

Evaluation of Capuchin Application Programming Interface

Implementing a Mobile TV Client

XUAN FENG



**KTH Information and
Communication Technology**

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:97

Evaluation of Capuchin Application Programming Interface: Implementing a Mobile TV Client

Xuan Feng

07 August 2009

Examiner: Professor Gerald Q. Maguire Jr., KTH
Supervisor: Nils Edvardsson, Ericsson

School of Information and Communication Technology
Royal Institute of Technology
Stockholm, Sweden

Abstract

The purpose of this research was to evaluate the Capuchin API launched by Sony Ericsson at Lund, Sweden in 2008. The Capuchin API bridges Adobe's Flash graphics and effects with JSR support from Java ME. We evaluated Capuchin API with regard to its suitability for a Mobile TV application. We tested this API in Ericsson's TV lab where we had access to live TV streams and online multimedia resources by implementing a Mobile TV client. This test application was named "Min TV", in English: "My TV".

Using Capuchin in the Ericsson TV lab environment has shown that it has some benefits, but also has many drawbacks. The Flash developer can be used to create an animated user interface and Java developers can do complex programming. At this early stage Capuchin technology is not mature enough, nor is it suitable for Mobile TV client development. Only after Sony Ericsson adds features such as soft keys, easier debugging of Flash Lite standalone applications, test emulator support in the software development kit, and more data communication methods than string and number, only then it will be a suitable technology for Mobile TV applications.

Ericsson's current Mobile TV application client was built using a framework called ECAF, which supports a graphics frontend and Java ME as backend. We compared ECAF and Min TV with respect to parameters such as: flexibility, performance, memory footprint, code size, and cost of skinning. (All these parameters are explained in detail in the methodology chapter.)

As a possible future technology for Mobile TV, we evaluated a number of different presentation/graphics technologies including HECL, SVG Tiny, MIDP 3.0, .NET Compact Framework, etc. Moreover, we examined if a pure Flash Lite client application is a viable solution for Mobile TV. The comparison of different presentation technologies showed that Java ME is a comprehensive platform for mobile development offering all the necessary support from third party graphical user interface makers. .NET CF also looks like a good option for development with the scaled down capabilities for different programming languages supported using CLR.

Keywords

Capuchin API, JAVA ME, Action Script 2.0, VoD, Linear TV, Min TV, Mobile TV, presentation technologies, experimental evaluation

Sammanfattning

Syftet med denna forskning var att utvärdera Capuchin API lanserades av Sony Ericsson i Lund, Sverige 2008. Den Capuchin API broar Adobe Flash grafik och effekter med JSR stöd från Java ME. Vi utvärderade Capuchin API med avseende på dess lämplighet för ett mobil-tv ansökan. Vi testade detta API i Ericssons TV lab där vi hade tillgång till TV-strömmar och online multimediaresurser genom en mobil-TV-klient. Detta test ansökan hette "Min TV", på engelska: "My TV".

Använda Capuchin i Ericsson TV lab miljö har visat att det har vissa fördelar, men också många nackdelar. Flash-utvecklare kan användas för att skapa en animerad användargränssnitt och Java utvecklare kan göra komplexa programmering. På detta tidiga stadium Capuchin tekniken inte mogen, det är inte heller lämpliga för mobil-TV-klient utveckling. Först efter Sony Ericsson lägger till detaljer såsom mjuka nycklar, enklare felsökning av Flash Lite fristående program, testa emulator stöd i Software Development Kit, och mer data kommunikationsmetoder än string och antal, först då kommer det att vara en lämplig teknik för mobil-TV-program .

Ericssons nuvarande mobil-tv ansökan klient byggdes med hjälp av en ram som kallas ECAF, som stöder en grafiska gränssnittet och Java ME som backend. Vi jämförde ECAF och min TV med avseende på parametrar såsom flexibilitet, prestanda, minne fotavtryck kod storlek och kostnaden för avhudning. (Alla dessa parametrar förklaras i detalj i den metod kapitel.)

Som en möjlig framtida teknik för mobil-TV Vi utvärderade ett antal olika presentation / grafik teknik inklusive HECL, SVG Tiny, MIDP 3.0., NET Compact Framework, etc. Dessutom har vi examad om en ren Flash Lite klientprogrammet är en hållbar lösning för mobil-TV. Jämförelsen mellan olika presentation teknik visade att Java ME är en övergripande plattform för mobila utvecklingen erbjuder allt nödvändigt stöd från tredje part grafiskt användargränssnitt beslutsfattare. . NET CF också ser ut som ett bra alternativ för utvecklingen med ned kapacitet för olika programspråk som stöds med hjälp av CLR.

Nyckelord

Capuchin API, Java ME, Action Script 2.0, VoD, Linjär TV, Min TV, mobil-TV, presentation teknik, experimentell utvärdering

Acknowledgements

First, I express sincere gratitude to my examiner Professor Gerald Q. "Chip" Maguire Jr. His encouragement and suggestions helped me a lot to start and complete the thesis. He told me not to be panic, so that I would always see the possibility of success in this thesis project. He also suggested me to write something everyday; as a result I had a lot of raw material for the thesis. I also deeply thank my supervisor Nils Edvardsson at Ericsson for his continuous support and encouragement. He taught me how to think critically and showed me a different way to approach research problems. I am greatly indebted to Christian Olofsson (Consultant to Ericsson from Tactel AB) for extending his extreme support in helping me start my project and in setting up the mobile lab environment.

Thanks to all the employees at Ericsson who offered their technical support and expertise related to the CTV lab and networking. I would like to thank my manager Hans Bystörm for being a very delightful person in the department. In short, I feel lucky for having the privilege and the opportunity to work with real experts in a zealous working environment.

Last, but not least, I would like to thank my family and friends for their love and presence. I pay my gratitude to my father for the motivation he gave me, my mother for her commendable prayers & wishes, and rest of my family for their support and love for me.

Contents

Abstract	i
Sammanfattning	ii
Acknowledgements	iii
Contents	iv
List of Figures	vi
List of Tables	vii
List of Abbreviations	viii
1 Introduction	1
1.1 Aim of this research	1
1.2 Scope	1
1.3 Delimitations	2
1.4 Contribution	2
1.5 Structure of Thesis	2
2 Background	3
2.1 Mobile TV	3
2.2 Linear TV versus Video on Demand	3
2.3 ECAF	4
2.4 What is Project Capuchin?	4
2.5 Flash Lite versus Java ME attribute comparison	5
2.6 Different approaches for using Project Capuchin	6
2.7 SWF2JAR	7
2.8 Development	7
3 Min TV development	9
3.1 Min TV Communication Overview	9
3.2 Mobile handset and CTV lab setup	10
3.3 Setting up java environment for Project Capuchin	12
3.4 Implementation details	12
3.4.1 Using Capuchin API	12
3.4.1.1 The Flash Lite part	12
3.4.1.2 The java part	15
3.4.1.3 Data transfer in Min TV application	20
3.4.2 Making the Internet available to the mobile via a Laptop PC	21
3.4.2.1 Settings for C905	21
3.4.3 Development in Action Script 2.0	22
3.4.3.1 Pure Action Script files	22
3.4.3.2 Action Script in frames	23
3.4.3.3 Setting the Property	25
3.4.4 Compiling the application using Another Neat Tool (ANT)	26
3.4.4.1 Building using ANT	26
3.4.5 Debugging and Testing the Application	27
3.4.5.1 Testing Tools	27
3.4.5.2 Testing the streams	30

4	Findings and Discussions	31
4.1	Evaluation of Capuchin API	31
4.1.1	Advantages	31
4.1.2	Disadvantages	31
4.1.3	Conclusion	33
4.2	Capuchin versus ECAF	33
4.2.1	Animation capabilities.....	34
4.2.2	Code size in lines of code	34
4.2.3	Memory footprint	36
4.2.4	Reusability.....	37
4.2.5	Flexibility	37
4.3	Presentation technologies for mobile TV	37
4.3.1	Hecl support for Java ME	38
4.3.1.1	HTTP	38
4.3.1.2	K-XML	38
4.3.1.3	Other Hecl extensions	38
4.3.1.4	Evaluation of Hecl + Java ME for our Mobile TV application	39
4.3.2	SVG Tiny 1.2 support for Java ME	39
4.3.3	MIDP 3.0 [32]	40
4.3.4	Microsoft .NET Compact Framework (.NET CF).....	40
4.4	Pure Flash Lite TV Client Application Concept	40
5	Conclusions and future work	43
5.1	Conclusions	43
5.2	Future Work	43
	References	45
A.	Configuration files	47
A.1.	Fetching and inserting EPG data.....	47
A.2.	Loading an EPG into the EPG Aggregator.....	47
A.3.	Procedure	47
A.4.	CTV Properties file	48
A.5.	Problems and solutions.....	49
B.	Phone configurations	50
B.1.	8.1 megapixel Cyber-shot phone with Project Capuchin API	50
B.2.	Phone Model	50
B.3.	Platforms.....	50
B.4.	Screen Sizes	50
B.5.	JSRs & APIs.....	50
B.6.	Audio & Video.....	51
B.7.	Flash.....	51
B.8.	SVG.....	52
B.9.	Connectivity.....	52
B.10.	Regions	52
B.11.	Miscellaneous.....	52
C.	Source code for the ActionScript files	53
C.1.	retrieveInfo.as.....	53
C.2.	Channel.as	54
C.3.	Program.as	54
C.4.	XDList.as	55
C.5.	Session.as	57
C.6.	retrieveVodInfo.as.....	58
C.7.	vodProgram.as	58

List of Figures

Figure 1: Mobile TV Streams delivery to mobile	3
Figure 2: Capuchin High level diagram (Adapted from [12])	5
Figure 3: SWF2JAR compiling tool (Adapted from [13])	7
Figure 4: Min TV Communication Overview	9
Figure 5: C905, First Capuchin Enabled handset	10
Figure 6: CTV lab network configuration	10
Figure 7: Generating the EPG data	11
Figure 8: Setting the security level for the emulator	11
Figure 9: First Window	13
Figure 10: Set Screen Size	13
Figure 11: Design Interface	14
Figure 12: A sample .swf file	14
Figure 13: Sample Capuchin code for loading a .swf file	15
Figure 14: Min TV import statements in Java ME	16
Figure 15: StartApp() code for MIDlet startup	17
Figure 16: VoD xml accessing and parsing	18
Figure 17: Playing the video Stream	19
Figure 18: Data flow between Flash and Java	20
Figure 19: Java event notification to Flash	21
Figure 20: Settings for Windows Vista	22
Figure 21: Coding for a Specific Frame	23
Figure 22: Code for onKeyDown	24
Figure 23: Code for DataRequest and ExtendedEvents	24
Figure 24: Variable named currentTime	25
Figure 25: Code for the frame shown above	25
Figure 26: Time displayed in the upper right corner	25
Figure 27: Build.xml for Min TV	26
Figure 28: DebugMux while debugging using C905 DeviceExplorer	27
Figure 29: Connection proxy connecting C905	28
Figure 30: Setting the static IP address to the handset	28
Figure 31: Ericsson's Device Explorer showing output for the EPG fetcher MIDlet from C905	29
Figure 32: Running a VoD resource in VideoLAN's VLC player	30
Figure 33: ECAF Screen.xml	35
Figure 34: ECAF Screen1.xml	36

List of Tables

Table 1: Comparison between Flash Lite and Java ME	5
Table 2: Action Script files	23
Table 3: Relation between ECAF tags and Flash Lite	34

List of Abbreviations

ANT	Another Neat Tool
API	Application program interface
APN	Access Point Name
AS	Application Server
CODEC	Coder/Decoder
CTV	Converged TV
DataRequest	Request for data delivery from Action Script to Java ME
ECAF	Ericsson Client Application Framework
EPG	Electronic Program Guide
ESG	Electronic Service Guide same as Electronic Program Guide
ExtendedEvent	Event notification from Java ME to Action Script
FLV	Flash Video
GUI	Graphical user interface
HTTP	Hypertext transport protocol
IDE	Integrated development environment
IMS	IP Multimedia Subsystem
Java ME	Java Micro Edition (also as J2ME)
JRE	Java Runtime Environment
JSR	Java standardization request
JVM	Java Virtual Machine
LTV	Linear TV
LWUIT	LightWeight User Interface Toolkit
MMAPI	Mobile Media API (JSR-135)
MTV	Mobile TV
OMA	Open Mobile Alliance
OMA BCAST	Open Mobile Alliance Broadcast
RTSP	Real-time streaming protocol
SDK	Software development kit
SEMC	Sony Ericsson Mobile Communication
SIP	Session Initiation Protocol
SVG	Scalable Vector Graphics
TV	Television
UI	User Interface
VoD	Video on Demand
XML	Extensible mark-up language
3G	Third-generation mobile system

3GP	3GPP file format for multimedia
3GPP	Third Generation Partnership Project
.class	A blueprint to create objects
.fla	File extension for an Adobe (Macromedia) Flash source document
.NET CF	Microsoft .NET Compact framework

1 Introduction

With the evolution of technology, mobile phones have become more and more functional. Besides the basic function of making phone calls, sending and receiving SMS, etc., a lot of interesting and useful mobile applications have been developed to provide a better user experience. Users have the opportunity to check their e-mail or surf the Internet via their mobile phone. In the meanwhile, mobile TV has been introduced to allow users to watch television, movies, or other on-line multimedia on their phone. However, an unresolved question is how best to design a mobile TV client. In recent time, this has become the main concern for many mobile developers.

1.1 Aim of this research

The aim of this research project has been to explore new presentation technologies, to exploit the new graphics capabilities of modern cellular phones, and to find new ways to present data to the end user. Earlier, the Ericsson Client Application Framework (ECAAF) was used to create the existing Ericsson Mobile TV client. In our work we were looking for an alternative solution to replace ECAAF as it is not sufficiently scalable in terms of graphics and programming. Additionally, developers need to write a lot of code to create a front end in ECAAF.

The desire is to find an alternate technology which supports easy-to-develop graphics and effects with excellent programming capabilities. The first option which comes to mind is Adobe's Flash Lite [1], as it offers rapid graphics development and well established designer tools. Additionally, using Java provides a good base for services, numerous security functions, etc. Fortunately, Sony Ericsson's Capuchin Project has developed an API bridging the gap between Flash Lite and Java ME, by encapsulating Flash Lite content in Java ME applications. Thus Flash Lite can handle the presentation layer issues; while Java is used to feed the presentation layer with the necessary data. Therefore, in this project we opted to explore the flexibility of Capuchin API for a Mobile TV application.

1.2 Scope

There are three main parts of this thesis.

1. The first part is to evaluate the Sony Ericsson's Capuchin API [2], i.e. to evaluate what is good, what is bad, and what is missing from this API – with regard to a Mobile TV application.
2. The second part is to design and implement a mobile TV client using the Capuchin API and Java ME (for the backend communication). The frontend will be pure Flash using Flash Lite and Action Script 2.0. The Capuchin enabled client was implemented using Ericsson's Mobile TV environment. Thus the client can access Oracle (formerly Sun Microsystems) Application Servers (AS) in an IP Multimedia System (IMS) [3]. The user can browse the schedules of programs similar to those found in a newspaper, then select and watch their media selection. They can also see what videos are currently hot (i.e., the most in demand).

Two Flash graphical user interfaces (GUIs) have been implemented. One of the GUI themes has simple graphics and the other one has slightly fancier graphics. The presentation of the data, color management, and access to the streams are different in the second theme. The actual media content is displayed using Flash Lite. These Flash GUIs serve as examples of different styles of interfaces, having different color palettes, fonts, and screen layouts.

3. The third part is to compare different GUI technologies for Mobile TV. We examined and compared: HECL, SVG Tiny 1.2, .NET Compact Framework (CF) and MIDP 3.0 as possible front ends for a Mobile TV application.

1.3 Delimitations

The existing Ericsson mobile TV client is called the Converged TV client. It is IP Multimedia Subsystem (IMS) enabled; this that the client gets all its information through IMS by using Session Initiation Protocol (SIP) signaling.

IMS was designed to help operators develop cost-effective networks that allow the convergence of voice and data. It has been position by Ericsson (and other) as the key to building networks that can combine ongoing communication sessions with multimedia elements, such as sharing live videos while simultaneously supporting conversational voice communication. Ericsson's IMS Solution is a complete end-to-end offering for fixed and mobile operators that combine media such as voice, text, pictures, and video while giving users the tools to personalize their communication experience [3]. In our implementation, our client should have some of the Converged TV client's functionality and use the same communication method as well (i.e., use IMS). However, for our prototyping and testing we did **not** use SIP signaling to the IMS, due to lack of time and insufficient technical support with respect to the programming necessary to establish network connections. Additionally, we were unable to test the second Flash theme on real-time data from the application servers due to an upgrade of lab configuration (that was outside of our control).

1.4 Contribution

This research examines the question of whether the Capuchin API is suitable for mobile TV application development. If the answer is yes, then a secondary goal is to enumerate the benefits of using this technology and what are the disadvantages of using this technology. We will also compare this technology to other suitable presentation technologies for the Mobile TV, both for playing TV streams and for other on-line multimedia.

1.5 Structure of Thesis

This report consists of six chapters. The first chapter set out the goals and limitations of the project. Chapter two provides the reader with background necessary for the subsequent chapters, specifically concerning mobile TV, Linear TV (LTV), Video on Demand (VoD), and the basics of an Electronic Program Guide (EPG). It describes the existing ECAF based Mobile TV client, i.e., the Ericsson Converged TV client.

Chapter three presents a detailed overview of Ericsson's Capuchin API. It describes a number of different approaches for using the Capuchin API. The chapter concludes by explaining the six steps necessary to develop a Capuchin application.

Chapter four described the research methodology that was used. It provides detail about the development environment and how we conducted the research. This includes details of the design, implementation, debugging, and testing of the prototype ("Min TV") client.

The testing and evaluation of this new client are discussed in chapter five. Other potential presentation technologies and the existing ECAF based Converged TV client are compared with our Capuchin enabled Min TV client. This chapter enumerates in detail the benefits and the drawbacks of the Capuchin API for a Mobile TV client.

Chapter six summarizes our work and gives some concluding remarks about the Capuchin API's usefulness as well as suggests some potential future work.

2 Background

2.1 Mobile TV

Mobile TV is a service which allows cellular phone owners to watch television on their phones from a service provider. In our Mobile TV project we receive 10 streams in MPEG2-TS [4] from Alcom [5]. Each of these ten streams is being sent at 4 Mbits/s which is much faster than is desirable for the mobiles (as this would be taxing for current mobile devices and take a lot of the operator's capacity). Thus these streams are transcoded to a lower bit rate to make them more suitable for mobile phones (see Figure 1). The resulting data rate is ~200Kbits/s per stream. This specific data rate was chosen as it is a 3GPP [6]. file format (with the extension ".3GP") for a video stream encoded in H.264 [7] with audio AAC. The node labeled "EMTV" in the figure is a streaming server which simultaneously serves several mobiles with VoD content.

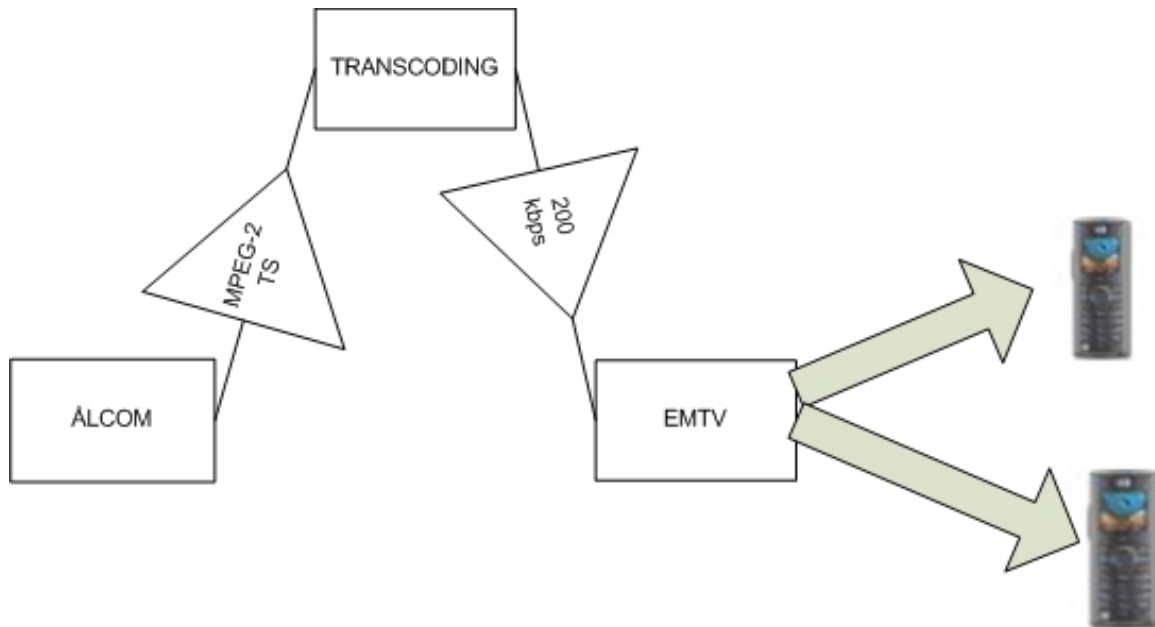


Figure 1: Mobile TV Streams delivery to mobile

2.2 Linear TV versus Video on Demand

A Linear TV (LTV) service allows end-users to watch what is currently being transmitted to all subscribers. This can be used for broadcast TV channels and programs. The end-user can choose to display the TV portal, select other services, or choose other TV channels. The end-user can learn what programs are available by an Electronic Program Guide (EPG). The EPG presents the programs of different TV channels. It includes information about the name of the program, when it will be transmitted, a description of the program, and the URL of the stream associated with this program.

