state DABWeb cannot use Packet Mode transmision, as the necessary hardware to implement this was not available at development time.

5.4 The MOTDataChannel DAB Device Drivers

This software layer has been written to provide access to any kind of DAB hardware device in a much higher level way than by direct interaction with `java.x.comm`. It provides a virtual communications channel through which serialized `MSCDataGroup` objects may pass, and also implements an architecture allowing low level natively written software drivers for specific devices to be plugged in.

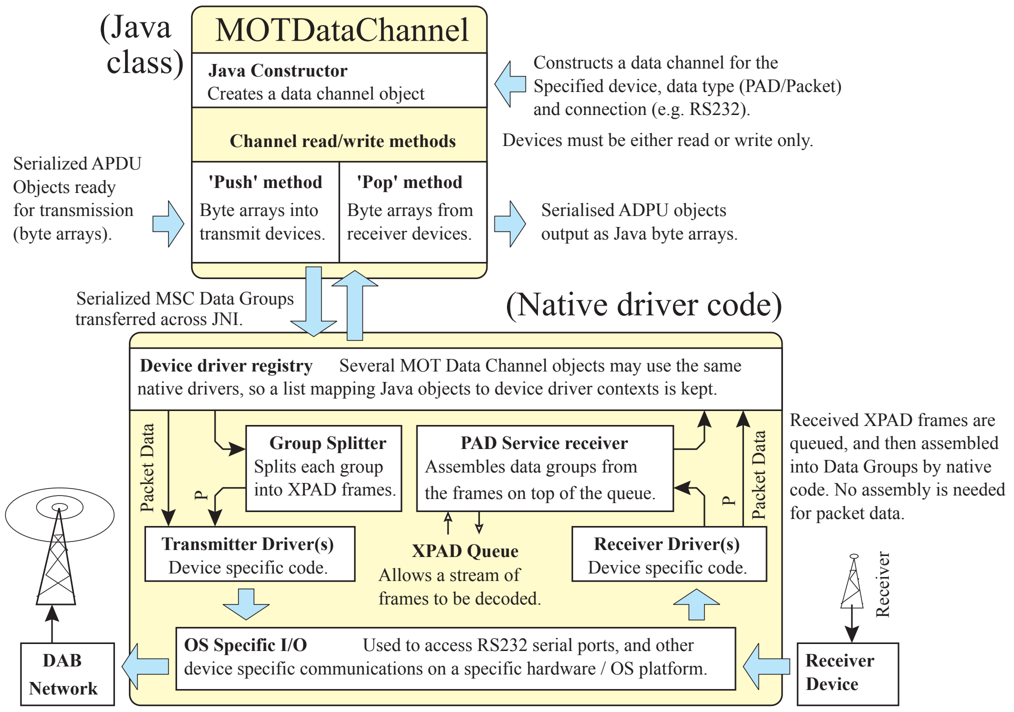

(Java class)
MOTDataChannel
Java Constructor
Creates a data channel object
Constructs a data channel for the Specified device, data type (PAD/Packet) and connection (e.g. RS232).
Channel read/write methods
Devices must be either read or write only.
Serialized APDU Objects ready for transmission (byte arrays).
'Push' method
Byte arrays into transmit devices.
'Pop' method
Byte arrays from receiver devices.
Serialised ADPU objects output as Java byte arrays.
Serialized MSC Data Groups transferred across JNI.
(Native driver code)
Device driver registry Several MOT Data Channel objects may use the same native drivers, so a list mapping Java objects to device driver contexts is kept.
Packet Data
Group Splitter
Splits each group into XPAD frames.
PAD Service receiver
Assembles data groups from the frames on top of the queue.
Packet Data
Received XPAD frames are queued, and then assembled into Data Groups by native code. No assembly is needed for packet data.
P
P
Transmitter Driver(s)
Device specific code.
XPAD Queue
Allows a stream of frames to be decoded.
Receiver Driver(s)
Device specific code.
Receiver
OS Specific I/O Used to access RS232 serial ports, and other device specific communications on a specific hardware / OS platform.
DAB Network
Receiver Device

Figure 5.4: Illustration of `MOTDataChannel` device driver architecture

At the highest level the `MOTDataChannel` Java class (see fig. 5.4) is presented which when instantiated will bind to a connected DAB receiver or transmitter device. Serialized `MSCDataGroup` objects (or other byte arrays) may be pushed into the channel in the case of transmitter devices, and complete `MSCDataGroup` APDUs are popped out of a receiver device channel as byte arrays. Below this lies a natively written driver library, which is modular, and may be compiled on both Win32 and UNIX platforms by linking platform specific I/O and device driver code.

