UPTEC IT 11 011

Examensarbete 30 hp
Juni 2011

An Ecosystem for a Unified Mobile, Tablet and TV Experience

Erik Hedberg

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

An Ecosystem for a Unified Mobile, Tablet and TV Experience

Erik Hedberg

There's a distinct trend showing in the community. The numbers of users of smartphones/tablets and IPTV services are increasing. Choosing a smartphone is quite natural when it comes to upgrading or buying a new mobile phone. IPTV is the new and modern way of providing TV transmissions. The transissions are equipped with certain extra features, such as applications. The applications are explicitly controlled by the TV remote. The purpose of this thesis project is to investigate on how to enrich the interaction between the user and the application, by introducing the smartphone as a replacement of the remote.

The possibility to create the interaction was conducted by investigating existing applications and the technologies available at the applications and smartphones/tablets. Through an explorative and iteratve process, prototypes have been developed in order to support the theories and evaluate the possible solutions. The prototypes have been communicating with the IPTV applications through a server, which was also prototyped. The result of the project is a set of recommendations on how to proceed with the development, function libraries for iOS, Android and IPTV appplications and also a server environment for supporting the communication.

Handledare: Ted Björling
Ämnesgranskare: Arnold Pears
Examinator: Anders Jansson
ISSN: 1401-5749, UPTEC IT11 011
Tryckt av: Reprocentralen ITC

1 Sammanfattning

Detta är ett examensarbete för Uppsala Universitet i samarbete med Accedo Broadband AB. Arbetets problembeskrivning har varit att konstruera ett ekosystem som erbjuder en rik interaktion mellan IPTV-applikationer och handhållna enheter (såsom smartphones och tablets).

Accedo Broadband AB är ett företag, baserat i Stockholm. Företaget ägnar sig åt att utveckla applikationer och tillhandahålla portaler till leverantörer av IPTV-tjänster. Då IPTV anses vara på stark frammarsch och teknologin för smartphones vara tämligen sofistikerad, önskar Accedo hitta ett ekosystem som kan användas för att skapa och erbjuda interaktion mellan dessa enheter.

Målet för examensarbetet är att ta fram klientbibliotek för utvalda mobila plattformar, bibliotek för webbbaserade IPTV-applikationer, en servermiljö och ett formellt protokoll som definierar kommunikationen och typer av meddelanden som skickas. Dessa bitar ska kunna användas till att utöka befintliga applikationers funktionalitet och nya applikationer.

Syftet med interaktionen är att kunna låta den vanlige hemanvändaren använda sin smartphone till att styra, eller på annat sätt kommunicera, med IPTV-applikationen. I dagsläget används explicit fjärrkontrollen till att interagera.

Då antalet tillgängliga mobila plattformar överstiger det antal som detta examensarbete kan omfatta, kommer antalet reduceras till en mängd som anses vara skälig. Faktorer som spelar in på hur de väljs ut är marknadsandelar och möjlighet till utveckling av applikationer av tredjepart.

Funktionaliteten hos ekosystemet kommer att testas och evalueras genom att bygga ut en befintlig applikation och även skapa en sandlåda som fungerar som proof-of-concept för funktioner.

Contents

2 Introduction

2.1 Motivation

There are currently two macro trends prevalent in the realm of home electronics:

- The popularity of hand held, "smart" devices, such as iPhone and Android based cell phones, increases by the day. What was seen as an exotic product a couple of years ago is soon to become a commodity. Small applications, so called "Apps", have become a first-class citizen, both in the consumer area and in the developer area.
- TV sets and companion devices become more and more connected to the Internet. The traditional, linear TV viewing experience is vanishing in favour of an interactive and multifaceted experience. Just like in the context of the hand held devices, much of the usage of these TV sets will be revolving around the so called "Apps".

Both these trends converge in their dependence on the Internet and the services provided here. There is certainly a lot of aspects that separates these devices from each other, such as the available screen real estate, the methods of interaction and the overall performance. Nevertheless, the fact that they all provide a connection to the Internet as well as a possibility for a somewhat rich experience, they bear much resemblance to each other. Even if the individual potential for these two types of devices is large, it is believed they convey an even greater potential when combined.

2.2 Goal

The goal of this thesis was defined to establish a methodology for implementing an ecosystem providing the rich interaction between IPTV-apps and hand held devices. It wasn't supposed to only result in a theoretical implementation, but also a prototype implementation demonstrating the various aspects of the ecosystem. Included aspects were:

- A server-side module for device pairing and session creation, able to route rich instructions from one device to another.
- A protocol/instruction set for relaying information between devices in a secure and performance efficient manner.
- Client-side API:s for Connected TVs and IPTV STB:s, written in JavaScript, and hand held specific libraries, iOS and Android based.

- Proof-of-concept implementations showing an enhanced version of an existing application and a playground for demonstrating the possibilities offered.

2.2.1 Limitations

The modules and protocol which that were to be developed were foremost directed towards evaluation and demonstration of the ecosystem. The intention of the work wasn't to develop a commercial product, ready for the market.

The importance of not creating a mobile application that reached a level of sophistication that outnumbered the IPTV applications and thus removing the need for the latter was clearly emphasized. An example of this case was an application replacing the functionality of the IPTV-app. Interaction is all about two parties mutually acting together. Based on this requirement, it was important to determine on where to direct the users' attention. Should the handheld device only be a complementary remote or a smart device listing data to be published on the IPTV application? Answering these questions might be problematic, as the purpose of the interaction may greatly vary between each application. A clear definition on restrictions and guidelines on how to find the equilibrium wouldn't be covered in this thesis.