Video on Demand (VoD) [8] is an interactive multimedia system. The customer selects a movie (or other prerecorded multimedia content) from a large multimedia database. VoD allows the user to select the content that they are interested in and the media playout is personalized, i.e., the user can pause, play, fast forward, and rewind their media selection. This allows end-users to have complete control over what they watch, when they watch it, and even how watch it. Thus there is no need to wait for a set time to watch your choice of movie; instead, you simply make your selection, then hit the play button when you are ready [9].

The format for the media to be shown on the mobile is 3GP. 3GP files are in an encoded, compressed format. A user can choose different versions content – offering high definition or lower definition (with corresponding higher and lower throughputs being required to deliver the content to the user). Further details of this format can be found in [6].

2.3 ECAF

As mentioned earlier Ericsson's current Mobile TV application client was built using a framework called Ericsson Client Application Framework (ECAF). ECAF supports graphics for creating a front end and Java ME as a backend. All graphical components and user interaction in applications developed with ECAF are defined in XML-pages which are written and organized according to how the ECAF interprets the XML-logic. Compared with Internet browser, ECAF parses XML-pages while a general browser parses HTML-pages. And one more difference between XML and HTML is that XML allows for any sophisticated user interaction while HTML doesn't. Except for this, they have many similarities. As for an example the XML-pages, that defines how the applications look and feel, how the users interact as well as all the other application contents, may be put on a server. ECAF is not used to just parse any XML-pages. The XML-page has to be encoded especially for ECAF. As a result, the designer needs to know what kind of structure to design and what kind of tags to use [9].

The text between `<!--` and `-->` will be ignored when ECAF parses the XML-document, thus it will **not** have an affect on the appearance or function of the application. The four main tags are: `xml`, `ECAF`, `layer`, and `text`. The `xml`-tag serves to identify which version of XML ECAF is being used. ECAF needs the `ECAF`-tag to be able to parse the XML correctly. Those two tags must always be present in all XML-documents and must be written in a particular order. All components of the application should be placed inside the `ECAF`-tag. Everything that is supposed to be displayed by the application has to be put inside a `layer`-tag. ECAF is able to work with several layers at the same time. A number of graphical interactive tags are also available, these include: `event`, `image`, `group`, `link`, `open`, and `move` tags.

2.4 What is Project Capuchin?

Project Capuchin is a technology developed by Sony Ericsson to provide a high quality GUI and good programming support [11]. Since it is developed by Sony Ericsson, it is available in their phones released after June 2008. The first phone in this series was model C905.

Project Capuchin is a Java ME API which makes it possible for Java to run a Flash Lite content file (.swf) and to display the output on the phone's screen. An important aspect of this method is mixing the two worlds of Flash Lite and Java ME, and enforcing the relationship between User Interface (UI) designers and developers [12]. Figure 2 shows a high level architecture presentation of Capuchin. As can be seen in the figure, the user interface is designed and implemented in Adobe Flash Lite while the application's semantics are implemented in Java ME.

All system events such as key events are forwarded from Java to Flash Lite and the Flash Lite player will process these events. If Flash Lite needs to access some information, it asks Java for help. Communication between Flash Lite and Java are handled through a middleware (working as a translator). This middleware class listens to Flash Lite requests, transfers the requests to Java, and sends the response back to Flash Lite. The data transfer between Flash Lite and Java is bi-directional: Flash Lite can send requests to and receive events from Java; while Java can send events to and listens to requests from Flash Lite.

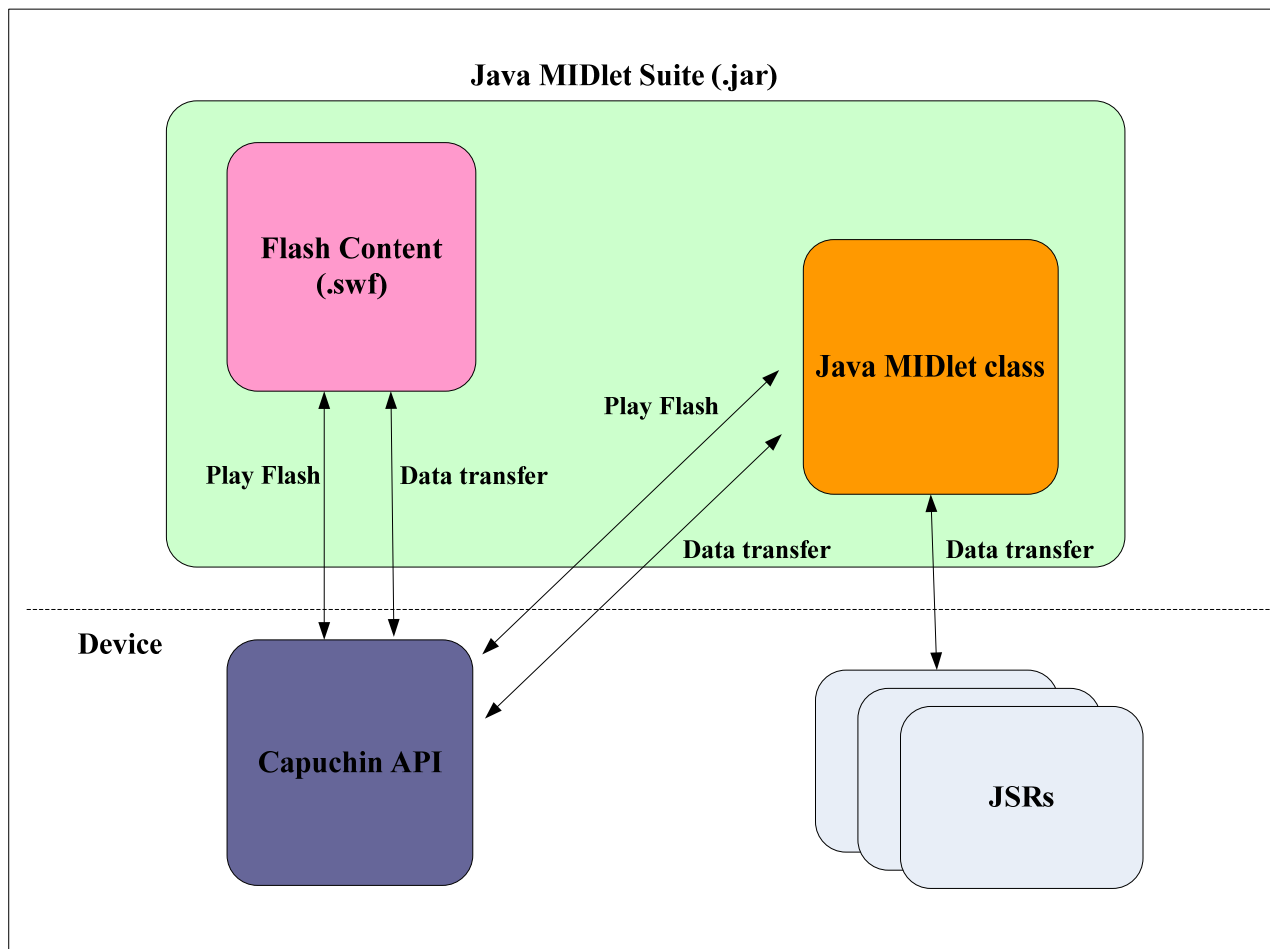


Figure 2: Capuchin High level diagram (Adapted from [12])

2.5 Flash Lite versus Java ME attribute comparison

In order to find the strengths of two technologies and to know their weaknesses, the best method is to compare them with each other. Since the Capuchin API is using both the technologies, we should examine what is best for each of these two technologies. A simple comparison is shown in Table 1.

Table 1: Comparison between Flash Lite and Java ME

Flash Lite	Java ME
Pros Drag-and-drop toolbox Online Community books, forums, tutorials	Pros Wide platform access: JSRs Security: MIDP protection Distribution infrastructure using JAR Wide adoption language
Cons Difficult debugging No security solution like java runs in a sandbox Lack of distribution channel (only .swf is the way to run Flash on phones and jar is the only way to run application on the phone less for Flash Lite) memory/cpu consumption	Cons Lack of graphic design tools The UI is not support rich animations, transformation and effects difficult to keep separation between presentation and service layer Designers depend on programmers in UI development

The combined benefits and drawbacks of using both the technologies together according to Thomas Menguy are summarized below. (These items are based upon the presentation in [12].)

Advantages:

- Rapid development due to Adobe's Flash IDE. This makes iterative software prototyping and software testing relatively easy.
- Graphics are vector based (although bitmap support is also included). Vector graphics allows easy scaling, rotation, and other transformations *without* loss of graphic quality.
- Vector graphics also allows the programmer to pack more animation and graphics into the same file size that would be the case when using bitmapped animation and graphics.
- Web-based (desktop) Flash content can be converted to mobile content and vice versa, with minimal effort.
- This approach can take advantage of the many programmers and content producers who have Flash development skills. Their understanding of the IDE and of the scripting language facilitates porting applications from a desktop IDE to the mobile development environment.

Disadvantages:

- As of April 2008, only a minority of phones offer support for Flash Lite. However, this is changing rapidly in North America and Europe with carriers such as Verizon. Moreover Nokia, Sony Ericsson, and LG announced Flash Lite devices in 2006 and 2007. However, the limited availability of Flash Lite capable mobiles means a more limited audience compared with that of Java ME or Symbian platforms.
- Poor handling of sound.

2.6 Different approaches for using Project Capuchin

Project Capuchin can be used in three ways to create innovative content:

1. Pure Flash Lite Content (Full Flash skin and services, without Java ME support)

This is the simplest way of using Project Capuchin. We can create a pure Flash Lite application and encapsulate it in MIDlet suites (creating a .jar file) using Sony Ericsson's packaging tools. In this way, the Flash Lite content will be treated the same way as Java content, thus the same distribution infrastructure and system capabilities can be used as are currently used for Java ME content.

2. Java MIDlet using Project Capuchin as UI presentation layer (Full Flash skin and Java services)

In this approach, Flash Lite is used to handle the whole presentation layer; while Java acts as a service provider that transfers data to the presentation layer.

3. Java MIDlet using Project Capuchin for some UI components (Java and Flash UI, with Java services)

In some cases, it is not feasible to use Flash Lite as the only technology for the presentation layer, for example, in 3D games. Hence another more suitable Java technology, such as Mascot API, JSR 184, or JSR 239, will be needed. However, Project Capuchin can be used for presentation of some UI components such as menus in 3D games.

We used the second of these approach with Flash's UI as our front end and Java ME as the backend. We tried to implement all possible GUI features in Flash Lite. Some features could not be handled using Flash Lite, so we used Java ME's GUI, specifically MIDP 2.0's lcdui. We mostly used Java for data processing and video playing using JSR-135 (which is available in MIDP 2.0 as MMAPI).

2.7 SWF2JAR

SWF2JAR is a tool for automatically embedding a Flash Lite file in a .jar suite. This application can only handle Flash Lite files that do **not** use Project Capuchin's data transfer mechanisms (such as DataRequest or ExtendedEvent). If data transfer mechanism between Flash Lite and Java is used, then SWF2JAR cannot be used because Java code is needed to complete the communication.

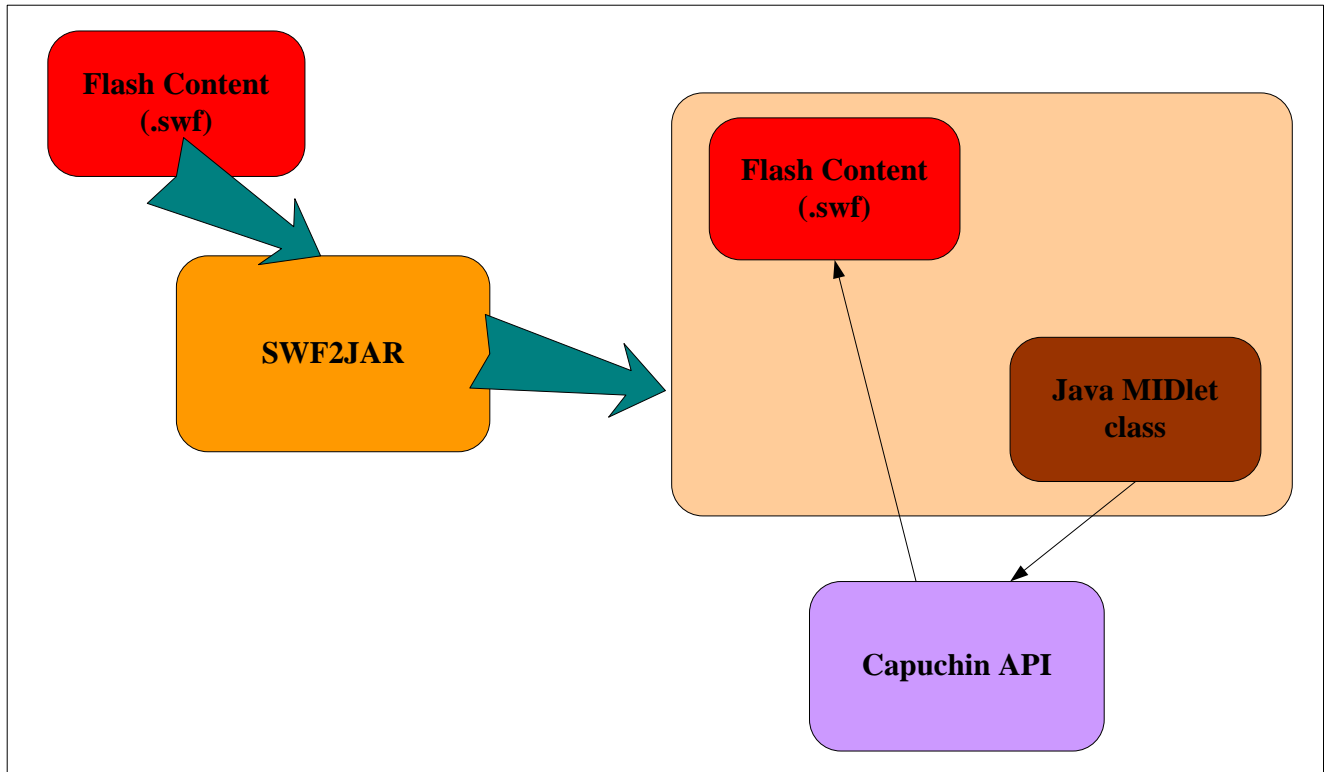


Figure 3: SWF2JAR compiling tool (Adapted from [13])

2.8 Development

There are six fundamental steps when developing Capuchin projects:

1. Applications requirement and UI draft.

Collect the application requirements and plan the UI.

2. Decide data transfer mechanisms.

Decide which of the available data transfer mechanisms to use: DataRequest, ExtendedEvent, or both. If Flash content needs to request information from the device, for example, as a result of user interaction, then the DataRequest mechanism is the most suitable choice. If Flash content should be notified when something happens in the phone, then the ExtendedEvent mechanism should be used.

3. Format of data transfer.

Developers should agree upon the format in which data will be passed. It is currently only possible to pass string and number data types. Agreement on the formats and how these can be identified must be reached between the Java programmer and the Flash designer.

4. Implement and test Flash UI.

The Flash designer/developer implements and simulates the Flash UI separately from Java using loadVariables() Action Script functions to read dummy data from an external file such as a text file.

5. Java implementation.

The Java programmer writes the Java code using the Project Capuchin API and implements the required services, for example, by using services from the relevant JSRs.

6. Test and release the application.

The Java programmer tests the application in the device and releases it.

3 Min TV development

This chapter describes the research approach (methodology) adopted for the implementation of our application – Min TV, in English: “My TV”. After the overview of Min TV, the lab environment and the programming required to realize the two GUI themes are discussed.

3.1 Min TV Communication Overview

Figure 4 shows how Min TV client communicates with the two most important application servers. When Java first receives a request from Flash Lite, it sends an HTTP request to the EPG server to ask for fresh data. An XML file transferred using the OMA BCAST standard [14] will be returned to Java. After Java parses this XML file, it sends back a response containing the information to Flash Lite. If the end-user would like to watch streaming TV or a movie, then a Real-time streaming protocol (RTSP) request is sent from Java to the media server. The media server will send back the appropriate .3gp file (a file format that can be played on mobile devices using a media player).

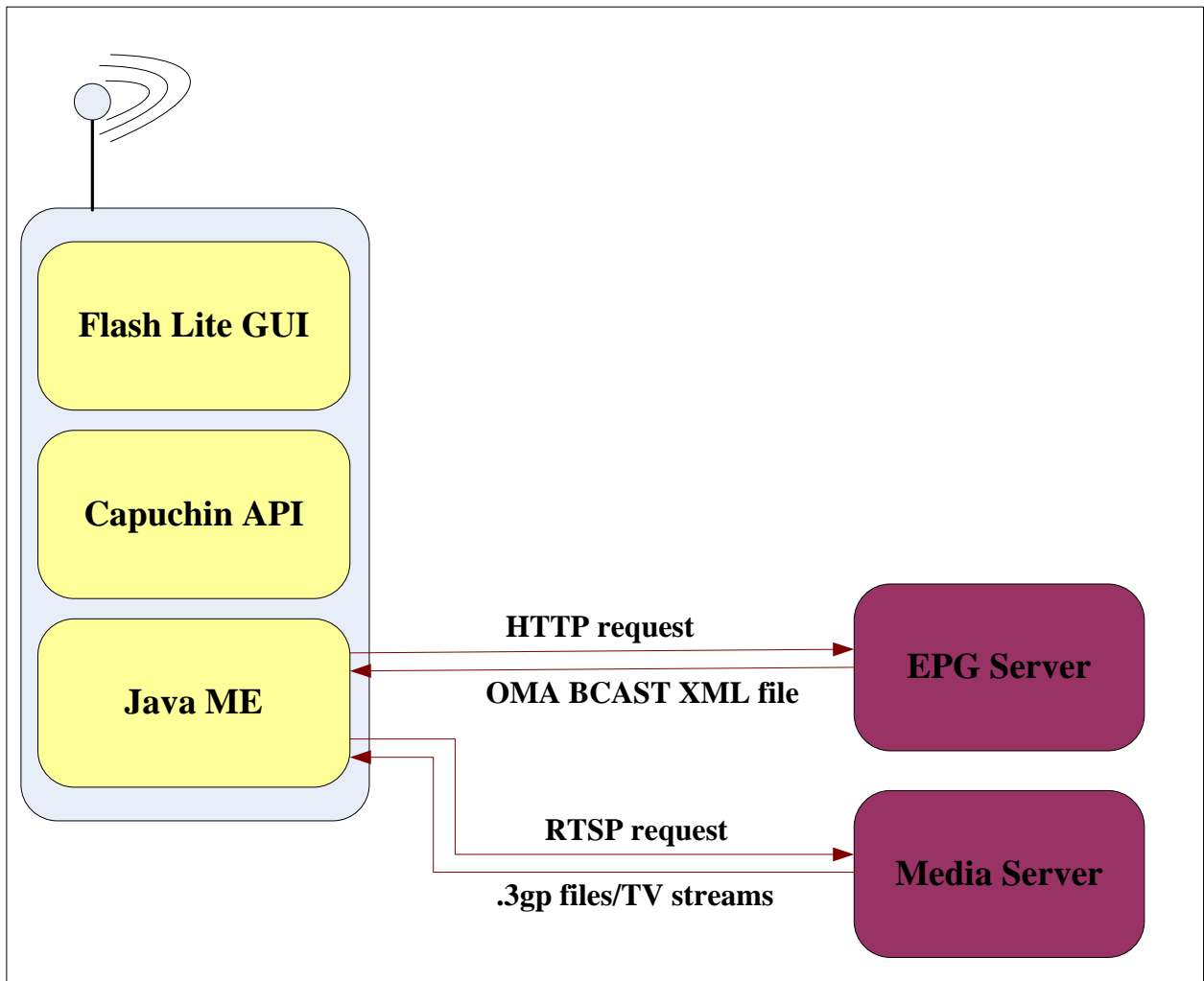


Figure 4: Min TV Communication Overview

3.2 Mobile handset and CTV lab setup

At the time of this research, there were only two handsets available in the market that support Capuchin. By the end of year 2009 there will be total of 6 handsets that support Capuchin. We used a Sony Ericsson C905 handset which is the first model that supports the software development kit (SDK) for the Capuchin API. The C905 mobile phone is an A200 series handset, this means that it has three soft keys (left, right, and middle keys) and has access to the 3G network. This handset is shown in Figure 5.