The native library manages all buffering and further splitting/assembling of serialized `MSCDataGroup` objects into/from groups of XPAD frames. From Java an `MOTDataChannel` object is instantiated (see table 5.1) with a given receiver (read only) or transmitter (write only) type, which type of transport is to be used (PAD/Packet mode boolean indicator) and details of how the device is connected to the host computer.

```
MOTDataChannel myChannel = new MOTDataChannel(
MOTDataChannel.philips752, "COM1", true ) ;
```

Table 5.1: Opening a new `MOTDataChannel` for a receiver device

The level of `MSCDataGroup` was chosen for the `MOTDataChannel`, as this is the first layer in the MOT protocol stack that is common to both packet mode data services and PAD based data services. The native library code is built as a dynamically linkable library, which the Java `MOTDataChannel` class binds to at class load time. The `MOTDataChannel` class can then be instantiated many times, and each instantiation registers itself with the same native code library at construction time. The native library keeps track of all current receiver/transmitter devices being accessed by means of a registry and allows more than one `MOTDataChannel` to be opened per device (for devices that support this). In effect the native library provides an easy and transparent mechanism for Java programs to access data transfer services provided by DAB appliances. Once an `MOTDataChannel` is created for a DAB receiver, all that the client Java application needs to do is keep polling the channel.

The pseudo code example in table 5.2 only shows an `MSCDataGroup` object being created from the output of the `MOTDataChannel`, but in fact any kind of data could be transmitted and received over this data link.

5.4.1 Implementation of the MOTDataChannel System

The library provides *core* code to handle splitting and assembling of `MSCDataGroup` APDUs from XPAD frame groups or packet groups. Also provided is a clean set of

```
// An undefined byte array to accept incoming data
byte[] newData ;

// We use this serialised APDU class to decode incoming data groups
MSCDataGroup newGroup ;

while ( 1 == 1 ) {
  newData = myChannel.popDataGroup() ;

  if ( newData != null ) {
    newGroup = new MSCDataGroup( newData ) ;
    // Process the new MSC data group object here ......
    }
}
```

Table 5.2: Reading `MSCDataGroup` objects from an `MOTDataChannel`

interface calls allowing each receiver/transmitter device driver to be represented as an Abstract Data Type to the rest of the library allowing new DAB receivers/transmitters to be written with relative ease.

Serial port access under Win32 (Windows 95 and NT) proved to be problematic in user mode code. The initial Win32 serial I/O was written in a multithreaded fashion with a native thread reading data from the serial port, and continually filling a circular buffer which was accessed from another thread being polled by `MOTDataChannel` objects.

This multithreaded code appeared to work, until the system was halted with a user abort (CRTL + 'C'). When this happened, the JVM cleaned up all code and exited to the host operating system. Unfortunately the serial port reader thread continued to run (outside the control of the JVM under Windows), and corrupted memory causing the whole machine to crash.

The fix for this has been to access the serial port using non-multithreading code. This proved unreliable under Windows NT, as bytes are occasionally lost from the input stream. Multithreaded access to a Win32 serial port allows the use of *overlapped read* events, which are a much more stable way to access the serial port. Single threaded access to the Win32 serial port only allows for polled access, leading to the occasional byte loss under Windows NT. However the single threaded code works well under

Windows 95.

5.4.2 MOTDataChannel Device Driver Requirements

Writing new device drivers for the library requires three function call stubs to be completed. A device initialization call, finalization call, and polling call must all be implemented. Data is passed to/read from the device driver each time the library calls the polling function. Device drivers may handle either packet mode or PAD mode data or both, the required type being specified to the driver on initialization. It is up to the Java `MOTDataChannel` class to ensure that a device driver type is available for PAD/Packet data.

Additionally two 100% Java based device drivers have been written which are not part of the core `MOTDataChannel` framework. The first of the two is a PAD receiver device driver which emulates the communication protocol used by the transmitter hardware. Using this transmitter emulating device driver, a direct null modem (RS232) connection can be made between a DABWeb client (which presents the same RS232 input interface as a transmitter device) and the normally configured DABWeb server (which believes that it is connected to a transmitter, when in fact it is directly connected to the client). This configuration is used for testing purposes in the laboratory. The second 100% Java driver is used to feed PAD services into the transmitter. Both Java drivers do not make use of the native library, instead using `java.x.comm` directly. The same software interface is provided via the `MOTDataChannel` object which contains special code to instantiate a `java.x.comm` based driver.