Figure 5: C905, First Capuchin Enabled handset

Before testing the phone in the lab environment we need to set some properties in order to access the wireless lab network. We used a data account with internet settings named “Raza in lab”. The access point name (APN) was set to “dmmp.g-external”. Similarly streaming settings to access live TV were also set to “Raza in lab”. Figure 6 shows the IP addresses of each of the nodes in our test network.

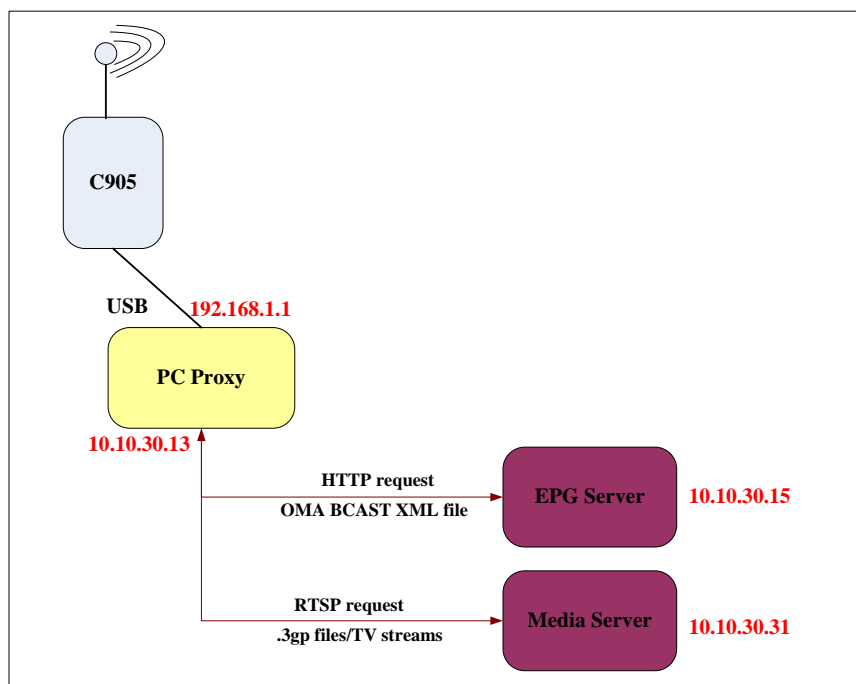


Figure 6: CTV lab network configuration

We fetch the data for the service guide every week from a database server and put it in to the EPG server (running using a Java Application Server). This input consists of two files: an XMLTV file and a Converged TV (CTV) Profile. The CTV Profile is a workaround to define some extra metadata needed when creating the Access and PreviewData fragments. The process is explained in detail in appendix A. This process is shown schematically in Figure 7.

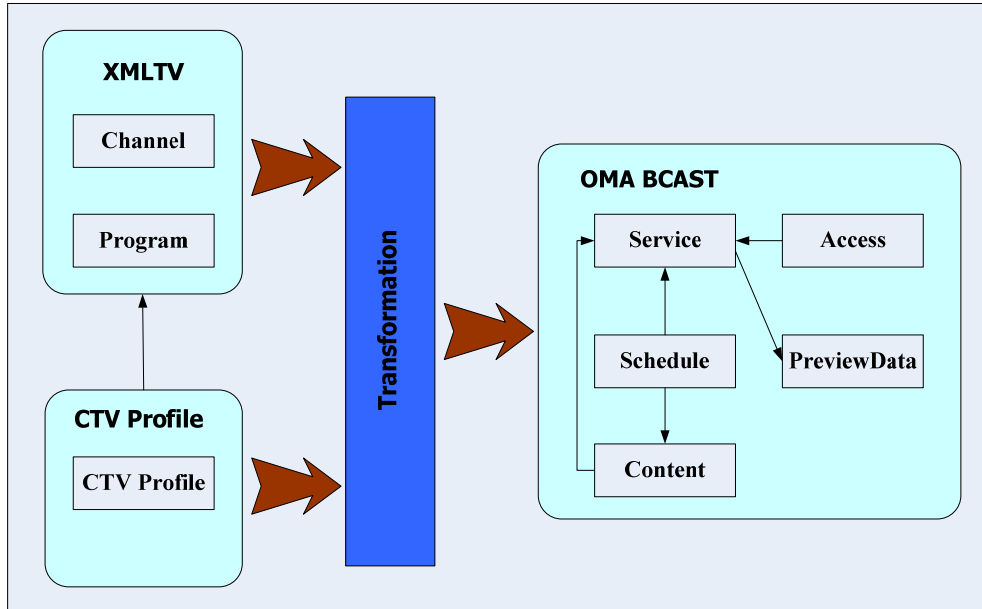


Figure 7: Generating the EPG data

We had to set some preferences in the SDK. In the network configuration of the “Preferences of Emulator” we set the HTTP proxy to be “www-proxy.ericsson.se” with port “8080”. The memory monitor and network monitoring were also enabled. The security was set to “manufacturer” in the “Security Domain” (see Figure 8).

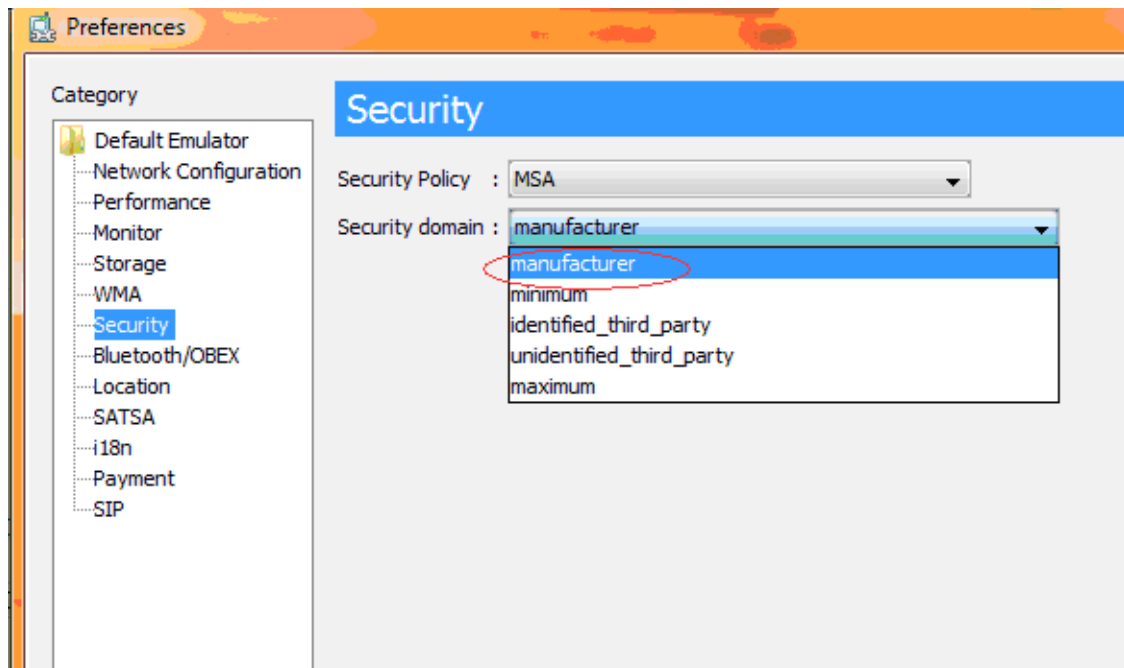


Figure 8: Setting the security level for the emulator

3.3 Setting up java environment for Project Capuchin

The Java environment described in this section consists of:

1. Java Runtime Environment (JRE) 1.6 or later. JRE is available at <http://www.java.com/en/download/manual.jsp> . Download and install it.
2. Sony Ericsson SDK for the Java ME platform. Sony Ericsson SDK for the Java ME Platform, found at Sony Ericsson website at: http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp .
3. Eclipse. Download Eclipse Integrated Development Environment, available for download at: <http://www.eclipse.org/downloads/> .
4. EclipseME plugin. This can be installed by using the EclipseME update site: <http://eclipseme.org/docs/installation.html> .
5. Project Capuchin Classes. The Project Capuchin API is available for download at: <http://developer.sonyericsson.com> .

3.4 Implementation details

3.4.1 Using Capuchin API

Two programming languages are used in parallel (Action Script 2.0 and Java ME). We developed an application in Java ME which does tasks such as XML parsing, playing the video streams after accessing them from media server, implementing the soft keys to move forward and backward in the video, implementing an Alert signal, and implementing the Capuchin code.

The graphics and the screens are created using Flash Lite, while Action Script is used to implement the communication between the Flash Lite and the Java ME using the Capuchin API. Phone control keys such as up, down, left, and right keys are also implemented using ActionScript. You can see it Figure 22.

We made a .jar file of the Capuchin classes which are available from the Capuchin web page on the Sony Ericsson website.

3.4.1.1 The Flash Lite part

The Flash files are created using Adobe Device Central CS3 [15]. In this program we first choose to create a new mobile Flash File, as shown in Figure 9.

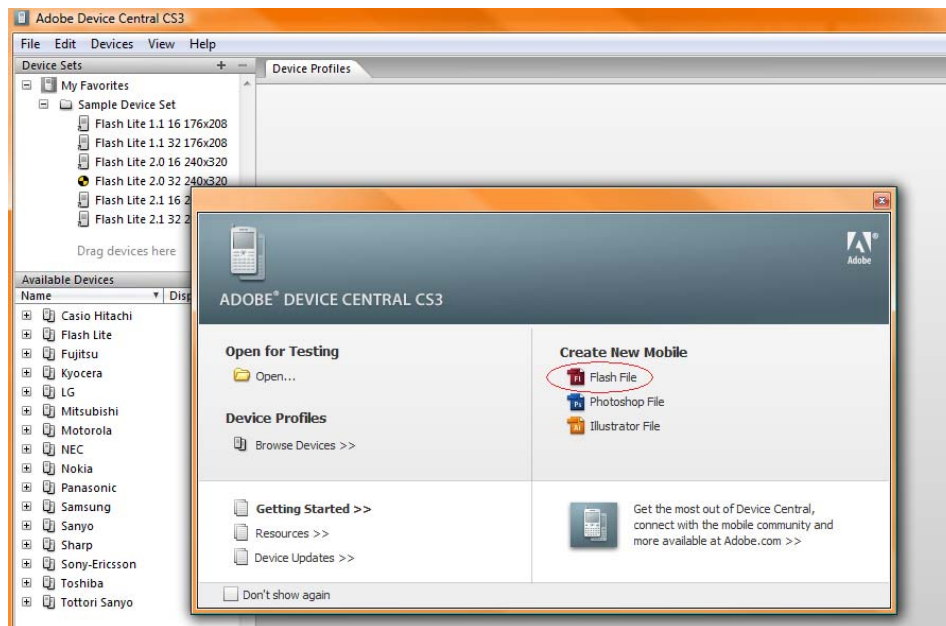


Figure 9: First Window

Next we have to choose different parameters such as: Player version, Action Script version, and the screen size. Our Min TV application requires a screen size of 273*302 (due to lack of full screen capability and soft key support at the same time in Capuchin) instead of the standard 240*320 pixels. This is set by specifying a custom size, as shown in Figure 10.

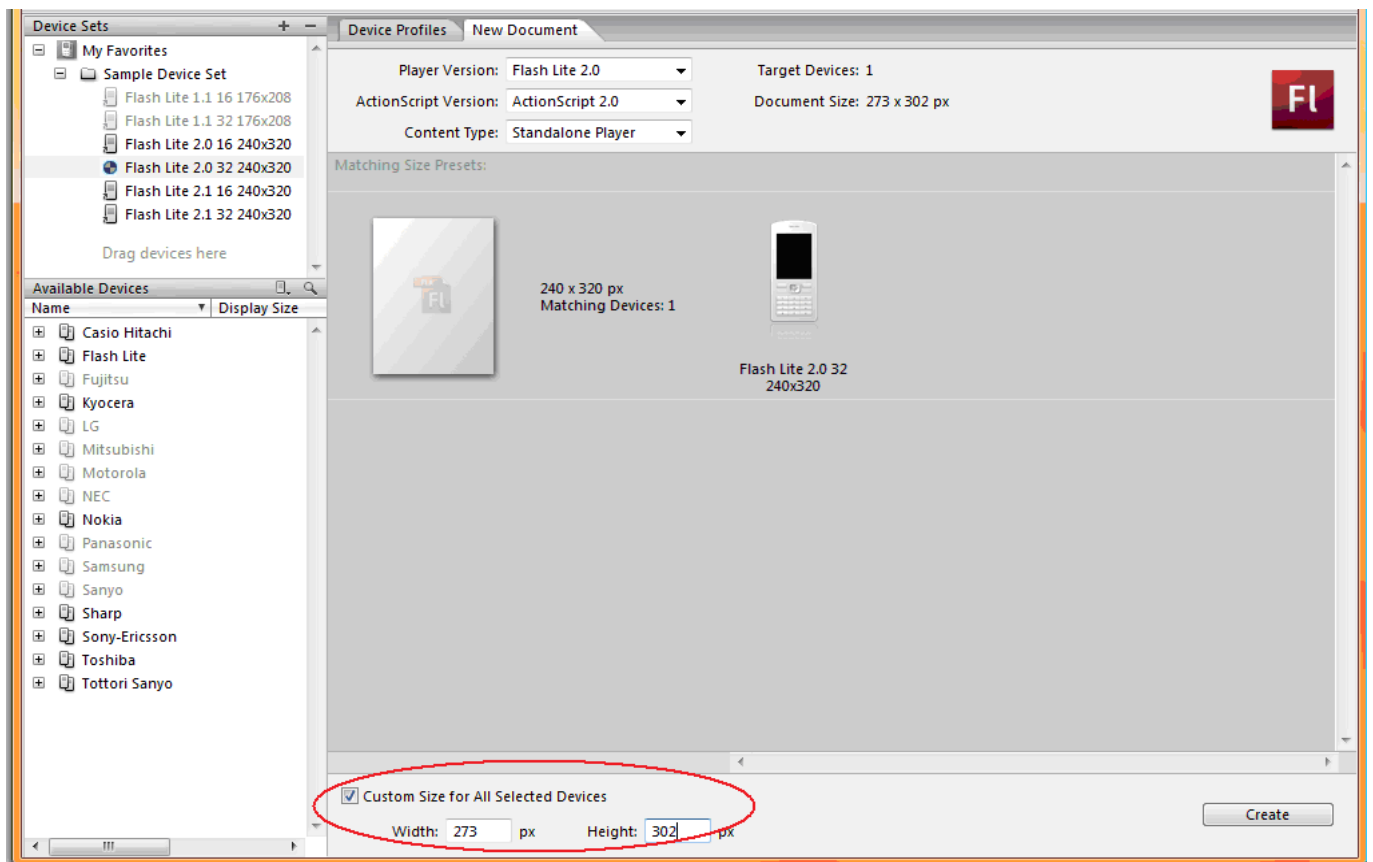


Figure 10: Set Screen Size

After pressing the “Create” button we get the Adobe Device CS3 Professional start up screen (see Figure 11). The interface is like following. This interface displays the timeline, the layers, and the scene mode, etc. Details of how to use Adobe Device Central CS3 can be seen in [16].

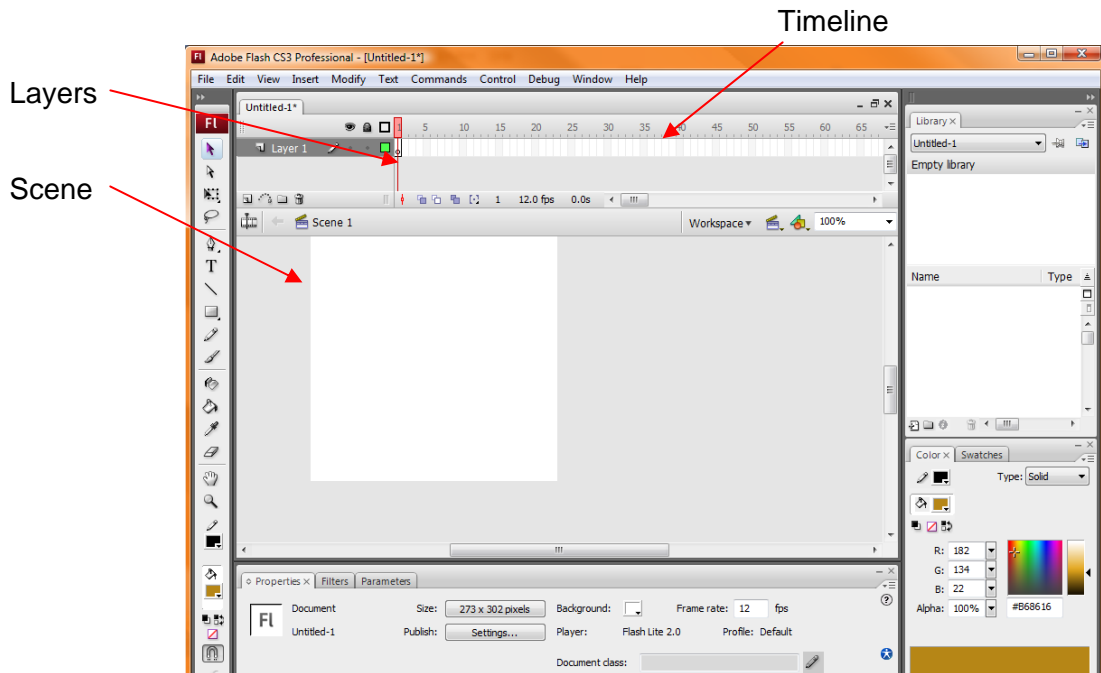


Figure 11: Design Interface

After generating our GUI we output the results as a Flash file. This creates a file with the extension: “.fla”. Next we compile this to produce a file with the extension “.swf” using “Ctrl+Enter” (test movie). A sample of such a compiled .swf file is shown in Figure 12.

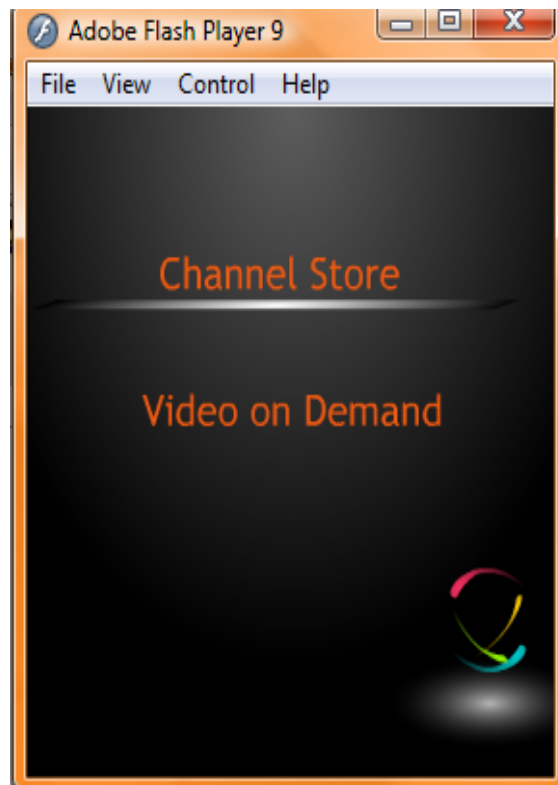


Figure 12: A sample .swf file

3.4.1.2 The java part

The .swf file is exported to the Java ME Capuchin code where it is added as a resource (to a project). Figure 13 shows sample Capuchin code for loading a file “theTwo3ASV.swf”.

```
try {
    InputStream inp = getClass().getResourceAsStream("/themeTwo3ASV.swf");
    flashImage = FlashImage.createImage(inp, null);
    flashCanvas = new FlashCanvas(flashImage);
    flashPlayer = FlashPlayer.createFlashPlayer(flashImage, flashCanvas);
    // flashCanvas.setFullScreenMode(true);
    flashImage.setFlashDataRequestListener(this);

    flashCanvas.addCommand(back);
    flashCanvas.addCommand(next);
    flashCanvas.addCommand(quit1);

    flashCanvas.setCommandListener(this);

    flashImage.setFlashEventManager(this);
}
catch (Exception e)
{
    e.printStackTrace();
}
```

Figure 13: Sample Capuchin code for loading a .swf file

The themeTwo3ASV.swf file is loaded as a resource to the project by using the method `getResourceStream`. `FlashImage` is created to provide an `InputStream` as an argument to the `flashImage` factory method. This `flashImage` is then put on a `flashCanvas`. On the mobile phone screen we have `flashCanvas` and on this canvas it shows the `flashImage` which we get from the `inputStream`.

This is what the constructor looks like. It is the same whether loading this particular .swf file or any other .swf file. If we want to load another .swf file, all we need to do is to change the name of the file in the code and put the file in the resource folder.