5.4.3 Integrating MOTDataChannel with the DABWeb Client

Since the `MOTDataChannel` class outputs complete serialized `MSCDataGroup` objects, the Java decoder and APDU classes for higher level layers of the protocol stack need only concern themselves with the caching of incomplete MOT objects.

The use of native code in the reference implementation does confer an advantage in the porting a DABWeb client to a pervasive computing device with limited memory resources. Such devices often do not have a JVM. For example a watch or palmtop organizer would be a highly desirable to use as a DABWeb client. To port the DABWeb client would require the upper layers of the MOT protocol stack to be rewritten in C, appropriate native device drivers and I/O code written for the target platform, and the whole system integrated with a browser and cache.

5.5 Device Driver Implementations

Data to be transmitted must be ready for sending each time the transmitter system makes a frame request. In PAD transport mode a 54 byte PAD frame is typically transmitted once every 24 milliseconds as part of each subchannel audio frame. Audio frames typically contain 2 bytes of F-PAD (Fixed PAD), which can be used for application specific purposes - for example broadcasting *dynamic labels* which textually describe the current audio program - and 52 bytes of XPAD data framed appropriately. PAD data is sent to the transmitter in ISO MPEG frames [45], each time the transmitter makes a frame request.

Receiver specific device drivers must be implemented as specified by each manufacturer. The RDI interface standard is common however, though special purpose hardware is required to decode PAD or Packet mode data from an RDI data stream. DABWeb uses the Fraunhofer DSP card for this purpose.

5.5.1 PAD data from the Philips 752

The rack mounted Philips 752 DAB reference receiver (see fig. 5.5) may be tuned by means of front panel controls, and PAD data is output from the currently selected subchannel by means of an RS232 interface. In addition the Philips receiver also supports RDI data output via optical and electronic interfaces. PAD data is output

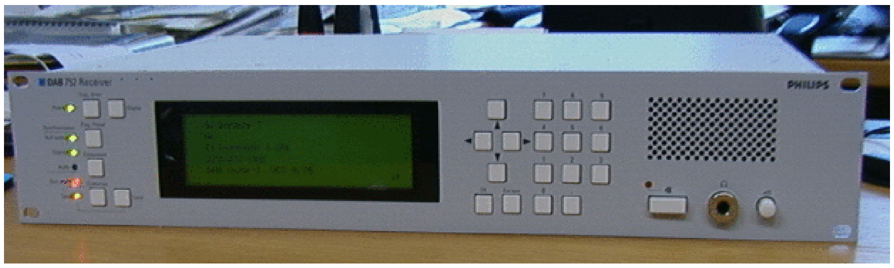


Figure 5.5: The Philips 752 reference receiver

from the receiver once every 24 milliseconds at 38400 baud in an ISO MPEG frame which closely resembles the PAD data input format of the DAB transmitter.

Even if no data is fed into the transmitter, the 752 will still output a *garbage* ISO MPEG frame of a pre-configured length, which is problematic to DABWeb should the server miss one 24 millisecond time slot. The introduction of sequence numbers to each XPAD frame, or ensuring that no XPAD frame slots are missed by the DABWeb server cures this problem.

5.5.2 The Fraunhofer DSP PCMCIA card

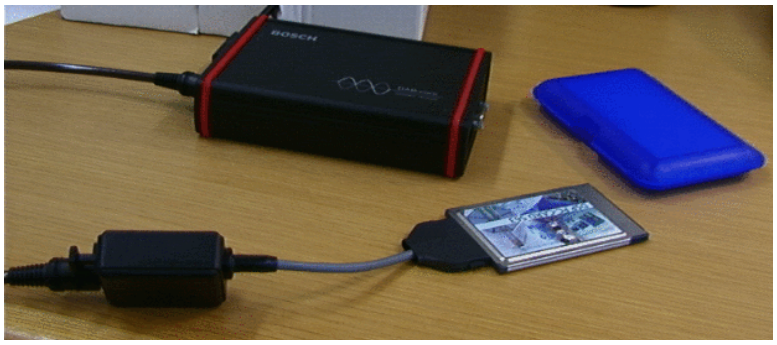


Figure 5.6: The DSP card connected to a BOSCH DAB core receiver

The DSP card from Fraunhoffer (see fig. 5.6 bottom right) is connected to an optical RDI output via an optical fiber cable and an optical-electronic extender box. Test application software is supplied with the card which can allow, for example, the audio part of a subchannel to be captured to a data file. Once captured audio data files may be played back using an MPEG audio player on the host computer.

It has been found that RDI output from the BOSCH DAB Core receiver must be set to a 20-bit configuration, which required a custom modification to the DEAPspace group's tuning software. Alternatively the card can be connected to the optical output of the Philips 752 receiver which works without any further configuration.

A Win32 software library is supplied with the card (binaries only) which provides an API for accessing ensemble components. The PAD part of any subchannel can be selected via the API, and then each relevant PAD frame received will be placed in a circular buffer by the DSP card. Since the buffer is held in shared memory between the host PC computer and the card, the DABWeb client can read and process the PAD frames.

A native driver has been written to handle decoding XPAD frames from subchannel data. Problems associated with the Philips receiver, and *phantom* frames being output when none were transmitted should not occur with the DSP card, as only valid frames are placed into the circular buffer. For this reason each frame must be encoded by the DABWeb server exactly as expected by the card (according to the DAB standard) in order for the card to decode it.

5.5.3 The BOSCH DABCore stand alone receiver unit

This unit (see fig. 5.6 top middle) is tuned via an RS232 serial link with Java software developed by the DEAPspace group. It provides analogue audio output, and also data output via an optical RDI interface. Initially BOSCH had indicated that it was interested in making the PAD data component of the current subchannel output from the receiver's RS232 port, plans which have apparently been placed on hold. A software upgrade to the receiver's flash ROM would be involved. Connection links on the

Printed Circuit Board of the DABCore allow RDI data to be extracted electronically in addition to the optical output supplied.

5.5.4 The DABCore PCI based receiver

The internal PCI card receiver from BOSCH allows the host PC to directly access ensemble data components via the host's PCI bus. Tuning software is supplied, and BOSCH have made the source tree of this software available to the DEAPspace group (for Linux). The PCI card also has an external optical RDI output.

The provided software for Windows NT includes a slide-show data service display, and also comprehensive audio tuning and capture software. As the Linux software sources to drive the card are available, a native driver will be created in the future allowing DABWeb services to be received using the card.

This card also comes equipped with an RDI optical output. Initial tests indicate that the RDI output is compatible with the Fraunhofer card with no further configuration.

5.6 Integration of the BEAST RK system with DABWeb

At the highest level of abstraction the BEAST RK system has been implemented as two visible Java objects and one interface. They provide complete access to all decryption, encryption and key management functions required by DABWeb, allowing encryption and decryption of content, and management of secret keys :

- The `BEASTKey` class - A `BEASTKey` object is used only by the Server to encrypt multimedia content, and handle the generation and storage of new private and session keys. Instances of `BEASTCard` are used by both the client and server, since both need to use the card to hold private keys.
- The `BEASTCard` class - `BEASTCard` objects allow high level calls to be made to the BEAST system for decryption operations. It encapsulates all smart card related

aspects of the BEAST system including decryption, user credit management and uploading of new secret keys to a card, which must be supplied as a `BEASTKey` object.

- The `DABWebGui` interface - This Java interface has been designed to allow the current state of a BEAST smart card to be reflected to users of DABWeb through some form of GUI. The interface informs the GUI when a new smart card becomes available or is removed, and the remaining credits on the current smart card. It is managed by the `BEASTCard` object.

In addition to server-side encryption the `BEASTKey` object is also able to decrypt content. This function only works if it holds the correct secret keys, and so was used only during development of the algorithm to ensure that content could be correctly encrypted and then decrypted. The ability of the `BEASTKey` object to decrypt required the implementation of the smart card based code on the host. This was used during development to ensure that the smart card was generating the correct result for each decryption.

5.6.1 Implementation of the BEAST RK system

Development of the BEAST RK system was subdivided into three separate software systems :

- The on-card software, which executes decryption operations using the secret keys contained in the smart card.
- The card terminal communications software which handles all communication between the smart card and the host.
- The provision of a high level software layer which handles encryption and decryption of blocks and the uploading of keys and credits to a card with simple function calls. Also required is a feedback interface which informs applications

of the current state of the smart card - whether one is present in a card terminal and ready to decrypt, and how many credits it contains.