However, before loading the .swf file we need to parse the XML and retrieve the relevant data from the EPG and VoD servers. The classes for the parsing come from the southend libraries. Therefore, these libraries need to be added to the Java ME project. An example list of import statements are shown in Figure 14.

```
Min TV.java
2  import java.util.Vector;
3
4  import se.southend.platform.CurrentPlatform;
5  import se.southend.platform.PlatformImpl;
6  import se.southend.platform.IApplication;
7  import se.southend.xml.XMLParser;
8
9  import com.sonyericsson.capuchin.FlashDataRequest;
10 import com.sonyericsson.capuchin.FlashDataRequestListener;
11 import com.sonyericsson.capuchin.FlashEventListener;
12 import com.sonyericsson.capuchin.FlashEventManager;
13 import com.sonyericsson.capuchin.FlashImage;
14 import com.sonyericsson.capuchin.FlashPlayer;
15 import com.sonyericsson.capuchin.FlashCanvas;
16
17 import eef.utils.logger.Logger;
18 import eef.mtv.Config;
19 import eef.core.Store;
20 import eef.mtv.tv.Channellist;
21 import eef.mtv.vod.SimpleVod;
22 import eef.mtv.vod.VodFolder;
23 import eef.mtv.vod.VodContent;
24 import eef.mtv.vod.VodItem;
25 import eef.mtv.broadcast.esg.ServiceGuide;
26 import eef.mtv.broadcast.esg.ServiceGuideData;
27 import eef.mtv.broadcast.esg.ServiceGuideListener;
28 import eef.utils.logger.Appender;
29 import eef.utils.logger.FileAppender;
30 import eef.utils.logger.Logger;
31 import eef.utils.logger.OSDAppender;
32 import eef.mtv.tv.Channellist;
33 import eef.mtv.tv.EpgList;
34
35 import java.io.*;
36 import javax.microedition.media.*;
37 import javax.microedition.media.control.*;
```

For parsing

For Capuchin

For VoD server

For EPG

Figure 14: Min TV import statements in Java ME

In the Java startApp() method when the MIDlet starts we load the EPG and VoD as shown in Figure 15. We begin by creating an instance of the TestMtvEsg class to fetch the ESG – this will give the user information about programs that are available now or in the near future. Then we create an instance of the VodDat class so that we can retrieve the VoD catalog, thus the user will be able to choose any VoD file available from the VoD server. In this test code we wait for 45,000 ms = 45 seconds for the VoD catalog information. This value was chosen to make sure that the VoD data as a whole can be retrieved. Note that the combination of sleeping periods means that we give the EPG server 50,000 ms = 50 seconds to start delivering EPG data. This value was set so that all the EPG data can be retrieved.

```

144 protected void startApp()
145 {
146     test = new TestMtvEsg();
147     voddat=new VodDat();
148
149     Logger.getRootLogger().debug("startApp()");
150     CurrentPlatform.getInstance().setApplication(this);
151     final PlatformImpl platform = new PlatformImpl();
152     CurrentPlatform.getInstance().setPlatform(platform);
153
154     System.out.println("TESTING OMA SG retrieval");
155     //Config.getInstance().setOmaEntryPointUrl("http://192.36.163.40:8181/dist-interaction-channel-war/InteractionChannel/ServiceGu
156     //Config.getInstance().setOmaEntryPointUrl("http://localhost:8080/epg/epg_5_letter_lang_fixed.xml");
157     Config.getInstance().setOmaEntryPointUrl("http://10.10.30.15:8282/dist-interaction-channel-war/InteractionChannel/ServiceGuide/
158
159     test.startFetch();
160
161     try{
162         //Thread.currentThread().sleep(5000);
163         Thread.currentThread().sleep(45000);
164     }
165     catch(Exception e){}
166
167     voddat.setVOD();
168
169     // Allowing enough fresh air to TestMtvEsg class to breath for find the EPG !
170     try{
171         //Thread.currentThread().sleep(20000);
172         Thread.currentThread().sleep(5000);
173     }
174     catch(Exception e){}
175
176     // Getting the EPG data to be displayed on the screen
177     applimplepg=test.getEPG();
178

```

The address of the Electronic Program Guide containing information like name, start & end time, description etc

Figure 15: StartApp() code for MIDlet startup

The Video on Demand catalog information is encoded in XML, thus it can be accessed and parsed as shown in Figure 16.

```
public void setVOD()
{
    System.out.println("TESTING VoD Catalog retrieval");
    HttpURLConnection conn=null;
    InputStream is=null;
    try
    {
        Store.getInstance().put("vod_item_genre","All");
        //conn = (HttpURLConnection)Connector.open("http://localhost:8080/simulator/vod.jsp");
        //conn = (HttpURLConnection)Connector.open("http://192.36.163.40:8181/vod-distributor-war/service?from=15&sublevel=1&lang=en");
        conn = (HttpURLConnection)Connector.open("http://10.10.30.31:8080/iap-mobilevod-war/vodList?sublevel=1&lang=en");
        //String contentLength = conn.getHeaderField("Content-Length");
        is = conn.openInputStream();
        XMLParser parser=new XMLParser();
        parser.parseLowerCase(false);
        parser.open(is, new SimpleVod());
        //get the rootfolder...
        VodFolder vodfolder = Config.getInstance().getVodCatalog();

        printVodFolderInfo(vodfolder);
    }
    catch (Exception e)
    {System.out.println("Exception caught in ApplicationImpl");    e.printStackTrace();}
}

private void printVodFolderInfo(VodFolder vf)
{
    Vector items = vf.getItems();
    System.out.println("Vod Category(Movie/Drama/Comedy/Music) size is "+items.size());
    System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    System.out.println("Vod ITEM NAME IS "+getItemName()+" Yeah");
    System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

    if(getItemName().equals("Comedy"))
    {
        vodcomedycount=items.size();
    }
}
```

The address of the VoD Catalog containing the media information like name, url, description etc

Figure 16: VoD xml accessing and parsing

We play the RTSP streams using the MMAPI as specified in JSR-135 [17]. Sample code showing how this playout can be invoked is shown in Figure 17.

```
public class TvStream extends MIDlet
{
    public Form f=new Form("");
    public Player p;
    public VideoControl vc;
    public TestMtvEsg tester=new TestMtvEsg();

    public void startApp()
    {
    }
    // This function plays the TV stream
    // It is called by ApplicationImpl in DataRequest when "Watch" is pressed in flash
    public void play(String urlToPlay,Display d)
    {
        //String nameOfProgram=tester.getSelectedProgramName();
        String nameOfProgram=TestMtvEsg.forTvStreamProgName;
        Ticker t=new Ticker("Accessing :: "+nameOfProgram+" At URL "+ urlToPlay);
        // play("rtsp://10.10.30.11/live/conv01.sdp");
        try
        {
            f.setTicker(t);
            //good rtsp VoD streams are
            //rtsp://10.10.30.11:554/mtvclip/401_low.3gp
            //rtsp://10.10.30.11:554/mtvclip/401_high.3gp
            p=Manager.createPlayer(urlToPlay);
            p.realize();
            //get the video controll
            vc=(VideoControl) p.getControl("VideoControl");
            if(vc!=null)
            {
                f.append((Item)vc.initDisplayMode(vc.USE_GUI_PRIMITIVE,"javax.microedition.lcdui.Item"));
                p.start();
                f.setTitle("Playing the TV !");
            }
        }
    }
}
```

Figure 17: Playing the video Stream

As we can see in the figure, method play is used to play the TV Steams. It has two arguments: urlToPlay and d. It creates a player and passes the URL to it, then the actual TV stream can be played by the player.

3.4.1.3 Data transfer in Min TV application

The Capuchin uses Flash DataRequests to send requests to Java. Java replies by setting the property parameters. Sample code showing this communication is shown in Figure 18. In this case a DataRequest is created with the string value “vodRequest”. When the onLoad function is triggered, the request will be sent to Java. The Java code check to see if this is the “vodRequest”, if so then it sets the properties of the DataRequest dr. When the Flash code invokes the SayHelloWorld() method it simple access the properties of the DataRequest dataRequestVod.



Figure 18: Data flow between Flash and Java

If Java needs to send an event to Flash it uses `extendedEvents` as shown in Figure 19.



Figure 19: Java event notification to Flash

Once the Capuchin application is compiled using Another Neat Tool (ANT) in Java ME, the file can be sent via Bluetooth or USB connection to the C905 handset as a .jar file. See section 3.4.4.1 on page 26.

3.4.2 Making the Internet available to the mobile via a Laptop PC

3.4.2.1 Settings for C905

The mobile phone which we are using is a member of the Sony Ericsson A200 series, thus it has 3 soft keys and can access a 3G network. However, we can also connect the mobile using a USB cable to a laptop or a desktop PC for development purposes. This requires that we install the device drivers via the installation CD available in the phone’s package.* As a result this PC will be able to see all of the traffic going to and from the mobile handset. This is very useful for development purposes as if the phone were to communicate via a 3G network we would not be easily able to see the traffic that the phone is sending and receiving; this also ensures that we know what competing traffic is being set (as we can easily watch all of the traffic on the fixed networks in our lab environment); additionally if we used a commercial 3G network we might run into problems due to the amount of traffic that we are going to exchange when testing our code.

Secondly we need to set “ON” two properties on the mobile phone. This can be done by going to the “Connectivity” tab in the “Settings” menu, then:

1. Set “ON” the Local connection property in the internet settings.

* Alternatively we can use the PCSuite. This can be found via any search engine by searching for “PCSuite C905 download”.

2. Go to the USB --> USB Mode --> Connect --> from here choose “Via Computer”.

This setting causes the phone to access the internet via USB, rather than over the air.

When using Microsoft’s Windows VISTA, after we have connected the mobile phone we see that there is a local area network (LAN) connection in the network settings for the mobile phone (see Figure 20). If we examine the properties of this connection we will see that it is a 10 Mbps connection. Using the properties of this LAN connection we can manually select the TCP/IP properties and click on the radio box “Set Manual”, then we can manually set the IP address of the phone to “192.168.1.1”. The set subnet mask will automatically be set to 255.255.255.0.

Next go to local Area Connection as depicted in Figure 20 and go to its properties. There are two tabs. The second tab is labeled “Sharing”. Selecting this allows this PC to share the internet with the LAN connected mobile phone by choosing that LAN from the list of available networks. This PC will act as a network address translator (proxy) for the phone.

Details of the features of the Sony Ericsson C905 handset are listed in Appendix B.

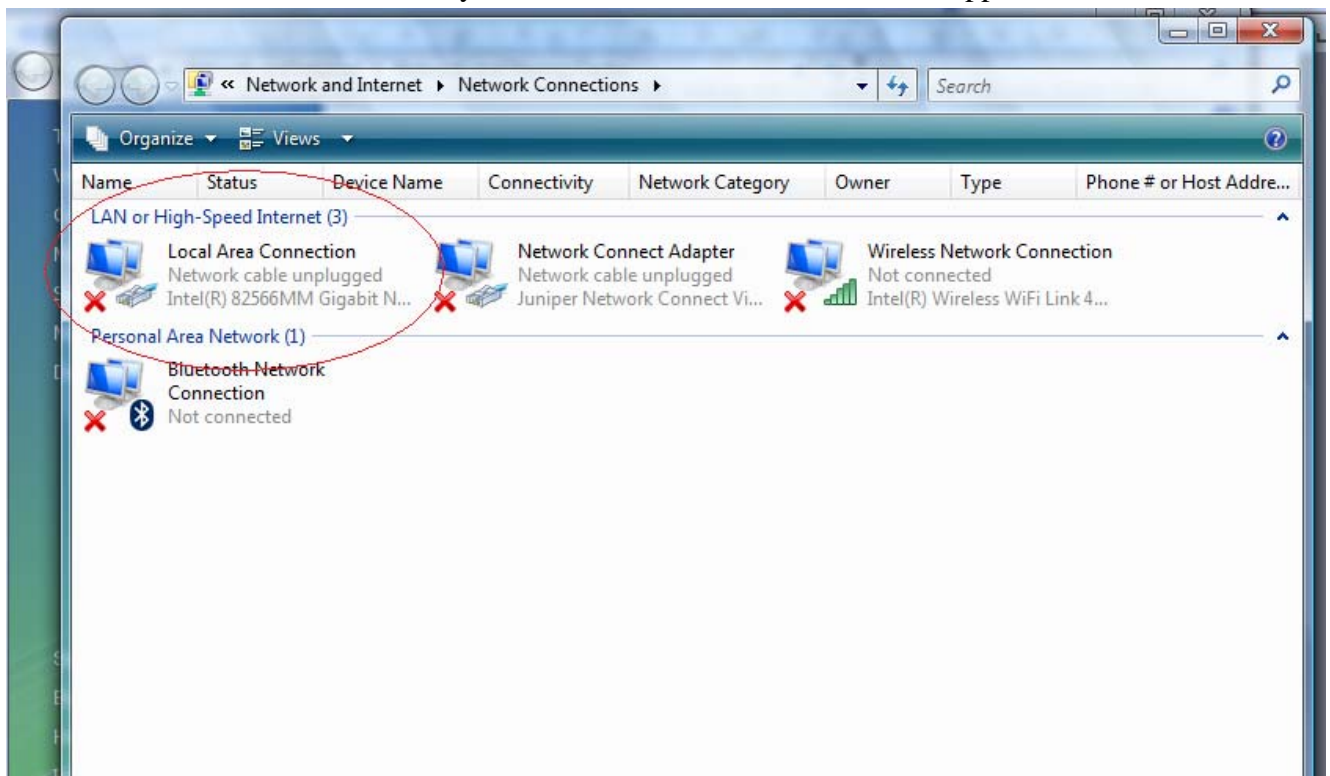


Figure 20: Settings for Windows Vista

3.4.3 Development in Action Script 2.0

When making a Flash Lite file with communication, you need Action Script to support it. We used Action Script 2.0 in several ways. For background about Action Script see [18][19].

3.4.3.1 Pure Action Script files

An Action Script file is similar to the .class file in a Java project. We implemented a “class” in each Action Script file. Subsequently we can call the methods of it at in any frame in the .fla file. We have written a number of Action Script files, specifically: Channel.as, DataProcess.as, DataRequest.as, ExtendedEvents.as, Program.as, retrieveInfo.as, retrieveVodInfo.as, Session.as, vodProgram.as, XDList.as, etc. A brief summary of these is given in Table 2. The source code of these files can be seen in Appendix C.

Table 2: Action Script files

ExtendedEvents.as	This script defines a listener object in Flash and registers this with an event in Java. When the event is triggered in Java, Flash is notified and the necessary data is passed from Java to Flash. This file is a class from Capuchin API. Although we are not able to see the details of how it is implemented, cell phones that support Capuchin API can execute this file.
retrieveInfo.as	This file is used to process the large string received from Java containing all the channel information, so that you can extract the information you need (such as: channel name, program name, etc).
Channel.as	This file is used to get the channel name and the programs of this specific channel.
Program.as	This script extracts detailed information about each program in a channel, such as: program name, start time, end time, description, etc.
XDList.as	This class makes a list and shows the channels and programs in a scrolling way. It tells how many items can be shown on one page, how to scroll down the page, and how to know which program is selected.
DataRequest.as	This is another class Capuchin API. DataRequest is used for asynchronous communication when Flash content wants to request data from Java. It is implemented in the phones supporting Capuchin.
Session.as	This script saves session information so that you can access the data in different MovieClips.
retrieveVodInfo.as	This script is similar to retrieveInfo.as, but it is used to retrieve VoD information.
vodProgram.as	This script is similar to Program.as, but it is used to extract the resources concerning VoD.

3.4.3.2 Action Script in frames

We need to call classes to perform various functions. The interaction between the script (a .as file) and Flash (as written in a .fla file) is encoded as an Action Script in a frame. An example of who to specific an action for a frame is shown in Figure 21.

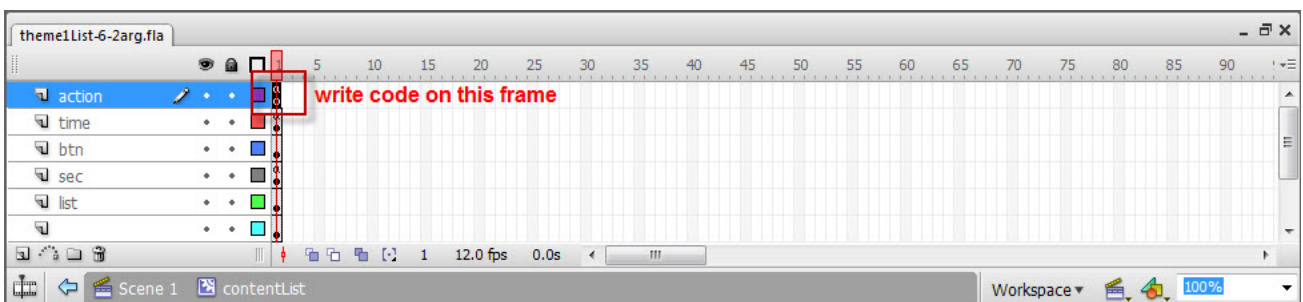


Figure 21: Coding for a Specific Frame

For example, if we want to write codes to control the up, down, left, or right keys, we need to create an instance of an Object and add event listeners for onKeyDown. An example of this code is seen in Figure 22.

```

////////////////////////////////// KEYCATCHER
var keyListenerChn:Object = new Object();
keyListenerChn.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.UP :
            xdlistPro.prevItem();
            break;
        case Key.DOWN :
            xdlistPro.nextItem();
            break;
        case Key.RIGHT :
            Session.setSelectedTxt();
            _root.gotoAndPlay("programmes");
            break;
    }
};
Key.addListener(keyListenerChn);

```

Figure 22: Code for onKeyDown

Alternatively we can add code to process DataRequest and ExtendedEvents, as shown in Figure 23.

```

//////////////////////////////////Data Request for EPG data//////////////////////////////////
var myDataRequest = new DataRequest("sendEPG");
myDataRequest.request();

_global.EPGdata="";

//////////////////////////////////Extended Events to get the EPG data//////////////////////////////////
var myEventListener:Object = new Object();
ExtendedEvents.CapuchinEvent.addListener(myEventListener);

myEventListener.onEvent = function (SomeText,wholeString)
{
    if (SomeText == "EPG")
    {
        _global.EPGdata = wholeString;
        var retri:retrieveInfo = new retrieveInfo();
        var menuItems:Array = new Array();
        var strProgram:Array = new Array();
        var str:Array =retri.getChannels(_global.EPGdata);
        for (var i=0; i<str.length;i++)
        {
            var temp:String=str[i].getChannelName();
            menuItems[i]={itemName:temp};

            var temp:Array=str[i].getProgramList();

```

Figure 23: Code for DataRequest and ExtendedEvents

If the DataRequest mechanism is used, then a DataRequest object is created in ActionScript. The argument passed in the constructor is preserved for Java later (this method can be used to pass the DataRequest identifier from Flash to Java). The DataRequest object has a function called request () which is used to send a request to Java.

An Object is defined in ActionScript. This object is later registered as a FlashEventListener. The listener myEventListener has a function handler called onEvent, this function is called by Java when the event is triggered.

3.4.3.3 Setting the Property

When writing the Flash Lite Gui we can name elements and write code to be attached to these elements in order to show something. For example, in Figure 24, the selected element has a variable name of `currentTime`. The code for this frame is shown in Figure 25.

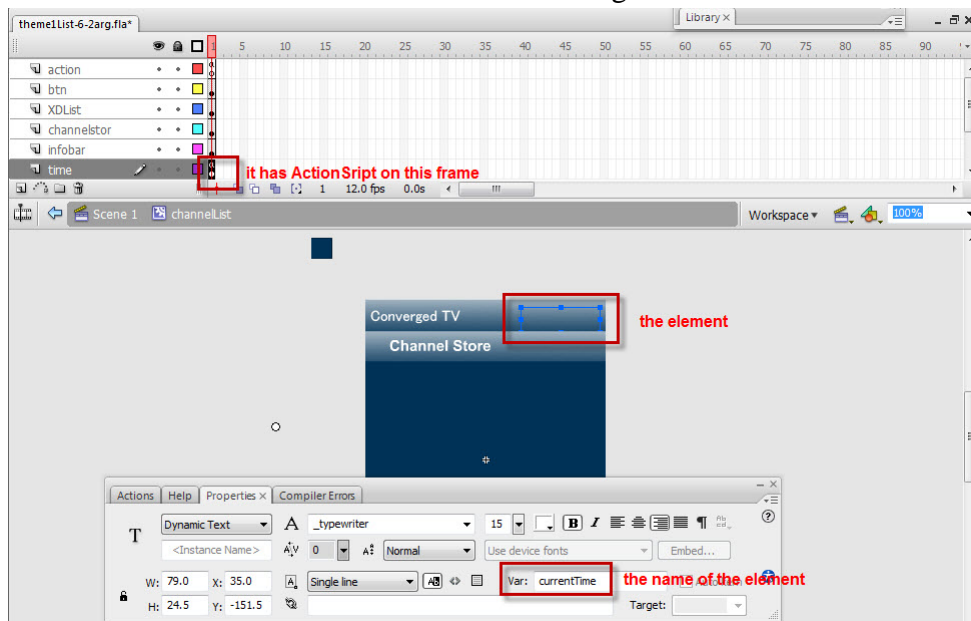


Figure 24: Variable named `currentTime`

```

var today = new Date();
var minutes = today.getMinutes();
var hours = today.getHours();

if (minutes < 10) {
    is_zero_min = "0";
}
else {
    is_zero_min = "";
}

currentTime = hours + ":" + is_zero_min + minutes;

```

Figure 25: Code for the frame shown above

If later you test the movie, you can see the time in the upper right corner, as shown in Figure 26

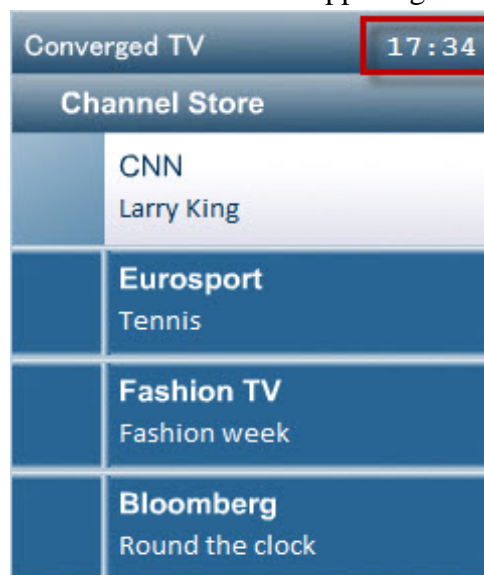


Figure 26: Time displayed in the upper right corner

3.4.4 Compiling the application using Another Neat Tool (ANT)

Apache Another Neat Tool (ANT) [20] is considered by many the Holy Grail of build tools in the Java™ development world. Most Java projects have some sort of custom build process attached to them in the form of an ANT build script. Keeping with this tradition, our Min TV project is compiled and managed by using ANT as a compiling tool within the Eclipse Integrated Development Environment (IDE) [21].