This high level software layer providing access to the BEAST RK system is encapsulated in the `BEASTCard` object as described previously. This provides all functionality relating to the smart card through the following calls :

```
boolean decrypt( byte[] content, byte[] sessionKey )
// This function decrypts BEAST encoded content (true on success)
```

This call will decrypt content using a smart card, given that the card is present and ready to decrypt, and that the correct session key is available.

```
boolean setKeys( BEASTKey keys, int credits )
// This function uploads secret keys and a credit value to the card
```

The `setKeys` function is used in order to initialise new BEAST JavaCards (which already have the applet downloaded onto them). It takes a `BEASTKey` object, and downloads the three keys contained within that object, along with the given number of credits.

```
int credits()
// This function returns the remaining credits on the card
```

Finally the `credits` function will always return the number of credits present in the current smart card - none if no BEAST smart card is available.

Below the two high level objects (`BEASTKey` and `BEASTCard`) lie the card-host communications layer which has been implemented using OpenCard, below which lies the JavaCard applet.

5.6.2 The Development of the BEAST JavaCard Applet

Applet programs developed for a JavaCard must be written according to a defined structure. Each must have an *entry point* which is called by the Card OS when a new APDU has been received, and that applet is currently selected. The APDU entry point must be able to process the APDU, and return an appropriate response or a failure message in the correct ISO format [37]. Some data types are unavailable or only partially available to applets, notably strings and integers - only short integers may be used. In addition the JavaCard OS only supports a subset of the complete standard JavaOS library.

Memory limitations are imposed on the Java programs executing on a JavaCard. Once source code has been written for the JavaCard it is compiled into Java bytecode using the standard `javac` compiler. Next the compiled `.class` file is converted into a JavaCard specific format which removes all long string references to other Java classes. Usually a JVM binds classes to other library classes dynamically at run time. To do this the full textual name of each library class used is contained with the Java class file so that the JVM can look up the appropriate libraries.

On the JavaCard a different system is used to ensure that the minimum number of bytes are used to store applet code. The conversion of a normal Java class file to a JavaCard applet file involves removing all textual names to other classes, and replacing them with numeric tokens. The JavaCard OS has only a limited number of classes which may be dynamically bound to, and so it is a simple matter to identify each of them with a short integer or single byte.

Several other JavaCard virtual machine issues exist when every function and class name string is replaced with a token. In the case of constructor methods, another special token is needed to identify which method in a class is a constructor, since comparison between the class name string and the method name strings in a class is no longer possible.

In addition to reducing dynamic binding labels to tokens, the conversion of a Java class to a JavaCard applet file also signs the applet so that it may be downloaded to the JavaCard. A tool is provided to download applet files to the card.

5.7 The current state of the DABWeb reference implementation

The DABWeb implementation currently consists of the client and server programs, both tested and fully working, the BEAST RK JavaCard applet - also fully working - and a small utility program which is used to upload secret keys to the on-card applet. The server program can be demonstrated outputting complete web trees from WebFS - optionally encrypted - for transmission. At present transmission is only done using PAD mode, as we do not have the hardware or protocol specifications that are needed to implement full subchannel (packet mode) transmissions. More work would be needed to develop this aspect of the project. From a component view, my work on the DABWeb project has yielded a complete, tested implementation of the MOT protocol stack, the BEAST RK encryption, decryption and charging system, a client side cache manager, and the software abstraction layer for specific DAB hardware device drivers. These components have been integrated to form demonstration client and server programs.

The demonstration server program is executed at the command line. Once it has begun generating XPAD frames for transmission in PAD mode, and the transmitter has been correctly configured, receiver clients may be started. The client system being used in the laboratory is a ThinkPad 760ED, with a 133MHz processor and Windows 95. I have written and tested a device driver for the Philips 752 receiver on this platform, which along with the rest of the client software allows the reception and caching of web sites. The client system has a GUI interface which lists each file in a web site as it is decoded, so the client's progress can be monitored. Also the GUI displays a list of partially complete files that are waiting for a carousel to be broadcast again so they may be completely received. This client functionality has been tested and found to be fully working, with the maximum number of partially complete files that client will hold in memory being configurable. Both the client and server software systems make use of the `MOTDataChannel` virtual DAB device driver system for this demonstration. I have also implemented and tested a null modem device driver, which allows a client and server machine to be connected together directly by cable for laboratory testing purposes. The client side null modem driver emulates a transmitter system, allowing the usual transmitter driver to be configured on the server side.

My implementation work on the decryption system has yielded a working and tested JavaCard BEAST RK applet, and a full set of OpenCard services to deal with the BEAST RK encryption and decryption functions as detailed earlier in this chapter. Timings when calling the high level decrypt function in the `BEASTCardService` object indicate that to decrypt the first 20 bytes of content using the card takes 200 milliseconds. These timings were made using a PCMCIA smart card reader and associated PC/SC driver software which was used used beneth the OpenCard software. The `BEASTCard` object provides a high level function call to decrypt an entire array of content given that array along with the 20 byte session key. The `BEASTCard` object coordinates the decryption effort making use of the `BEASTCardService` object for the on-card part, and then completes the off-card part using the SEAL algorithm.

The `BEASTCard` code has been integrated with the DABWeb server program allowing web trees to be encrypted before broadcast. This integrated server system has been tested and was found to work. The BEAST RK decryption system has also been fully integrated with the client program, and has been tested together with a PCMCIA smart card reader decrypting a real-time PAD data stream that has been fed to the ThinkPad from the Philips receiver. The client system has been found to work well, and completely fulfilled the design requirements set out at the beginning of this chapter. Another additional Java program called `BEASTtest` is used to upload keysets and credits to the BEAST RK applet on a JavaCard.

Further work is needed on the scheduler part of the server system, which is rudimentary at present. The locations of web trees to be broadcast by the server are currently hard coded into the Java source of the server program, and web trees are then broadcast in a round robin looped fashion. A more advanced scheduling system would be needed for a full scale DABWeb trial which allowed broadcasters to quickly add or remove web sites from the schedule. A server GUI would also be useful. On the client side, DABWeb drivers for the Fraunhofer DSP 536 card, and the BOSCH PCI receiver card are currently incomplete, and further work is needed to make these drivers operational.

The use of Java to implement the central components of the DABWeb system has been successful. There were concerns that the use of a virtual machine with intermittent garbage collection enabled would make the performance of real time communications software unacceptably slow but this has not happened. Both the client and server have proved to be capable of handling the relatively low throughput requirements for decoding PAD data services. It should also be noted that much of the decoding work is actually performed by the lower level C device driver code, which outputs complete `MSCDataGroup` objects for further processing in Java. The hybrid Java/C approach has worked well, with the demonstration client system being able to robustly multitask on the Windows 95 ThinkPad whilst decoding an incoming stream of web sites.

Chapter 6

Conclusion

The completed DABWeb system provides a demonstration platform for the viability of wireless data services over DAB networks. The Java software implemented compiles to a collection of JAR[1] libraries which can quickly be incorporated into other data broadcast projects, following the reusable component philosophy of Java.

The initial work presented here could quickly be combined with other wireless technologies so that a back channel can be incorporated into the system (much like MEMO [33]) where user download requests could allow content to be scheduled *juke box* style. In this way many users can surf in a interactive manner, with experimental trials needed to determine the latency and scaleability of such a system.

DABWeb could also be adapted to handle other content types, for example to broadcast software updates. For this type of application, DABWeb client software could be extended to allow the latest versions of software packages to be installed with no user intervention. The encryption system would have to be expanded to allow such upgrades to be targeted at specific clients - presumably those who own the software being upgraded - additionally allowing the clients to trust the source of the software updates.

Since a modular strategy has been employed through development, this system can

[1] Java Archive

be quickly rebuilt to demonstrate different possible configurations. The reuse of components should also allow rapid development of in-car DAB data services, where only a new receiver driver would be required, and integration between the existing DAB-Web system and a PAN to network between the car and other pervasive computing devices.

Bibliography

[1] BackWeb, inc. http://www.backweb.com/, March 2000.

[2] Bluetooth homepage. http://www.bluetooth.com/.

[3] Allison C., Huang F., and Livesey M.J. Object Coherence in Distributed Interaction. In Correia, Chambel, and Davenport, editors, *Multimedia'99*, http://www.egmm99.di.fct.unl.pt/, 1999. Springer-Verlag: New York.

[4] Allison C., P. Harrington, F. Huang, and Livesey M.J. Scalable Services for Resource Management in Distributed and Networked Environments. In *SDNE'96*, pages 98–105. Macau, IEEE Press, June 1996.

[5] Fuhrhop C., Kraft A., and Kubis R. Object Carousel Simulator for Broadcast Applications. In Chambel Correia and Davenport, editors, *Multimedia'99*, http://www.egmm99.di.fct.unl.pt/, 1999. Springer-Verlag: New York.

[6] Su C. and Tassiulas L. Broadcast scheduling for information distribution. In *Proceedings of IEEE INFOCOM*, Los Alamitos, CA, USA, April 1997. IEEE Computer Society Press.