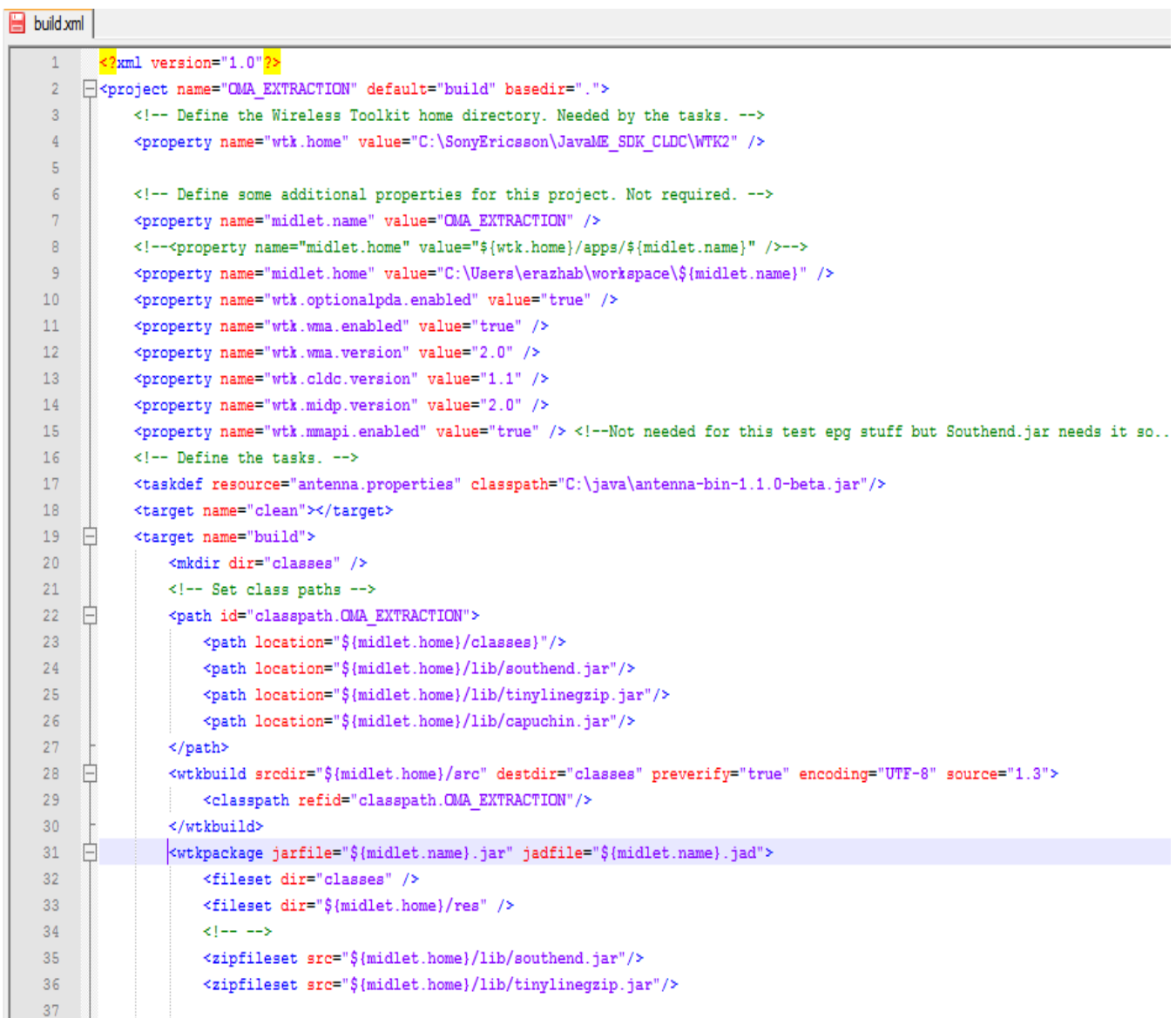
3.4.4.1 Building using ANT

After ANT has been successfully installed, go to command prompt and write “ant”. If it shows:

Buildfile: build.xml does not exist!

Build failed

This means that the ANT is installed correctly. Now write a build.xml file and place it in the same folder as the main code. Build.xml is a configuration file where you define the process of compiling, building and deploying. The Min TV application build.xml file is shown in Figure 27.



```
1 <?xml version="1.0"?>
2 <project name="OMA_EXTRACTION" default="build" basedir=".">
3   <!-- Define the Wireless Toolkit home directory. Needed by the tasks. -->
4   <property name="wtk.home" value="C:\SonyEricsson\JavaME_SDK_CLDC\WTK2" />
5
6   <!-- Define some additional properties for this project. Not required. -->
7   <property name="midlet.name" value="OMA_EXTRACTION" />
8   <!--<property name="midlet.home" value="${wtk.home}/apps/${midlet.name}" />-->
9   <property name="midlet.home" value="C:\Users\erazhab\workspace\${midlet.name}" />
10  <property name="wtk.optionalpda.enabled" value="true" />
11  <property name="wtk.wma.enabled" value="true" />
12  <property name="wtk.wma.version" value="2.0" />
13  <property name="wtk.cldc.version" value="1.1" />
14  <property name="wtk.midp.version" value="2.0" />
15  <property name="wtk.mmapi.enabled" value="true" /> <!--Not needed for this test epg stuff but Southend.jar needs it so..
16  <!-- Define the tasks. -->
17  <taskdef resource="antenna.properties" classpath="C:\java\antenna-bin-1.1.0-beta.jar"/>
18  <target name="clean"></target>
19  <target name="build">
20    <mkdir dir="classes" />
21    <!-- Set class paths -->
22    <path id="classpath.OMA_EXTRACTION">
23      <path location="${midlet.home}/classes"/>
24      <path location="${midlet.home}/lib/southend.jar"/>
25      <path location="${midlet.home}/lib/tinylinegzip.jar"/>
26      <path location="${midlet.home}/lib/capuchin.jar"/>
27    </path>
28    <wtkbuild srcdir="${midlet.home}/src" destdir="classes" preverify="true" encoding="UTF-8" source="1.3">
29      <classpath refid="classpath.OMA_EXTRACTION"/>
30    </wtkbuild>
31    <wtkpackage jarfile="${midlet.name}.jar" jadfile="${midlet.name}.jad">
32      <fileset dir="classes" />
33      <fileset dir="${midlet.home}/res" />
34      <!-- -->
35      <zipfileset src="${midlet.home}/lib/southend.jar"/>
36      <zipfileset src="${midlet.home}/lib/tinylinegzip.jar"/>
37  </target>
</project>
```

Figure 27: Build.xml for Min TV

The first line shows the document type declaration which is xml version 1.0. The second line is the project tag. Each buildfile includes one project tag and all the instructions are written in it. The project is named OMA_EXTRACTION. The rest defines some properties, tasks, classpaths, and Wireless Toolkit home directory, etc.

3.4.5 Debugging and Testing the Application

Testing is only possible on Capuchin capable devices such as the Sony Ericsson C905 Cyber-shot handset. Unfortunately, the Sony Ericsson emulator cannot (yet) read a Capuchin .jar file. The reason is that the C905 handset is under the Java Platform 8 sub category (JP 8.4), for which the device skin is still to be released (although sub category JP 8.0 is available). Note that the C905 is the first JP 8.4 handset [22].

3.4.5.1 Testing Tools

We used two debugging tools: Ericsson’s DebugMux and Sony Ericsson’s DeviceExplorer. We will describe them separately in the following sections.

3.4.5.1.1 DebugMux

Ericsson’s DebugMux is a logger. It shows the phone’s outputs, e.g. the status of the heap, stacks and what file the phone is currently trying to access. We used it mostly to see what RTSP stream it is accessing at the backend from the VOD Server.

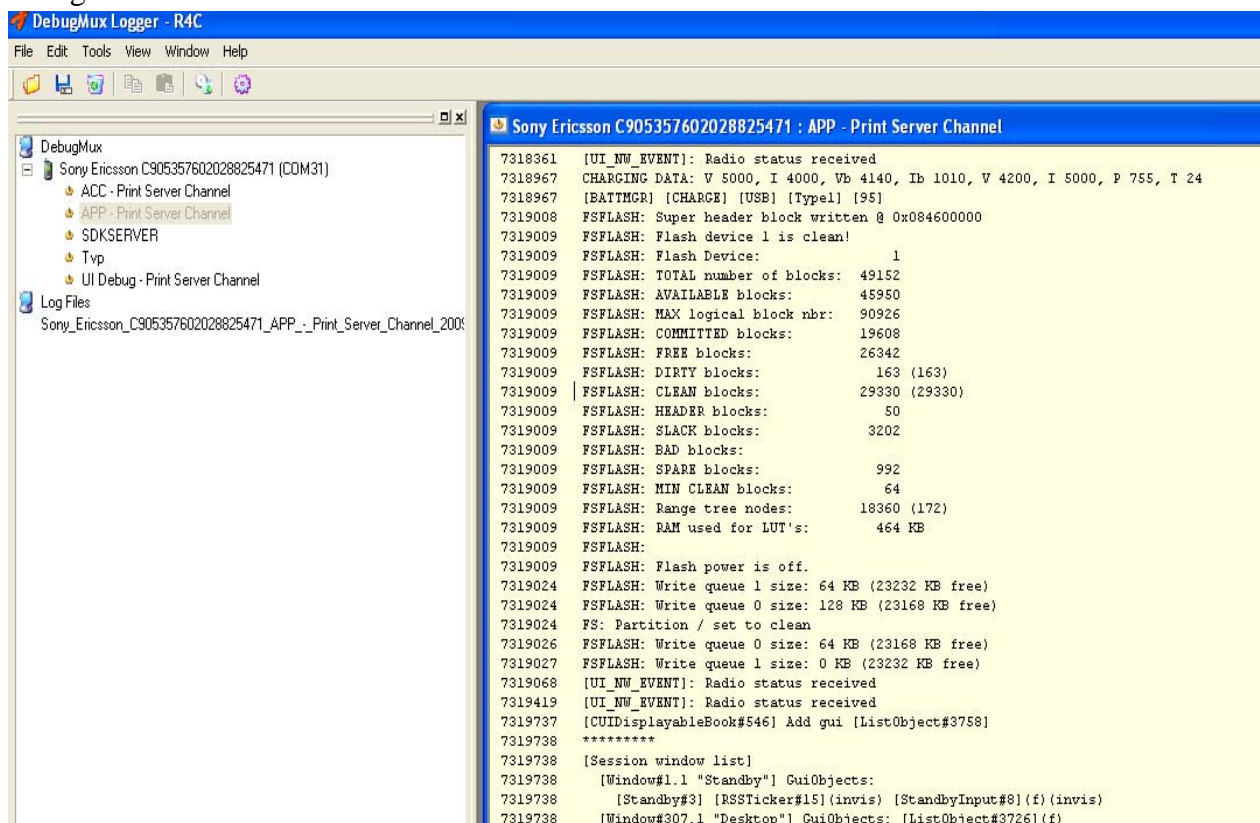


Figure 28: DebugMux while debugging using C905 DeviceExplorer

3.4.5.1.2 Sony Ericsson's DeviceExplorer

To use it we need to have a connection proxy up and running with the phone. How to set this proxy up is described below and in section 3.4.2.1 on page 21. The Connection Proxy is the "glue" between the laptop and the Sony Ericsson mobile phone. It communicates with the phone using the Ethernet driver installed as a device driver. When the communication between the proxy and handset is functioning properly the connection proxy shows the visual identification of the connected handset (in our case for C905) as shown in Figure 29.

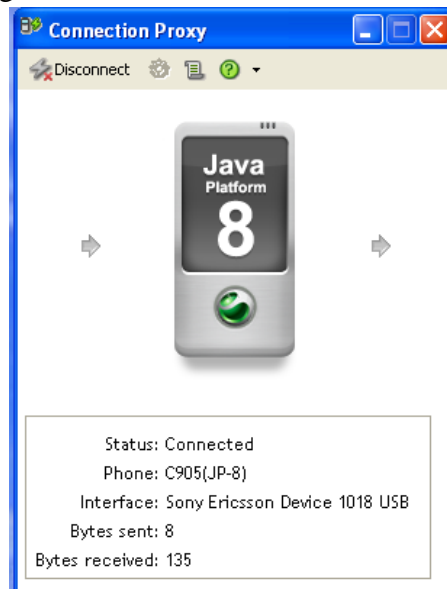


Figure 29: Connection proxy connecting C905

Before making the connection to the proxy, we need to assign a static or dynamic IP (depending on the situation) to the handset. This IP address can be configured as shown in Figure 30.

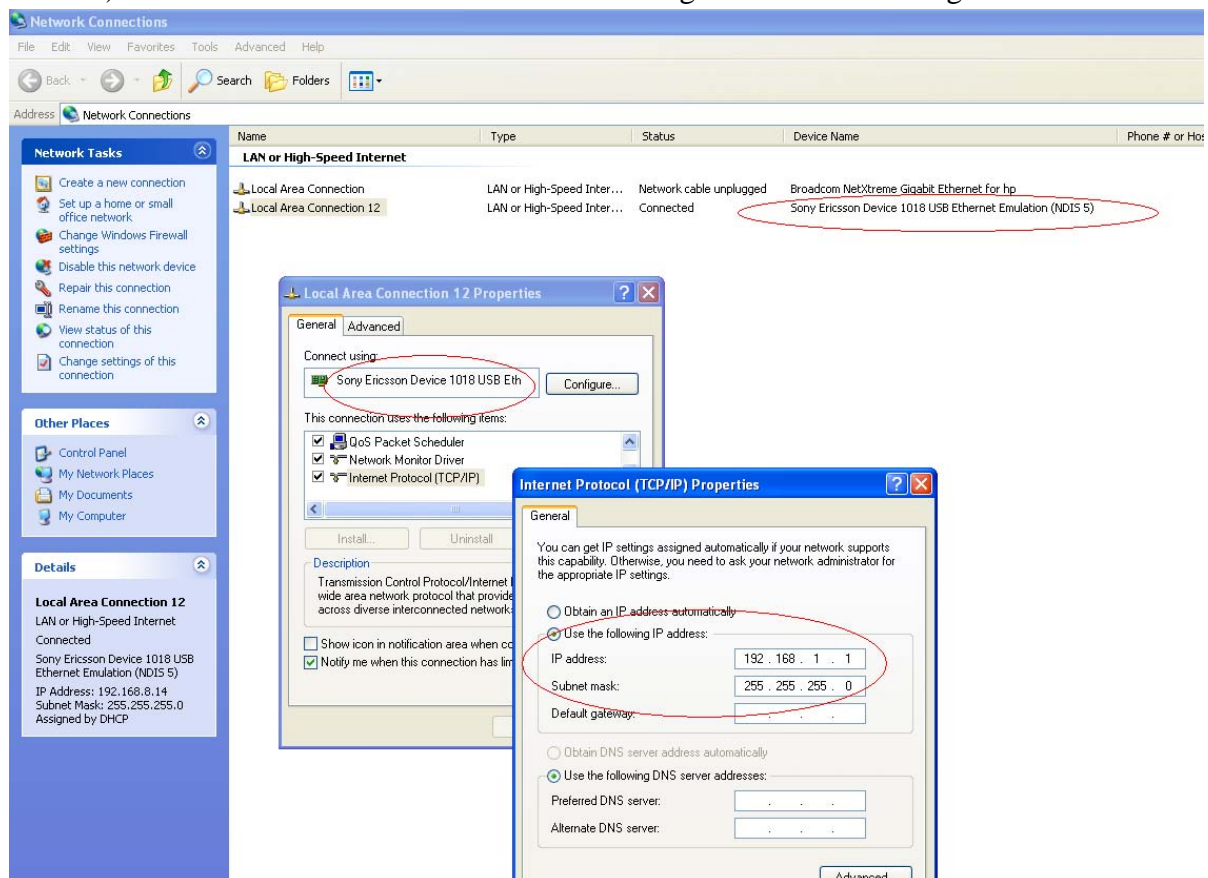


Figure 30: Setting the static IP address to the handset

We used Microsoft's Windows XP in the CTV lab environment to connect to Ericsson's Device Explorer [23], as our laptop is running Microsoft's Windows Vista which does **not** allow us to connect to Device Explorer. Device Explorer gives a three-pane window that provisions and controls the MIDlets in an easy-to-use way (see Figure 31). The Device Explorer view pane (labeled "Sony Ericsson MIDlet Favorites" in the figure) shows MIDlets that are currently installed on this C905, the File Explorer pane (labeled "Sony Ericsson Device Explorer" in the figure) shows the file structure of the laptop to which the C905 is connected, and the console pane (labeled "Console" in the figure) provides the information we request from the phone, including garbage collection and memory utilization information.

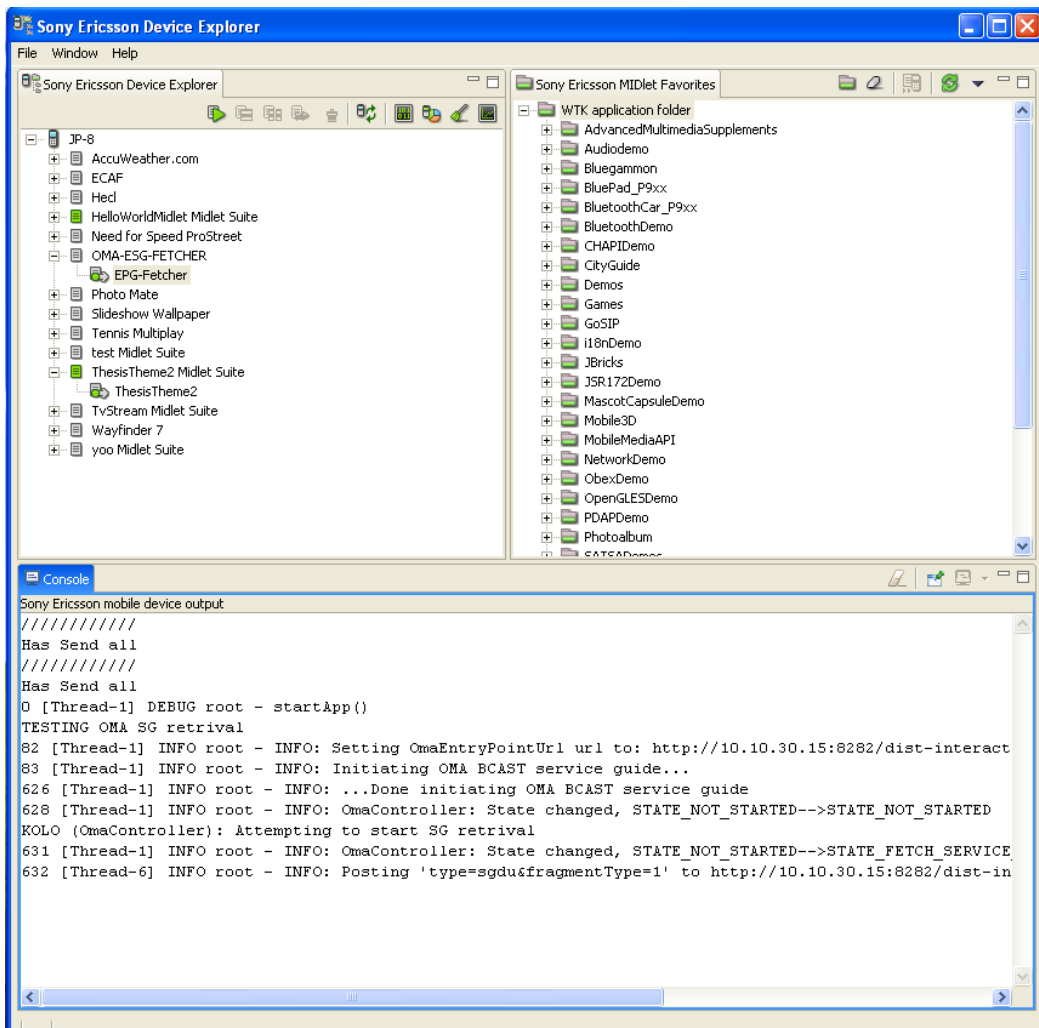


Figure 31: Ericsson's Device Explorer showing output for the EPG fetcher MIDlet from C905

One key feature of Ericsson' Device Explore is the "drag and drop" of jar/jad files from a File Explorer pane (the usual Windows based view of a directory and its files) to the Device Explorer pane of Ericsson's Device Explore. This enables the programmer to easy installation MIDlets on the phone. Additionally, Device Explorer provides button controls to easily start, pause, resume, and stop a MIDlet executing on the phone.