[7] Horstmann C.S. and Cornell G. *Core Java - Volume 1 - Fundamentals*. Prentice Hall PTR & Sun Microsystems Press, 1997.

[8] Perkins C.S., Hodson O., and Hardman V. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network*, 1998.

[9] Flanagan D. *Java In A Nutshell*. O'Reilly & Associates, Inc. (USA), 1997.

[10] Gifford D. Polychannel systems for mass digital communication. *Communications of the ACM*, 33(2), February 1990.

[11] Gifford D., Lucassen J., and Berlin S. The architecture for large scale information systems. In *ACM Symposium on Operating System Principles*, pages 161–170, 1985.

[12] DVB; Framing structure, channel coding and modulation for digital terrestrial television. Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France. EN 300 744.

[13] J. Postel Ed. Transmission Control Protocol. RFC 793.

[14] Digital Audio Broadcasting (DAB) Distribution interfaces; Ensemble Transport Interface (ETI). Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France, September 1997. ETS 300 799.

[15] Eureka. A Europe-wide Network for Industrial Research and Development. http://eureka.belspo.be.

[16] Eureka 147 WG D RDI Task Force. Digital Audio Broadcast System Specification of the Receiver Data Interface (RDI). Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France, November 1996. Issue 1.4.

[17] Herman G., Gopal G., Lee K., and Weinrib A. The Datacycle architecture for very high throughput database systems. In *Proceedings of ACM SIGMOD*, May 1994.

[18] Robert Bosch Multimedia-Systeme GmbH and Co. KG. Bosch Multimedia Homepage. http://www.boschmultimedia.de/, August 1999.

[19] INRIA. Web Canal Documentation. http://www.inria.fr/, 1998.

[20] Gemmell J., Gray J., and Schooler E. Fcast Multicast File Distribution. *IEEE Network*, 14(1):58–69, 2000.

[21] Macker J. and Dang W. The Multicast Dissemination Protocol Framework. Technical report, IETF Draft, 1996.

[22] Wong J. Broadcast delivery. *Proceedings of the IEEE*, 76(12), December 1988.

[23] Ljungquist J.H. Transport protocols for IP-Traffic over DVB-T. Master's thesis, Royal Institute of Technology, Department of Teleinformatics, Computer Communications Laboratory, Stockholm, 1999.

[24] Savetz K., Randall N., and Lepage Y. *Mbone : Multicasting Tomorrow's Internet*. IDG : London, 1998.

[25] Tan K. and Xu J. Energy efficient filtering of nonuniform broadcast. In *Proceedings of the 16th International Conference in Distributed Computing Systems*, pages 520–527, 1996.

[26] Rizzo L. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communication Review*, 27(2):24–36, 1997.

[27] Vicisiano L. and Crowcroft J. One to Many Bulk-Data Transfer in the Mbone. In *HIPPARCH'97*, Uppsala, Sweeden, 1997.

[28] The Linux Operating System. http://www.linux.org.

[29] Ammar M. Response time in a Teletext system: An individual User's perspective. *IEEE Transactions on Communications*, 35(11):1159–1170, 1987.

[30] Ammar M. and Wong J. The design of Teletext broadcast cycles. *Performance Evaluation*, 5, November 1985.

[31] Ammar M. and Wong J. On the optimality of cyclic transmission in Teletext systems. *IEEE Transactions on Communications*, 35(1):68–73, January 1987.

[32] Marimba, inc. http://www.marimba.com/, July 1997.

[33] The ACTS Project AC 054 - Multimedia Environment For Mobiles (MEMO). http://memo.lboro.ac.uk/, 1999.

[34] Digital Audio Broadcast (DAB) Multimedia Object Transfer (MOT) Protocol. Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France, September 1998. EN 301 234 v1.2.1.

[35] Negroponte N. *Being Digital.* Coronet Books, Hodder and Stoughton, A division of Hodder Headline PLC, 338 Euston Road, London NW1 3BH, 1995.

[36] Vaidya N. and Hameed S. Data broadcast in asymmetric wireless environments. In *First International Workshop on Satellite-based Information Services (WOSBIS)*, November 1996.

[37] The International Standards Organization. ISO 7816 Smart Card Communications Standard. Technical report, ISO, 1 Rue de Varembe, Case Postale 56, CH-1211 Geneva 20, Switzerland.

[38] Zwahlen P. Design and Implementation of a Secure Web-Based File System (Crypto) WebFS. Master's thesis, Eurecom Instititе, Corperate Communications Department, http://www.eurecom.fr/Corporate/, June 1999.