The only difference between DebugMux logger and Device Explorer is that the first one only shows the printout commands using `System.out.println(" ")` for research and Development phones. We need to enable the java mode before using it. On the other hand Device Explorer can run with any phone which is java enabled through connection proxy by connecting with the Ethernet driver of that phone.

3.4.5.2 Testing the streams

VideoLAN's VLC player [24] was used to test the streams to see if the media server was up and running. This player can be invoked as shown in Figure 32.

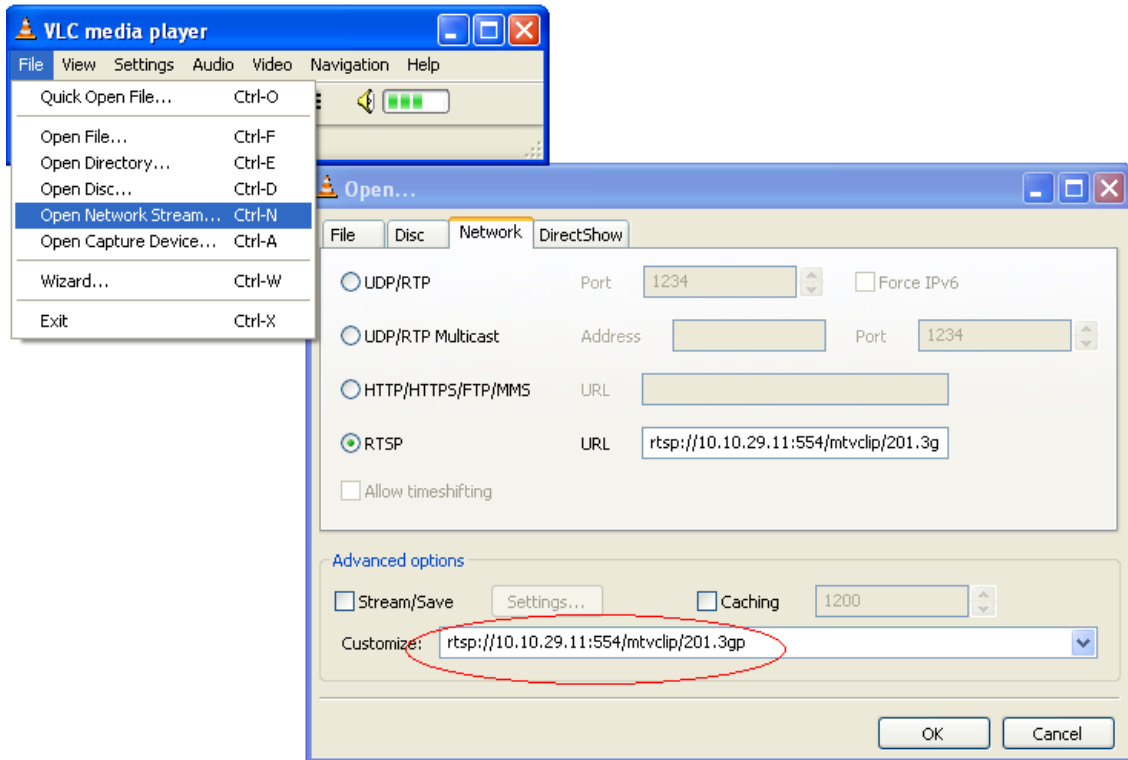


Figure 32: Running a VoD resource in VideoLAN's VLC player

This IP address shown in the figure is the real address of the stream on media server. This must be tested in the CTV lab while connecting to the internal network.

4 Findings and Discussions

This chapter describes the results of testing designing, implementing, and testing a mobile TV application using the Capuchin API in our testbed.

4.1 Evaluation of Capuchin API

First we consider the Capuchin API itself.

4.1.1 Advantages

The Capuchin API offers a number of advantages over writing a pure Flash Lite application. However, there are some limitations that have to be considered.

- The Capuchin API effectively bridges Flash Lite and Java ME, as expected. This enables the developer to easily create a GUI using Flash Lite and implement the semantics of the application using Java ME.
- Although the Capuchin API is in an early stage of maturity, the documentation for the APIs, the installation guide, and the examples were all well written and easy to understand[†].
- The Capuchin API is a more efficient way of processing and rendering data than using Flash Lite alone. In terms of performance, Project Capuchin renders Flash Lite content in the same way as a phone render Flash Lite content in the native browser. However, using Java for calculations and more extensive processing improves the performance compared to native Flash Lite applications. This optimization can be achieved while keeping the amount of data and the number of DataRequest and ExtendedEvent constructions to the minimum. For example, if Java has to update 100 notes, a DataRequest for each note should not be used as this would create 100 DataRequest objects in Flash Lite and would consume a lot of memory. Unfortunately, creating a single DataRequest with 100 notes is also **ineffective**, since it leads to a large overhead if the notes are not going to be presented at the same time. Because Java performs calculations more efficiently than Flash Lite objects creation and extensive computations should be done in Java as much as possible. The optimal number of notes to pass in a single request is not yet known.

4.1.2 Disadvantages

While Capuchin API offers a number of advantages, there are also some disadvantages of using it.

- Currently the Capuchin API only allows String and Integer as transfer methods. Other important data types for data transfers between Flash and Java such as vectors, doubles, and floats are not supported. These data types are needed in order to transfer extensive data to/from Java ME. Note that translating all of the values to and from strings is a very inefficient way to transfer data on a single platform.
- The current Sony Ericsson wireless toolkit emulator does **not** have an SDK for Capuchin. Thus testing can only be done using mobile handsets which are Capuchin enabled. This creates trouble as for testing the .jar file it needs to be sent to the phone in order to execute it after compiling and building it in ANT.
- The Capuchin API which Sony Ericsson has released is implementation less. This means it is actually a stub. A stub is a client side representation of a remote object that is used to invoke methods on the implementation of remote object. See for example details of remote

[†] I wish to give our thanks to the Sony Ericsson developers for providing this ease of use. It was greatly appreciated.

procedure calls as defined in RPC [25]. You need to know what functions are available and in which scenario can be used. This means that if we are using Eclipse as an IDE, then adding the Capuchin Stub will help building the code, but will give null values if we try to run the code (unless there is a server application that implements the semantics of each of the calls).

- Unfortunately, currently the soft keys are **not** supported. This means we cannot write code in our Action Script to handle these keys. If a developer needs to have the keys such as “Back”, “Forward”, “Exit”, and “Play” there are two options:
 1. Handle the soft keys completely in Java ME. The developer will use command Listeners and command Actions, then execute the appropriate .swf file for playing or traversing back. This means that there will be not be large swf scene in the code -- but rather a lot of small .swf scenes.
 2. Combing Java ME and Action Script. After handling the command in Java ME it will be sent to Action Script using Capuchin classes, then the Action Script will handle the command by playing the file or traversing backward or forwardward.
- If we use the soft keys in Java ME, then the relevant .swf files need to be loaded as a resource which makes the application consume more resource. This approach will also create a lot of redundancy in the Java ME code for loading the swf file, as the same code will be repeated again and again with the different .swf files. This redundancy also occurs in the action listener code and the code that attaches the commands to the different Flash canvas. Another approach is to have all the movie clips linked together using a single action script with one scene. In this approach Capuchin’s extendedEvent can be used to send the Java ME commands back to Flash where they are listened for and handled. However, there are two drawbacks to this approach. First is that if Action Script needs to send data back to the Java ME after it has listened to the event it is notified about, then it has to use Capuchin’s dataRequest. This method involves signaling between the extendedEvent and dataRequest implementation in Java ME by using flags to notify the two parts of code. A second drawback is that all the screens must have the same name for the soft keys. This means that we cannot change the name of soft keys by dynamic binding, i.e., they can not depend on the Flash screen. So either VoD or the EPG Guide will have same names for the soft keys. Unfortunately, this means that sometimes the “Next” key will act like a “forward” button and it will take the user to the next Flash screen, but sometimes it will act like a “Play” button an start the playing of the actual stream. This would mean that the user has to guess what the “Next” key means in each particular Flash screen’s context.
- When using the dataRequested transfer method, from Java ME to Flash, as we can only send the data in two formats (i.e. as an integer or as a string), these transfers will be insufficient. If we want to send a long String with 1300 characters there will be a buffer overflow error. Unfortunately, sometimes we need a long string in order to send Flash all the channel names, the respective program names, the start and end times, and the description of the programs so that the Action Script can split this string into the relevant parts and display it appropriately. An alternative solution is to split this string in Java, then send parts of this information to Flash (for instance substring by substring). This will require a lot of dr.setProperty() notes for each of above mentioned attributes and a loop. Moreover this will **not** work if we have dynamic data of unknown length, as we can not know the number of property attributes that would be needed. Another alternative is to use the FlashEventManager interface in Capuchin API together with the CommandListener interface from Java ME’s LCDUI, however - whenever the screen is changed the string data is posted to Flash Lite creating a lot of redundancy.
- Currently we cannot debug Flash Lite using DeviceExplorer. We can only see the console sent by Java ME. This makes it nearly impossible to debug Flash Lite problems. On the 3rd

- We tried the option of loading multiple Flash files as a resource in Java ME in order to solve the problem of having different soft key names on different canvases. While we managed to do this in theory, a practical problem remains as every screen in Flash canvases is loaded as an independent Flash file, i.e. on top of one another. Two possible reasons are:
 1. Data transfer using ExtendedEvents is not currently working. The data cannot be transferred from Java to Flash from one Flash canvas to another Flash canvas. This means that although Java is sending an extendedEvent as a result of a dataRequest from Flash, if the listener is on another screen it does not work.
 2. Screen switching is a mess. When we switch from Screen₁ to Screen₂, we are able go back to Screen₁. But the problem is now the soft keys do not work, hence no matter how many times we press the “Select” or “Back” key again, it stays on the same screen and does not switch again.
- In full screen mode, the soft keys do not show up on the Flash canvas. We think there are two reasons for this. First is that since Capuchin does not support soft keys it has no mechanism to handle the keys in full screen mode. The other possibility is that the soft key commands are attached to the screen via Java ME, thus since the Flash canvas is loaded as a resource file *on top* of LCDUI screen, when in full screen mode the soft keys remain on the second layer while canvas shows up on the top layer. In order to see the underlying screen we want to see, we have to click twice on the Flash canvas.
- If the compiled .swf Flash Lite file is larger than 5MB, it cannot be played on the mobile. Thus while we can include animations we can not include too many animations. Our simple application with a movie clip with splashing light effects and rotation was about 8.65 MB – thus this could not be played.

4.1.3 Conclusion

No full screen mode, no support for soft keys, poorly developed debugging software, inefficient data transfer methods (using data Request and a reply from Java ME using setProperty()), make the current environment unsuitable for developing a mobile TV application. Additionally, because the Capuchin development API is a stub and there is no support for the Java ME developer to see the working of the basic API and there is no SDK emulator we need to use an actual phone to test Capuchin applications; this reduces the speed of developing code.

In summary, although Capuchin is a good technology which merges Flash Lite and Java ME, it is currently very difficult to develop professional applications in which Java ME has to continuously interact with Flash Lite. However, applications that have very little interaction are relatively easy to develop.

4.2 Capuchin versus ECAF

In this section we will compare the Capuchin and ECAF technologies in terms of parameters such as: memory size, lines of code, processing time, etc.

4.2.1 Animation capabilities

Theoretically, Flash is far superior for creating animations compared to any other currently available software. However, we will compare it with ECAF. Some interesting ECAF animations were compared with their Flash Lite alternative and we observed that there is a clear relationship between ECAF tags and Flash Lite (see Table 3). Note that the graphic concept of “tweening” (a shortened version of “inbetweening”) involves automatically generating the frames in between to key frames – without needing to explicitly define each of the in between frames.

Table 3: Relation between ECAF tags and Flash Lite

ECAF tags	Flash Lite
<move>	Motion Tween in a defined path
<frame >	Frame by frame motion
<rollover>	In Action Script using “onRollover” event
<transparency>	Transparency of image
<wave>	Wave surfing like effect
<rectangle>	Simple choose and draw from pallet
<animation> using <frame>	The same effect can be done using the layering concept on the screen
<image>	Simply insert an image by importing
<ticker>	A simple tween motion between two frames

Additionally ECZAF supports several other possible animations:

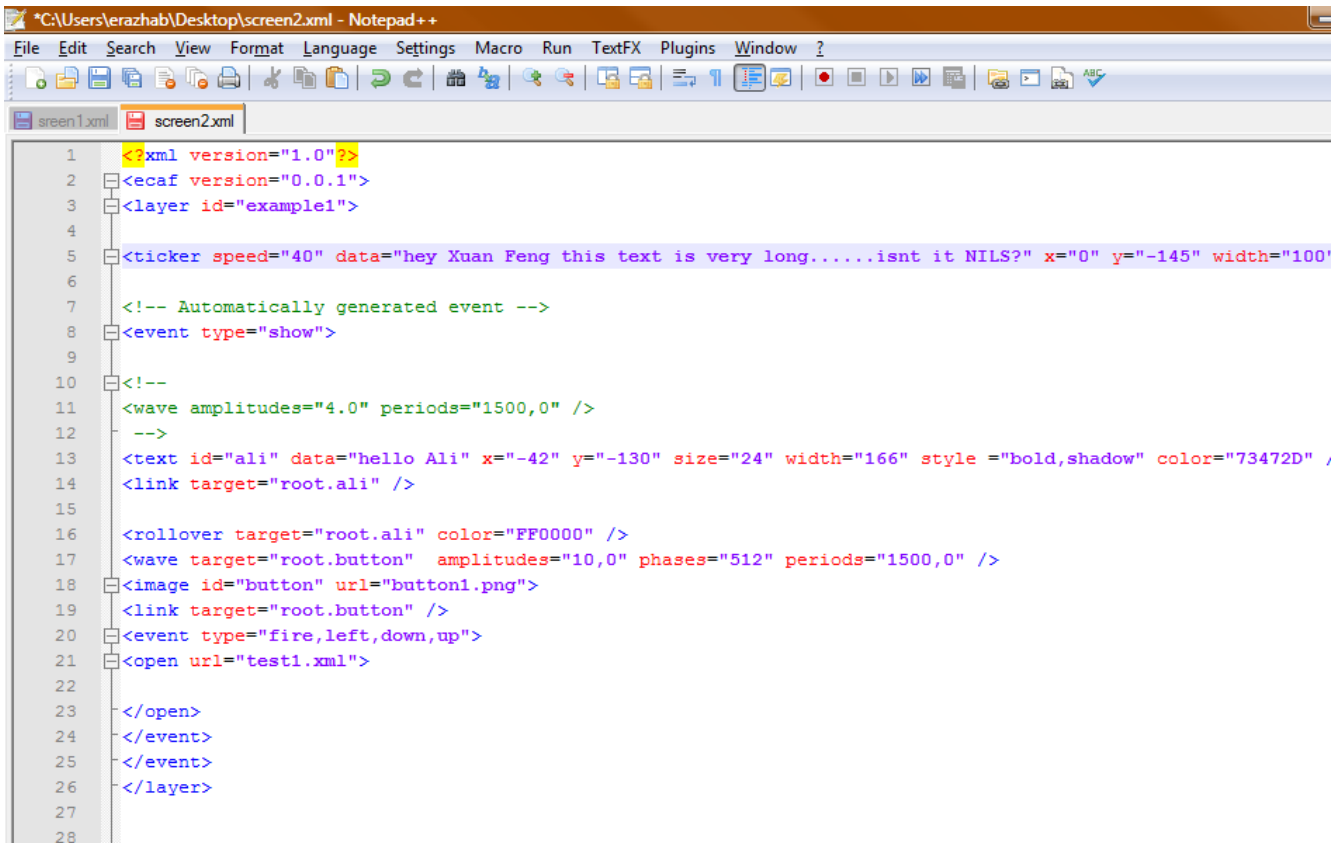
- <pointerarea > This plugin defines an area on the screen to use for detecting pointer gestures.
- <pointerflick > Add the possibility to drag & flick parent graphical component on one axis and at the same time have other graphical components follow the movement.
- <tooltip > The purpose of this tag is to display a tooltip. The text can be changed using the rollovertag.
- <Scale > The purpose of this tag is to scale images from one size to another, in a zooming fashion.
- <flipimage > The purpose of this tag is to display an animation on the screen.
- <dragndrop > Add touch drag & drop effect on parent graphical component.

We observe that ECAF only supports the above 10 or 15 types of animations. While other types of animations such as rotate in a 3D fashion, stroke hinting, dashed/solid/ragged/stipple effects on a tween, shape tween, background color, foreground color , setting the dimensions dynamically, easing effect (from fast to slow or vise versa) , are not available in ECAF. Moreover in Flash Lite there is a complete drag and drop toolbox for creating objects and animating them. Hence Flash Lite offers a much more complete and powerful means of generating animations.

4.2.2 Code size in lines of code

The simple ECAF application containing the animation (with splashing light effects and rotation) mentioned above had two .xml files named: screen.xml and screen1.xml. These are shown in Figure 33 and Figure 34 (respectively).

The total number of lines of codes in both files is 26+36=52 lines. Comparing this with the number of lines of code used in Action Script for Flash, it took only 5 lines and the rest was handled as properties. In this comparison we only include the screen development lines of code and the Java ME code was not considered. Note that this count did not include the libraries used in Capuchin or in ECAF and the .jad files. If we compare on the basis of number of lines of Java involved, then the number of lines of code in both are almost equal.



```
1 <?xml version="1.0"?>
2 <ecaf version="0.0.1">
3 <layer id="example1">
4
5 <ticker speed="40" data="hey Xuan Feng this text is very long.....isnt it NILS?" x="0" y="-145" width="100"
6
7 <!-- Automatically generated event -->
8 <event type="show">
9
10 <!--
11 <wave amplitudes="4.0" periods="1500,0" />
12 -->
13 <text id="ali" data="hello Ali" x="-42" y="-130" size="24" width="166" style ="bold,shadow" color="73472D" ,
14 <link target="root.ali" />
15
16 <rollover target="root.ali" color="FF0000" />
17 <wave target="root.button" amplitudes="10,0" phases="512" periods="1500,0" />
18 <image id="button" url="button1.png">
19 <link target="root.button" />
20 <event type="fire,left,down,up">
21 <open url="test1.xml">
22
23 </open>
24 </event>
25 </event>
26 </layer>
27
28
```

Figure 33: ECAF Screen.xml


```
*C:\Users\erazhab\Desktop\sreen1.xml - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
sreen1.xml
3 <layer id="example3">
4 <!-- Automatically generated event -->
5 <event type="show">
6 <link target="root.btnOne" />
7 <move target="root.selector" destination="root.btnOne" duration="150"
8 interpolation="decelerating" />
9 </event>
10 <!-- Background image -->
11 <image url="bg.png" x="0" y="-20" />
12 <!-- Button 1 -->
13 <image id="btnOne" url="button1.png" x="-50" y="-40">
14 <event type="down">
15 <link target="root.btnTwo" />
16 <move target="root.selector" destination="root.btnTwo" duration="1000"
17 interpolation="decelerating" />
18 </event>
19 </image>
20 <!-- Button 2 -->
21 <image id="btnTwo" url="button2.png" x="-50" y="20">
22 <!-- Keypressed generated event -->
23 <event type="up">
24 <link target="parent.parent.btnOne" />
25 <move target="root.selector" destination="root.btnOne" duration="150"
26 interpolation="decelerating" />
27 </event>
28 </image>
29 <!-- Selector image -->
30 <image id="selector" url="selector.png" x="0" y="0" />
31 </layer>
32 <!-- Second Layer -->
33 <layer id="example2">
34
35 </layer>
36 </ecaf>
```

Figure 34: ECAF Screen1.xml

4.2.3 Memory footprint

Flash has a smaller memory footprint as compared to ECAF when doing typical simple animations. To compare the two for a more complex case, we implemented the same application using both methods, and observed that it is possible to implement complex animations in ECAF and the similar animations using Capuchin in Flash Lite. We observed that in this more complex application the total memory resources used were roughly the same.

We cannot compare the application in an emulator using a profiler as there is no skin available for the emulator for JP 8.4 phones which are Capuchin enabled. Another approach would be to use a common background application tester in order to see the running time and memory size of the object loaded. Unfortunately, we can not do this as Java runs in a sandbox and does not allow other applications to access the array, heaps, and stacks which are already in use by other application. However, another thread could be implementing that runs in the same Java Virtual Machine (JVM), hence it could access the information about heap free space and collect other information of interest.

Thus we considered a possible way to compare both ECAF and Capuchin applications using the same metric. Hence we compared the jar size before putting it on the phone for testing. The jar file size of ECAF for our sample application was 589 kb which was much larger than the jar file size of Capuchin - 136kb. Thus the Capuchin application is only ~23% of the size of the ECAF application, when we compared the applications using the formula:

$$(\text{Capuchin Jar Size}/\text{ECAF Jar Size}) * 100\%$$

We believe that Capuchin will have a larger footprint when more effects or animations are used (rather than only simple animations); however, we do not have numerical examines to establish this yet. Moreover, animation in ECAF is not very efficient. For example, in ECAF texts will not ticker if the text does not fit in some percentage of the width defined in the <ticker> tag.

4.2.4 Reusability

Both ECAF and Capuchin are equally reusable. In ECAF we can reuse the code in other XML files or by adding more layers in the same file. In Flash Lite a MovieClip can be considered as a class and it can be reused many times by initiating instances of it.

4.2.5 Flexibility

Flash Lite is more flexible to develop and program in. It uses Action Script as a programming language. Moreover, the drag and drop feature of the development environment makes it very easy to draw changes and add additional effects. Additionally, there is a property pallet where we can set the properties of items. In ECAF we have to make changes in the XML tags, hence we need to write new XML files and delete the old ones to make changes. Furthermore, there is no drag and drop pallet in the ECAF development environment.

4.3 Presentation technologies for mobile TV

There have been three major technologies that have been developed to meet the needs of the cellular phone community. They are:

- Java Micro Edition (Java ME)
- Microsoft's .NET Mobile Edition
- Adobe's (formerly Macromedia's) Flash Lite

There are a number of other technologies that are in use, but the majority of development today is centered on these three main technologies. Some of these other languages are:

- Hecl -- Hecl is a small language written in Java to facilitate writing applications for a cellular phone. See section 4.3.1.
- ECAF—works along side Java ME (see section 2.3 on page 4)
- LWUIT [26] -- The LightWeight User Interface Toolkit (LWUIT) is an UI library that is binded together with applications and helps content developers create compelling and consistent Java ME applications. It supports a basic set of visual components, flexible layouts, style and theming, animated screen transitions, a simple and useful event handling mechanism, and more.
- SVG Tiny 1.2 [27]—has support for Java using JSR 287 (see section 4.3.2)

In this section we will evaluate these various technologies for the specific task of creating a Mobile TV application. For such an application we need:

- XML parsing,
- a means to play the video stream(s),
- soft keys to enable the user to easily move forward and backward in the video,
- a means of drawing graphics on a 2D screen, and
- easy means of communication between the graphics API and the backend programming language.

We will judge the technologies in terms of each of these criteria. If one of these technologies supports all of these features, then we can create a mobile TV application using this technology. If the underlying technology does not support all of these criteria, then we can see if there is another technology that can be used along side it to implement our desired mobile TV application.

4.3.1 Hecl support for Java ME

Hecl was designed by David Welton to make it easy for a Java programmer to develop applications for a cellular phone. Details of Hecl can be found at [28].

4.3.1.1 HTTP

Even a basic Java ME-enabled cell phone can access web pages. However, Hecl provides some additional (and quite basic) HTTP commands:

- `http.geturl` Fetch the contents of a URL
- `http.formatQuery` URL Encode a request

Hecl also includes three short cuts, `http.data` to get the data from `http.geturl` response, `http.ncode` to get a numeric code from such a response, and `status` to get the status of such a response.

4.3.1.2 K-XML

The K-XML extension provides a kXML 2 parser for Hecl. In particular it implements an `XmlPull` parser (for further details see the interface documentation is at <http://xmlpull.org/>). This extension provides a number of kXML methods:

- `kxml.create` — Returns a kXML parser object
- `kxml.input` — Sets the input stream to the kXML-parser
- `kxml.nexttag` — Call `kxml.next` and return an event if the tag is `START_TAG` or `END_TAG`, otherwise throw an exception
- `kxml.next` — Get the next parsing event
- `kxml.requirestart` — check if the current event is `START_TAG`
- `kxml.requireend` — check if the current event is `END_TAG`
- `kxml.attrcount` — return the number of attributes of the current (start) tag
- `kxml.attrvalue` — returns the value of the attribute
- `kxml.getname` — return the name of the current element for a `START_TAG` or `END_TAG`

4.3.1.3 Other Hecl extensions

Hecl also provides functions to implement a `RecordStore`, functions for encoding and decoding base 64 strings, and a small set of functions to facilitate file interactions. For details see the Hecl documentation at <http://www.hecl.org/docs/>.

4.3.1.4 Evaluation of Hecl + Java ME for our Mobile TV application

As seen above Hecl provides an easy and effective means of parsing XML, Java can invoke a media player (such as VideoLAN's VLC – see section 3.4.5.2 on page 30), Java can implement soft keys, Java has a number of tool kits for 2D graphics, and using one of these Java graphical tool kits there is not problem with communication as all of the communication can be done within Java. Thus Hecl + Java ME would seem to meet all of the criteria for implementing our Mobile TV application.

The weakness of Hecl + Java ME with regard to implementing our application is lack of scalable graphics (making nice text and graphics hard to render to the screen).

4.3.2 SVG Tiny 1.2 support for Java ME

Scalable Vector Graphics (SVG) [29] is an XML-based file format that describes two-dimensional vector graphics, both interactive and animated. SVG is strict in compliance with the XML syntax, and it has nothing to do image resolution. SVG graphics format has the following advantages:

- The image files are readable and easy to modify and edit.
- SVG files can embed JavaScript /ECMAScript to control the SVG objects.
- SVG can easily create a text index in order to achieve content-based image search
- SVG can be used to dynamically generate graphics such as using SVG to dynamically generate interactive maps embedded in web pages, and display it to the end-user.
- It supports for embedded external images including PNG, JPEG, SVG etc.

SVG Tiny, also known as SVGT, short for Scalable Vector Graphics, Tiny Profile, is also a subset of standards SVG. Its main goal is to provide vector graphics format for highly restricted mobile devices such as cellphones. The SVG Tiny 1.2 specification adds new features for the use of mobile devices. It is compatible with video, audio and scripts, as well as style-related graphical features such as gradients, stroke and text styles.

We can play HTTP resources in the SVG Tiny 1.2 viewer (which is supported by Java ME as per JSR 287) [30]. The code [31] for playing a video stream by accessing it from HTTP written in XML is simply:

```
<g requiredFeatures="http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo"
    transform="translate(-21,-34) scale(1.24) rotate(-30)">
    <rect x="6" y="166" width="184" height="140" fill="none" stroke="blue" stroke-width="4"
/>
    <video xlink:href="http://www.kth.se/~xuanf/ski.avi" audio-level=".8" type="video/x-
msvideo"
        x="10" y="170" width="176" height="132"/>
</g>
```

Its scalable vector feature makes it suitable for developing Mobile TV applications associating with Java ME, although it needs to face the problems like how to compete with Flash/Flex and the level of support from manufacturers in the local environment.

4.3.3 MIDP 3.0 [32]

As Java for mobiles was released in 1998, MIDP has been widely accepted and adopted by more and more manufacturers and developers. It has become a well known and popular programming environment for handsets due to its availability. Hence, it is a good choice for mobile application programmers.

Although it is popular, Java ME has some major drawbacks. For example, MIDP 1.0 and 2.0 do not support concurrency, background execution, or inter-MIDlet communications. MIDP 3.0, specified in JSR 271, is a new upcoming version of MIDP. MIDP 3.0 extends MIDP 2.0 by enhancing MIDlets to be auto-launched, run in the background, and it also enables inter-MIDlet communication. New User Interface functionality includes support for splash, idle screen and screensavers, text input into Canvas elements, tables, tabbed panes, splash screen, scalable images and animated GIF, menus, and form layouts help developers create compelling content which makes it easier to satisfy users' needs.

MIDP 3.0 would seem to be a suitable platform on which to develop a Mobile TV application. The enhanced functionality plus Java's high efficiency might make it the best choice for Mobile TV client.

4.3.4 Microsoft .NET Compact Framework (.NET CF)

The Microsoft .NET Compact Framework (.NET CF) is a version of the .NET Framework that is designed to run on Windows CE based mobile/embedded devices such as PDAs and mobile phones. The .NET Compact Framework uses some of the same class libraries as the full .NET Framework and also a few libraries specifically designed for mobile devices (for example, to support the Windows CE InputPanel a soft keyboard). The libraries in the .NET Compact Framework are scaled down to require less memory space [33].

To run applications developed for the .NET Compact Framework, the device must support the Microsoft .NET Compact Framework runtime. Some of the operating systems which include .NET CF are Microsoft's Windows CE 4.1, Microsoft Pocket PC, Microsoft Pocket PC 2002, and Microsoft's Smart phone 2003. The Microsoft .NET Compact Framework 3.5 Redistributable package contains the common language runtime and class libraries for the .NET Compact Framework.

.NET CF also supports ASP.net for GUI development, C#.net for XML parsing (XmlReader, XmlDocument) and ADO.net for handling data in the DataSet. The ADO.NET DataSet is natively an XML structure and therefore lends itself to XML-based communication and persistence very well. A dataset can be stored as local XML files between application sessions. XML retrieved from a dataset can easily be used in system integration between client and server, especially if the server also relies on the .NET Framework. The XML file passed to the server can then be used to instantiate and populate a dataset that can be processed on the server. Additional information is available at [34].

Thus Microsoft .NET Compact Framework is also an option for a Mobile TV application. But a prerequisite is that the mobile devices must run one of the embeded operating systems such as Pocket PC 2000, Pocket PC 2002, Pocket PC 2003, Pocket PC Phone Edition, or Windows CE .NET 4.1 or later.

4.4 Pure Flash Lite TV Client Application Concept

A pure Flash Lite client can be used to implement a Mobile TV application. As Flash Lite supports the requirements stated earlier. Specifically:

1. Graphics design

Since Flash is famous for the graphics development, it is not surprising that Flash Lite provides a good graphical interface.

2. Accessing an XML file

Before loading external XML data, it is vital to set the `ignoreWhite` property of the `my_xml` object to `true`, because this strips white space from between elements.

```
Var my_xml:XML = new XML();  
my_xml.ignoreWhite = true;
```

It is easy to load the XML into the XML object simply by passing the path to the XML file as a parameter to the `load()` method:

```
//absolute path to XML file in external domain  
my_xml.load("http://www.xuanworld.com");
```

3. XML parsing

The Flash Lite player on mobile devices does not contain an automatic way of parsing XML files into its native Action Script data structure. Therefore, we have to manually do this. The methods of the XML class (for example, `appendChild()`, `removeNode()`, and `insertBefore()`) enable the programmer to structure XML data inside Flash to send data to a server and to manipulate and interpret downloaded XML data.

However, in our project, the XML files are written in OMA BCAST format, and require extensive effort to do parse. It takes quite a lot of time to write all the code necessary to do this parsing. Some of the main fragments that need to be parsed include: the Service fragment, Schedule fragment, Content fragment, Access fragment, Preview data fragment, Interactivity data fragment and Supported structure of the service guide delivery descriptor.

4. Data display

To display the dynamic loaded XML data on the screen, it requires communication between the graphics API and the backend programming language. Here we need to create some variables and give them values so that they can be displayed.

Data processing can be done by writing pure ActionScript files (these can be considered as classes), then call these methods to communicate between the frontend and the backend.

5. Availability of Soft keys to move back and forth

In the Capuchin project, soft keys are not supported from inside Flash Lite using ActionScript; as these can only be set by Java. Therefore, Flash Lite does **not** have flexible control of these soft keys. However, in pure Flash Lite applications, it can exercise full control of these soft keys by adding a `KeyListener`. First, the set of soft keys are organized into the Action panel:

```
fsccommand2("SetSoftKeys", "Left", "Right");
```

Next, the programmer creates and registers an object to respond to keypress events by using the following code in the Actions panel:

```
var myListener:Object = new Object();  
myListener.onKeyDown = function() {  
    if (Key.getCode() == ExtendedKey.SOFT1) {  
        // Handle left soft keypress event.  
        status.text = "You pressed the Left soft key.";  
    } else if (Key.getCode() == ExtendedKey.SOFT2) {  
        // Handle right soft keypress event.  
        status.text = "You pressed the Right soft key.";  
    }  
};  
Key.addListener(myListener);
```

6. Playing the video streams

Flash Lite 2.0 does not play a Flash video, but it plays device video. In other words, Flash Lite 2.0 provides the rectangle for the device to play the video and passes several rudimentary commands to the underlying system to control the playing of that video. It appears that the video is being played by the Flash Lite player inline, but the actual work is being done by the device. However, this means that the video playback is device dependent, thus we will need to pay close attention to the device's capability. For instance, what video format(s) and CODEC(s) is/are supported by the device. To view video we have to utilize the device video (so it is really up to the phone to provide the appropriate video CODEC).

Using External Device Video is very useful because storage is always a concern on a mobile device and because this approach gives the flexibility of offering multiple files inside a Flash Lite application. However, this is not to say that there are not potential limitations; in fact certain carriers block some network ports and files, thus the file has to be completely downloaded *before* Flash Lite can call the device's video player to render it.

Streaming video is perhaps the most difficult aspect to implement in Flash Lite, because there are several limitations of the target device (the Sony Ericsson C905 handset that we are using). This is due to the fact that not all the devices actually support the streaming of a video and some operators block the streaming if a user is not using the operator's streaming services. In order to start the playback of a streaming video through an ActionScript, the developer needs only to pass the URL of the video to the play method.

```
video_instance.play("rtsp://server/folder/file.3GP");
```

Adobe has announced that Flash Lite 3.0 will support Flash Video (FLV) and that the Flash Media Server 3 will support Flash Lite. This feature helps all the Flash Lite developers create more interactive applications and offers much more visual impact for the users.

In a nutshell, although we did not have time to create the pure Flash Lite mobile TV client, our conclusion is that based upon the above analysis it would be possible to do so.

5 Conclusions and future work

This research concerned designing, implementing, and evaluating a mobile TV client using Sony Ericsson's Capuchin API [2]. The Capuchin API uses Java ME for backend development offering extensive JSR support; while the front end will be pure Flash Lite and Action Script 2.0.

The focus of this research was the evaluation in CTV lab environment of Sony Ericsson's Capuchin API, i.e. what is good, what is bad, and what is missing from this API in comparison to the existing Ericsson solutions. The Min TV client currently supports the use cases of LTV (Linear TV) and VoD (Video on Demand). Min TV represents a working prototype of a Capuchin based Mobile TV client.

5.1 Conclusions

Using Capuchin in the CTV lab environment has shown that Capuchin has some benefits, but suffers from many drawbacks. At this early stage Capuchin technology is not mature enough and does not offer a better solution for Mobile TV than the existing ECAF solution. However, if Sony Ericsson develops the missing features such as soft keys, easy debugging of Flash Lite standalone application by emulating the Java data coming from dataRequests or extendedEvents, and provides test emulator support in the SDK, then Capuchin would be better technology than ECAF for future Mobile TV applications.

The comparison of different presentation technologies showed that Java ME is a comprehensive platform for mobile development with all the support needed from third party GUI makers like HECL, SVG Tiny 1.2 etc. Additionally, .NET CF also looks like a good option for development with the scaled down capabilities of different programming language support using Common Language Runtime. MIDP 3.0 is also a possible choice for Mobile TV client with its enhanced functionality plus Java's high efficiency.

Moreover a pure Flash Lite application also looks quite suitable for Mobile TV. As Flash Lite has its own comprehensive programming language, called Action Script, which has support for all the basic programming requirements of the mobile TV application.

5.2 Future Work

- Implement the SIP protocol to communicate with IMS at CTV lab.

As we wrote in section 1.3, there was an existing Mobile TV client in CTV lab which was implemented using IMS as the communication method. Our Min TV application should also have been designed in the same or similar way to use SIP signaling through IMS, because it is an efficient and functional approach. Although we did not use SIP in our implementation, it would be good to add this functionality in the future.

- In order to compare the memory size of ECAF and Flash Lite, test and measure the heap free space, running time, and memory size of the object loaded a new thread should be created and run in the same JVM. (See section 4.2.3 on page 47)
- Development of a complete Flash Lite Mobile TV application.

Due to limited time we could only suggest the development of a complete Flash Lite Mobile TV client. However, we expect that if implemented, the user would have a wonderful experience due to the excellent graphical interface.

- Implement a MIDP 3.0 Enabled mobile TV client with enhanced LCDUI, inter MIDlet communication support, and developing shared LIBlets.

As written in section 4.3.3 on page40, the numerous additions and changes in MIDP 3.0 make it a better choice for Mobile TV application. Thus when the final version of JSR 271 has been released, the Mobile TV client can be developed using it.

References

- [1] Wikipedia, 2009, Adobe FlashLite , [online] [Available at] http://en.wikipedia.org/wiki/Adobe_Flash_Lite [Accessed on 07 May 2009]
- [2] Capuchin API, 2009, Project Capuchin Docs and Tools [online] [Available at] http://developer.sonyericsson.com/site/global/docstools/projectCapuchin/p_projectCapuchin.jsp [Accessed on 13 February 2009]
- [3] Ericsson, 2009, IMS product info [online][Available at] <http://prodcatalog.ericsson.se/Default.asp?ArticleID=C31A70E1-A7CD-4817-9E16-8257F5765034> [Accessed on 09 May 2009]
- [4] AfterDawn, 2009, MPEG-2 Transport Stream [online][Available at] http://www.afterdawn.com/glossary/terms/mpeg2_transport_stream.cfm [Accessed on 07 May 2009]
- [5] Ålands Datakommunikation Ab (Ålcom) <http://www.aland.tv/> [Accessed 07 July 2009]
- [6] ETSI 3GPP. 3GPP TS 26.244; Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP), Release-8, 11 December 2008, [online] [Available at] <http://www.3gpp.org/ftp/Specs/html-info/26244.htm> [Accessed on 09 July 2009]
- [7] Vcodex White Paper, 2007-2008, Overview of H.264 / AVC, [online][Available at] <http://www.vcodex.com/h264overview.html> [Accessed on 12 May 2009]
- [8] Harindra Rajapakshe and Derek Paul Quek, 1995, Video on Demand, [online][Available at] http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/shr/report.html [Accessed on 12 May 2009]
- [9] Northwestel Inc, 2009, Video on demand FAQ [online] [Available at] <http://www.nwtel.ca/services-and-products-frequently-asked-questions/video-on-demand-faq/> [Accessed on 13 February 2009]
- [10] Fredrik Bornskold and Robert Johansson, "Touchscreen GUI Design and Evaluation of an On-Device Portal", Master's Thesis in Computing Science, Department of Computing Science, Umeå University, December 7, 2008. [online] [Available at] <http://www.cs.umu.se/education/examina/Rapporter/BjornskioldJohansson.pdf> [Accessed on 08 July 2009]
- [11] Sony Ericsson. "More Project Capuchin resources: Service MXPs, Service API Generator tool and tutorial", June 2, 2009 [online] [Available at] http://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsjune09/p_new_projectcapuchin_mxps_serviceapi_generator_tool.jsp [Accessed on 10 July 2009]
- [12] Thomas Menguy, 2008, Capuchin: Sony Ericsson strikes back [online][Available at] <http://tmenguy.free.fr/TechBlog/?p=317> [Accessed on 21 April 2009]
- [13] Sony Ericsson, 2009, Getting started with Project Capuchin for Flash and Java developers, [online][Available at] http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp [Accessed on 09 July 2009]
- [14] Service Guide for Mobile Broadcast Services, Approved Version 1.0 – 12 Feb 2009 , Open Mobile Alliance OMA-TS-BCAST_Services-v1_0-20090212_A [online][Available at] http://www.openmobilealliance.org/Technical/release_program/bcast_v1_0.aspx [Accessed on 09 July 2009]
- [15] Adobe Device Central CS3, [online][Available at] http://www.clevelandmmug.org/presos/cs3_device_central_overview.pdf [Accessed on 09 July 2009]
- [16] Adobe Device Central CS3 video tutorials, [online][Available at] <http://bloggy.kuneri.net/2007/04/17/adobe-device-central-cs3-video-tutorials/> [Accessed on 09 July 2009]
- [17] Java Community Process, Mobile Media API, JSR-000135, 22 June 2006 [online] [Available at] <http://jcp.org/aboutJava/communityprocess/final/jsr135/> [Accessed on 09 July 2009]
- [18] Adobe Systems, 2007, Introduction to Flash® Lite™ 2.x Action Script, [online][Available at] http://livedocs.adobe.com/flash/9.0/main/flashlite2_as_intro.pdf [Accessed on 07 May 2009]
- [19] Action Script Technology Center, 2009, [online] [Available at] <http://www.adobe.com/devnet/actionsript/references/> [Accessed on 07 May 2009]