[39] Eureka 147 Partners. Radio broadcasting systems; Digital Audio Broadcast (DAB) to mobile, portable and fixed receivers. Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France, May 1997. ETS 300 401 Ed.2.

[40] PC Smart Card Work Group homepage. http://www.pcscworkgroup.com, 1999.

[41] Philips Electronics N.V. ASA Laboratory Eindhoven. http://www.sv.Philips.com/DAB, August 1998.

[42] Pointcast, inc. http://www.pointcast.com/, March 2000.

[43] Cooperstock J R. and Kotsopoulos S. Why Use a Fishing Line When You Have a Net? An Adaptive Multicast Data Distribution Protocol. In *USENIX'96*, 1996.

[44] Jain R. and Werth J. Airdisks and AirRAID : Modeling and scheduling periodic wireless data. *Computer architecture news*, 23(4):23–28, September 1995.

[45] Rhode and Schwarz. Rhode and Schwarz Internet Homepage - http://www.rsd.de. RHODE and SCHWARZ Gmbh and Co. KG Muhldorfstrasse 15, D-81671 Munich.

[46] Acharya S. *Broadcast Disks: Dissemination-based Data Management for Asymmetric Communication Environments*. PhD thesis, Brown University, May 1998.

[47] Hameed S. and Vaidya N. Log-time algorithms for scheduling simgle and multiple channel data broadcast. In *Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM)*, September 1997.

[48] Lucks S. BEAST : A fast block cipher for arbitrary blocksizes. In *IFIP Conference on Communications and Multimedia Security*, pages 144–153, http://th.informatik.uni-mannheim.de/m/lucks/papers.html, 1996. Chapman & Hall.

[49] McCanne S. Scalable Multimedia Communication Using IP Multicast and Lightweight Sessions. *IEEE Internet Computing*, 3(2):33–45, 1999.

[50] McCanne S. and Jacobson V. VIC: A Flexible Framework for Packet Video. *ACM Multimedia : San Francisco*, pages 511–522, November 1995.

[51] Fraunhofer Institut Integrierte Schaltungen. DSP PC-Card 563. http://www.fhg.de/english/index.html, 1998.

[52] Secure Systems Group, IBM Zurich Research Laboratory. IBM Smart Card for e-Business - JavaCard. Saumerstrasse 4, Rueschlikon CH-8803, Switzerland - http://bluez.zurich.ibm.com, 1999.

[53] Digital Audio Broadcasting (DAB) Distribution interfaces; Service Transport Interface (STI). Technical report, European Telecomunications Standards Institute - http://www.etsi.com/, F-06921 Sophia Antipolis CEDEX - France, December 1998. ETS 300 797 - V1.1.1.

[54] Berners-Lee T., Fielding R., and Frystyk H. Hypertext Transfer Protocol - HTTP/1.0. RFC 1945, May 1996.

[55] Bowen T., Gopal G., Herman G., Hickey T., Lee K., Mansfield W., Raitz J., and Weinrib A. The Datacycle architecture. *CACM*, 35(12), December 1992.

[56] Chiueh T. Scheduling for broadcast-based file systems. In *Proceedings of MOBIDATA Workshop*. Rutgers University, 1994.

[57] Imielinski T. and Badrinath B. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10), October 1994.

[58] Imielinski T., Viswanathan S., and Badrinath B. Energy efficient indexing on air. *SIGMOD*, May 1994.

[59] Imielinski T., Viswanathan S., and Badrinath B. Power efficient filtering of data on air. In *Proceedings of EDBT Conference*, 1994.

[60] The Teracom Group - http://www.teracom.se/. Medborgarplatsen 3, Box 17666 S-118 92, Stockholm.

[61] The Microsoft Corporation. WebTV - http://www.webtv.net/. Microsoft Corporation, One Microsoft Way, Ste.303, Redmond, WA 98052-8303.

[62] The OpenCard Consortium. The OpenCard Framework. http://www.opencard.org, 1999.

[63] Reports on the introduction of DAB in Switzerland. http://www.dab.ch/, August 1999.

[64] Hansmann U., Nicklous M.S., Schack T., and Seliger F. *Smart Card Application Development Using Java*. Springer (Awaiting Publication), 1999.

[65] WorldSpace Corporation. WorldSpace Radio Homepage - http:// www.worldspace.com/. 2400 N Street, Washington DC 20037 USA.