- [20] Apache Ant User Manual, [online][Available at] <http://ant.apache.org/manual/index.html> [Accessed 09 July 2009]
- [21] An Introduction to the Eclipse IDE, [online][Available at] <http://www.onjava.com/pub/a/onjava/2002/12/11/eclipse.html> [Accessed on 09 July 2009]
- [22] Mark Hirs, "Sony Ericsson C905 With Project Capuchin", Editorial, Streetdirectory, 2009 [online] [Available at] http://www.streetdirectory.com/travel_guide/128155/sony_ericsson/sony_ericsson_c905_with_project_capuchin.html [Accessed on 08 July 2009]
- [23] Sony Ericsson Mobile Communications AB, Sony Ericsson J2ME SDK 2.2.0 released, March 31, 2005, [online] [Available at] http://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsmar05/p_j2me_sdk.jsp [Accessed on 08 July 2009]
- [24] VideoLAN, VLC media player, 7 July 2009 [online] [Available at] <http://www.videolan.org/vlc/> [Accessed on 08 July 2009]
- [25] R. Thurlow, RPC: Remote Procedure Call Protocol Specification Version 2, IETF, Network Working Group, Request for Comments: 5531, Draft standard, May 2009. [online] [Available at] <http://tools.ietf.org/html/rfc5531> [Accessed on 07 July 2009]
- [26] LWUIT, [online] [Available at] <https://lwuit.dev.java.net/> [Accessed on 10 July 2009]
- [27] Introduction SVG Tiny 1.2, [online] [Available at] <http://www.w3.org/TR/SVGMobile12/intro.html> [Accessed on 10 July 2009]
- [28] Hecl, [online] [Available at] <http://www.hecl.org/> [Accessed on 10 July 2009]
- [29] W3C, 2005, Scalable Vector Graphics (SVG) Tiny 1.2 Specification [online] [Available at] <http://www.w3.org/TR/2005/WD-SVGMobile12-20051207/SVGMobile12.pdf> [Accessed on 07 May 2009]
- [30] Java Community Process, Scalable 2D Vector Graphics API 2.0 for Java™ ME, JSR-000287 [online] [Available at] <http://jcp.org/aboutJava/communityprocess/pfd/jsr287/index3.html> [Accessed on 08 July 2009]
- [31] SVG Tiny 1.2, 2008, Multimedia,[online][Available at] <http://web4.w3.org/TR/SVGMobile12/multimedia.html> [Accessed on 07 May 2009]
- [32] JSR 271: Mobile Information Device Profile 3, [online] [Available at] <http://jcp.org/en/jsr/detail?id=271> [Accessed on 10 July 2009]
- [33] Wikipedia on .NET CF, 2009,.NET Compact Framework, [online][Available at] http://en.wikipedia.org/wiki/.NET_Compact_Framework [Accessed on 07 May 2009]
- [34] Manage XML Using .NET Compact Framework, [online] [Available at] <http://msdn.microsoft.com/en-us/library/aa446526.aspx> [Accessed on 10 July 2009]

A. Configuration files

A.1. Fetching and inserting EPG data

The following process has to be done on weekly basis in the CTV lab to fetch the EPG data

1) log in to (ssh) 130.100.202.133 (root/root123)
ssh 130.100.202.133

2) From there, ssh till the "catcher", 130.100.202.70 (xmltv/xmltv123)
ssh -l xmltv 130.100.202.70

3) Here the xml file is called
epg_5_letter_lang_fixed.zip

4) Move the file with scp to /tmp on 130.100.202.133

e.g. from 130.100.202.133 run

```
scp xmltv@130.100.202.70:epg_5_letter_lang_fixed.zip /tmp
```

6) Use winscp or filezilla to copy it to your PC
address: 130.100.202.133
user: root
password: root123
port: 22

A.2. Loading an EPG into the EPG Aggregator

Here are instructions on how to upload an XMLTV EPG into the EPG Aggregator. We need:

- An XMLTV file containing an EPG for all the channels throughout the desired time period
- A CTV profile (See further below in this README)
- The tool SOAPUI

A.3. Procedure

- 1) Zip the XMLTV file and the CTV profile
- 2) Base64 encode the zipped file
e.g. using a tool like this <http://www.opinionatedgeek.com/dotnet/tools/Base64Encode/Default.aspx>
- 3) Now we will use SOAP UI tool upload this base64 encoded file.

In SoapUI: File->New WSDL project

Fill out the the "New WSDL project" dialog..
Project name: My Approver Service project

Initial WSDL: <http://10.10.30.32:8080/ApproverService/ApproverServiceImpl?wsdl>

Note: "Initial WSDL" is the URL to the WSDL of the Aggregators ApproverService

4) The ApproverService has a method insertPackage(arg0,arg1)

arg0 : Paste the base64 encoded data here

arg1 : set this to the aggregator user admin

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sg="http://sg.webservices.common.mtv.ericsson.com">
  <soapenv:Header/>
  <soapenv:Body>
    <sg:insertPackage>
      <!--Optional:-->
      <arg0>UEsDBBQAAAAIAAN/VTI4dAI7JGIMADfySw...QAAAAKGQMAAAA</arg0>
      <!--Optional:-->
      <arg1>admin</arg1>
    </sg:insertPackage>
  </soapenv:Body>
</soapenv:Envelope>
```

5) Send this request

You should get a successful response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:insertPackageResponse xmlns:ns2="http://sg.webservices.common.mtv.ericsson.com"/>
  </S:Body>
</S:Envelope>
```

The EPG is now loaded into the system

A.4. CTV Properties file

The CTV profile is a properties file containing additional information needed for

- Specifying a subset of the channels that should be imported at upload
- Declaring additional information needed to create the OMA BCAST fragments needed for the Mobile TV EPG

For more information, see file:

ExampleProfile.properties

A.5. Problems and solutions

If you get an error...

1) You can try emptying the EPG database...

- First undeploy the Aggregator from the EPG server (<http://10.10.30.32:4848>)
Aggregator-CXC1725098_1

- Empty the database by removing the tables in this order

 - FRAGMENT_FRAGMENT_REFERENCES

 - FRAGMENT

 - FRAGMENT_REFERENCES

 - FRAGMENT_SEARCH_CRITERIA

 - SEQUENCE

 - SUBSCRIBER

- Deploy the aggregator again

 - Browse Folders and select: /usr/local/SUNWappserver/domains/domain1/applications/j2ee-apps/Aggregator-CXC1725098_1

- Upload the EPG using the procedure above

2) If the mobile phone is not updated with EPG data, restart jboss on the MTV-Application server (10.10.30.15):

```
svcadm disable jboss
```

```
svcadm enable jboss (then wait approx 3-4 minutes)
```

check that new data has arrived to MTV aggregator with "Fragility"

address: <http://10.10.30.15:8282/dist-interaction-channel-war/InteractionChannel/ServiceGuide/>

B. Phone configurations

B.1. 8.1 megapixel Cyber-shot phone with Project Capuchin API

The C905 Cyber-shot is a HSDPA slider-format phone featuring a 8.1 megapixel camera with face detection, Xenon flash, auto focus, horizontal camera UI and illuminated icons for settings shortcuts. The C905 also features a 320x240 pixel 262K TFT QVGA display, 160MB internal memory, Wi-Fi, GPS and a Digital Living Network Alliance (DLNA)server to connect to compatible devices.

B.2. Phone Model

C905

B.3. Platforms

Sony Ericsson Java Platform 8

B.4. Screen Sizes

240x320

B.5. JSRs & APIs

- CLDC 1.1
- Project Capuchin
- WMA (JSR-120)
- MMAPI (JSR-135)
- Webservices (JSR-172)
- Security and trust (JSR-177)
- Location (JSR-179)
- SIP (JSR-180)
- Mobile 3D (JSR-184)
- Java Technology for the Wireless Industry (JSR-185)
- WMA 2.0 (JSR-205)
- Content Handler (JSR-211)
- SVG (JSR 226)
- Payment API (JSR 229)
- AMS Camera Capabilities only (JSR-234)
- Internationalization (JSR-238)
- Open GL ES (JSR-239)
- MSA (JSR 248)
- Mobile Sensor (JSR-256)
- File/PIM (JSR-75)
- Bluetooth (JSR-82)

B.6. Audio & Video

Audio CODEC: AAC+; Advanced Audio Coding; Adaptive MultiRate - Narrow Band; Adaptive MultiRate - Wide Band; DLS (ringtones); General MIDI (ringtones); MPEG-1 layer 3, MPEG-2 layer 3, MPEG-2.5 layer 3; RealAudio 8; SP-MIDI (ringtones); Wave format PCM; Windows Media Audio 8; Windows Media Audio 9; Extensible Music Format; enhanced AAC+ ; iMelody (ringtones)

Audio CODEC Encoding: Adaptive MultiRate - Narrow Band - Encoding

Audio Mime Type: audio/3gpp ; audio/amr-wb; audio/midi; audio/mid Receiving only; audio/mp3; audio/mpeg3 Receiving only; audio/mpeg Receiving only; audio/mpg3 Receiving only; audio/mpg Receiving only; audio/x-mp3 Receiving only; audio/x-mpeg Receiving only; audio/x-mpg Receiving only; audio/mp4; audio/mp4a-latm Receiving only; audio/rawhz; audio/rhz; AMR audio/amr; AMR audio/x-amr Receiving only; audio/vnd.rn-realaudio; audio/x-pn-realaudio Receiving only; audio/sp-midi Receiving only; audio/wav ; audio/x-wav Receiving only; audio/wma; audio/mobile-xmf; audio/x-midi Receiving only; audio/iMelody Receiving only; audio/x-iMelody Receiving only; audio/x-m4a; text/x-iMelody

Video CODEC: H.263 Profile 0 level 10; H.263 Profile 0 level 45; H.263 Profile 3 level 10; H.263 Profile 3 level 45; H.264 Level 1; H.264 Level 1b; MPEG-4 Level 0 Part 2 Visual Simple profile; MPEG-4 Level 0b Part 2 Visual Simple profile; MPEG-4 Level 1 Part 2 Visual Simple profile; MPEG-4 Level 2 Part 2 Visual Simple profile; MPEG-4 Level 3 Part 2 Visual Simple profile; Real Video 8; Windows Media Video 9 Local + streaming: QCIF@15 fps 128 kbit/s

Video CODEC Encoding: H.263 Profile 0 level 10 Encoding; MPEG-4 Level 0 Level 0b Level 1 Level 2 Level 3 Visual Simple profile

Video MIME Type: 3GPP video/3gpp; MP4 video/mp4; MP4 video/mp4v-se Receiving only; MP4 video/mpeg4 Receiving only; Real8 Video video/vnd.rnrealvideo; Real8 Video video/x-pn-realvideo Receiving only; SDP application/sdp; Windows Media video/wmv

B.7. Flash

Flash Lite version 2.0; Flash colour depth, 16-bit (536 colours); Flash in Screensaver; Flash in Wallpaper; Flash in browser; Heap size for Flash per content type (MB) 2; Language support for device fonts and input according to i-mode specifications; Enabled sound formats for Flash: MIDI

B.8. SVG

Anti-aliasing; Links; Opacity and gradients; SVGT 1.1; SVGT 1.2; System fonts

B.9. Connectivity

Network Data Support: EDGE; GSM/GPRS; UMTS; HSDPA

Radio Bands: 1800; 1900; 2100 ; 850 ; 900 MHz

Local Connectivity: Bluetooth Wireless Technology; WLAN (Wireless Local Area Network); GPS/aGPS

B.10. Regions

Americas; Asia Pacific (including China); Europe; Middle East, and Africa

B.11. Miscellaneous

Theme Version: Themes Version 4.7

External Storage: Memory Stick Micro™ (M2™)

Browser: NetFront™ v 3.4

DRM: OMA v1.0 ; OMA v2.0

C. Source code for the ActionScript files

C.1. retrieveInfo.as

```
1  /*
2  written by Xuan Feng
3  */
4  class retrieveInfo {
5
6      //constructor
7      public function retrieveInfo() {
8      }
9      //split the big string into channels
10     public function getChannels(totalInfo:String):Array {
11
12         var channels:Array = totalInfo.split("#");
13
14         var channelList:Array = new Array();
15         for (var i = 0; i<channels.length; ++i) {
16             channelList[i] = getChannel(channels[i]);
17         }
18         return channelList;
19     }
20
21     //get channel name and programs from big channel string
22     public function getChannel(channels:String):Channel {
23         var programList:Array = new Array();
24
25         var channel:Array = channels.split("#");
26         var channelName = channel[0];
27         var programsStr = channel[1];
28
29         var programs:Array = programsStr.split("&");
30
31         for (var i = 0; i<programs.length; ++i) {
32             programList[i] = getProgram(programs[i]);
33         }
34         return new Channel(channelName, programList);
35     }
36
37     //get detailed program information from program strings
38     public function getProgram(program:String):Program {
39
40         var programInfo:Array = program.split("^");
41         var programName:String = programInfo[0];
42         var startTime:String = programInfo[1];
43         var endTime:String = programInfo[2];
44         var description:String = programInfo[3];
45
46         return new Program(programName, startTime, endTime, description);
47     }
48 }
```

C.2. Channel.as

```
1 class Channel {
2     private var channelName:String;
3     private var programList:Array;
4
5     //constructor
6     public function Channel(channelName:String, programList:Array) {
7         this.channelName = channelName;
8         this.programList = programList;
9     }
10
11     //extract the name info
12     public function getChannelName():String {
13         return channelName;
14     }
15
16     //a list,each element is a string of one program
17     public function getProgramList():Array {
18         return programList;
19     }
20 }
```

C.3. Program.as

```
1 /*written by Xuan Feng*/
2 class Program {
3     private var programName:String;
4     private var startTime:String;
5     private var endTime:String;
6     private var pDescription:String;
7
8     //constructor
9     public function Program(programName:String, startTime:String, endTime:String, description:String) {
10         this.programName = programName;
11         this.startTime = startTime;
12         this.endTime = endTime;
13         this.pDescription = description;
14     }
15     public function getProgramName():String {
16         return programName;
17     }
18     public function getStartTime():String {
19         return startTime;
20     }
21     public function getEndTime():String {
22         return endTime;
23     }
24     public function getDescription():String {
25         return pDescription;
26     }
27 }
```

C.4. XDList.as

```
1 class XDList extends MovieClip {
2     // config
3
4     private static var cSelection:Number;
5     private var cScroll:Number;
6
7     private var items_mc:MovieClip;
8     private var scrollbar_mc:MovieClip;
9
10    // depths
11    private static var scrollbar_depth:Number = 2;
12    private static var items_depth:Number = 1;
13
14    // data properties
15    private static var dataSet:Object; // array of objects containing the data for the list
16    private var dataSetPro:Object;
17    private var nItems:Number; // total items in list
18    private var showingItems:Number;
19    private var sessionName:String;
20
21    // user defined
22    [Inspectable(defaultValue=4)]
23    public var nItemsShowing:Number; // total item showing in menu. does not support < 3
24    [Inspectable(defaultValue=0)]
25    public var itemPadding:Number; // vertical space between items
26    [Inspectable(defaultValue=250)]
27    public var scrollbarXpos:Number; // _x position of the scrollbar
28    [Inspectable(defaultValue="itemRenderer")]
29    public var itemLinkage:String; // linkage name of item renderer
30
31
32    // XDList Constructor
33    public function XDList() {
34        //
35    }
36    public function activate() {
37        setSelectedItem();
38    }
39    public function deactivate() {
40        hideSelectedItem();
41    }
42    //
43    private function setDataSource(listData:Object,proData:Object):Void {
44        // reset
45        cSelection = 1;
46        cScroll = 0;
47        // get new data
48        dataSet = listData;
49        dataSetPro = proData;
50        nItems = dataSet.length;
51        // build list
52        createList();
53    }
54
55    private function setDataSourcePro(chanName:String,progData:Object,proTime:Object):Void {
56        sessionName = chanName;
57
58        // reset
59        cSelection = 1;
60        cScroll = 0;
61        // get new data
62        dataSet = progData;
63        dataSetPro = proTime;
64        nItems = dataSet.length;
65        trace(nItems);
66        // build list
67        createList();
68    }
69
```

```

70 private function createList():Void {
71     // clear old item container
72     removeMovieClip(items_mc);
73
74     // create new item container
75     this.createEmptyMovieClip("items_mc", items_depth);
76
77     // check that showingItems isn't greater than total items
78     showingItems = Math.min(nItemsShowing, nItems);
79
80     // create the menu items
81     drawItems();
82
83     // create the menu scrollbar
84     //drawScrollbar();
85 }
86 private function drawItems():Void {
87     // create items
88     for (var i=1; i<=showingItems; i++) {
89         var cItem:MovieClip = items_mc.attachMovie(itemLinkage, "item"+i, i); // create item instance
90         cItem._x = 250;
91         cItem._y = (cItem._height+itemPadding)*(i-1);
92         cItem.itemName_str = dataSet[(i-1)+cScroll].itemName;
93         cItem.itemName.text = cItem.itemName_str;
94         cItem.currentPro_str = dataSetPro[(i-1)+cScroll].currentPro;
95         cItem.currentPro.text = cItem.currentPro_str;
96     }
97 }
98 private function setSelectedItem():Void {
99     var cItem = items_mc["item" + (cSelection - cScroll)];
100     //var cItem = items_mc["item" + cSelection];
101     cItem.gotoAndStop(2);
102     cItem.itemName.text = cItem.itemName_str;
103     cItem.currentPro.text = cItem.currentPro_str;
104 }
105 private function hideSelectedItem():Void {
106     var cItem = items_mc["item" + cSelection];
107     cItem.gotoAndStop(1);
108     cItem.itemName.text = cItem.itemName_str;
109     cItem.currentPro.text = cItem.currentPro_str;
110 }
111 public function prevItem():Void {
112     if (cSelection<=1) {
113         // loop around menu
114         cSelection = nItems;
115         if (nItems>showingItems) {
116             cScroll = nItems-showingItems;
117         }
118     } else {
119         // previous item
120         cSelection--;
121     }
122     // set cScroll, check for hint push
123     if ((cSelection-cScroll)<1 && cSelection>0) {
124         cScroll--;
125     }
126     // create the new list
127     createList();
128     setSelectedItem();
129 }

```

```

130     public function nextItem():Void {
131         if (cSelection>=nItems) {
132             // loop around menu
133             cSelection = 1;
134             cScroll = 0;
135         } else {
136             // next item
137             cSelection++;
138         }
139         // set cScroll, check for hint push
140         if ((cSelection-cScroll)>(showingItems) && cSelection<=nItems) {
141             cScroll++;
142         }
143         // create the new list
144         createList();
145         setSelectedItem();
146     }
147
148
149     public static function getSelectedTxt():String {
150         return dataSet[cSelection-1].itemName;
151     }
152 }

```

C.5. Session.as

```

1  /*Written by Xuan Feng*/
2  class Session{
3      private static var selectedChannel:String;
4      private static var selectedPro:String;
5      private static var selectedChannelTwo:String = "default";
6
7      public static function setSelectedTxt():Void {
8          selectedChannel = XDList.getSelectedTxt();
9      }
10
11     public static function getSelectedTxt():String {
12         return selectedChannel;
13     }
14     public function setSelectedVod(str:String):Void {
15         selectedChannel = str;
16     }
17     public function getSelectedVod():String {
18         return selectedChannel;
19     }
20     public function setSelectedContent(str:String):Void {
21         selectedChannel = str;
22     }
23     public function getSelectedContent():String {
24         return selectedChannel;
25     }
26     public static function setSelectedChannelTwo():Void {
27         selectedChannelTwo = XDItemGapper.getSelectedTxt();
28         trace("selectedChannelTwo is "+selectedChannelTwo);
29     }
30     public static function getSelectedChannelTwo():String {
31         return selectedChannelTwo;
32     }
33 }

```

C.6. retrieveVodInfo.as

```
1 class retrieveVodInfo {
2
3     public function retrieveVodInfo() {
4     }
5
6     public function getProgrammes(wholeString:String):Array {
7
8         var programmes:Array = wholeString.split(",");
9         return programmes;
10    }
11
12    public function getProgramDetails(program:String):vodProgram {
13
14        var programInfo:Array = program.split("&");
15        var programName:String = programInfo[0];
16        var programDes:String = programInfo[1];
17        var programIcon:String = programInfo[2];
18        var programURL:String = programInfo[3];
19
20        return new vodProgram(programName, programDes, programIcon, programURL);
21    }
22
23 }
```

C.7. vodProgram.as

```
1 class vodProgram {
2     private var programName:String;
3     private var programDescription:String;
4     private var programIcon:String;
5     private var programURL:String;
6
7     public function vodProgram(programName:String, programDescription:String, programIcon:String,
8         this.programName = programName;
9         this.programDescription = programDescription;
10        this.programIcon = programIcon;
11        this.programURL = programURL;
12    }
13    public function getProgramName():String {
14        return programName;
15    }
16    public function getProgramDescription():String {
17        return programDescription;
18    }
19    public function getProgramIcon():String {
20        return programIcon;
21    }
22    public function getProgramURL():String {
23        return programURL;
24    }
25 }
```