2.3 Methods

The statement that constitutes the thesis hypothesis is that there are methods for creating an ecosystem that provides a rich interaction between IPTV-apps and handheld devices. Provided this statement, there is a set of following questions.

2.3.1 Background

The background phase will discover the properties of smartphones and IPTV-apps. By getting a grasp of the fundamental features it will be possible to conclude what to utilize when defining the interaction. As a smartphone probably do possess many input mechanism, it would be of great advantage to reduce the ones handled into a number that fits within the time scope of this thesis project.

Targeted platforms

This phase will also establish what platforms to focus on. As there are many platforms competing over market shares, the set needs to be narrowed down. Considering the thesis goals, Android and iOS are already established members of this set. However, if there are other platforms with larger market share or with a significant growth, they might be considered as well.

2.3.2 Pre-study

Ready solutions

Following the background there will be a study of what ecosystems that already exist. If there are any suitable for expansion or to be used as role model for this ecosystem, they will be closely examined. The discovery process will be conducted by searching IEEE and ACM for articles and also using the knowledge at Accedo Broadband.

Scope of interaction

The phase will contain a brainstorming section with ideas on how to define what's regarded as interaction. The background material will be collected by studying existing IPTV-apps. Based on how they behave and what the users do to send instructions, it will be possible to define how to replace the remote with the smartphone and also come up with ideas of new applications. As the set of the smartphone's input methods is already defined, the brainstorming section will also involve a proposed set of instructions to send to the IPTV-app.

System architecture

As the system architecture is vaguely defined in the goal section, the properties and features need to be settled before proceeding with the exploring software and methods of development of the different actors of the ecosystem.

Communication

One of the core issues of this ecosystem is the network communication with respect to the IPTV applications. The intention is to have a bi-directional communication path between each client and server. It needs to be settled how this issue will be carried out, thus exploring different server architectures and today's methods of communication in IPTV applications. Taken the solution into account, the means of connecting the mobile devices to the server will also be settled.

Methods of mobile app development

In order to be able to target as large audience as possible, there might exist a solutions for cross-platform application development. By finding a set of these utilities and comparing those to the native ways of development, it will be concluded on for which to design the specific libraries.

Prototype

When all the issues above are settled, there will be time allocated for developing a throw-away prototype. The intention of the prototype is to demonstrate the methods of interaction in the ecosystem. The prototype will also acknowledge whether the suggested solutions are suitable for use or whether any phase needs to be redone in order to achieve the goal specified.

As the prototype have no requirements on reusability, it will be rather hasty assembled, although work as according to the specifications.

2.3.3 Design

When the prototype has settled on whether the proposed actors of the ecosystem behave as required, the three parts of the ecosystem will be designed according to the goal specification. The goal is to define and design the different libraries, server implementation and communication protocol (including the instruction set).

2.3.4 Implementation

Following the design derived from the previous phase, the different aspects of the ecosystem will be implemented. By following an iterative process, broken down into the step of programming and evaluating the progress by extending an existing application, it will be possible to verify that the process is on track.

2.3.5 Result

As the resulting libraries and implementations reaches an acceptable level of sophistication, they will be evaluated and discussed during this phase.

2.3.6 Future work

Lastly, ideas on how to improve the features of the system will be presented. As the previous steps progress, some suggestions on implementations and tweaks will be disregarded to avoid making the project not fit within the timeframe. Those neglected ideas will be explained in this phase.

3 Glossary

ACL	Access Control List - A list keeping reference of devices that are or aren't allowed access to a certain resource.
Accelerometer	An electronic device that monitors changes in acceleration among a set of specified axes. Used to detect motion.
AJAX	Asynchronous JavaScript And XML - see XHR
API	Application Programming Interface - A set of rules and instructions defined by an application to facilitate for other applications to communicate.
CATV	Cable TV - A technology for transmitting TV via coaxial cable.
Connected TV	The concept of a TV being connected to an Internet portal, provided by the manufacturer.
CSS	Cascading Style Sheets - Language that defines the look and formatting of markup code.
DOM	Document Object Model - Defines the standard for accessing elements in XML and HTML documents. Utilized by JS and CSS.
IDE	Integrated Development Environment - Software thaat provides comprehensive facilities for sotware development.
HTML	HyperText Markup Language - The building blocks of a web site. The browser interprets the markup and displays the site accordingly.
JS	JavaScript - A prototype-based scripting language commonly used in web applications.
JSON	JavaScript Object Notation - A lightweight data-interchange format, which is language independent consisting of two structures: Collection of name/value pairs and an ordered list of values.
QoS	Quality of Service - A set of technologies aimed to guarantee a certain data transmission performance.
SDK	Software Development Kit - A set of tools making it easier for a developer to create software for a certain platform.
STB	Set-top box - A device that receives a signal from an external source, which is converted and sent to a connected TV.
Swipe	A discrete gesture made on a smartphone's touch-

4 Background

4.1 IPTV and Connected TV

4.1.1 IPTV

IPTV is an emerging technology for television transmission. It's IP based and being carried out through a network owned by the distributor. What's special about IPTV is that each transmission is requested by the viewer. By comparing this to the regular CATV, where every transmission is sent in parallel at individual frequencies, one can note that there's more dedicated bandwidth for each transmission. The only transmission occupying the channel is the one that is requested. It's important to point out that the network used is a closed network and not open (as the Internet).

IPTV standards

There's no particular standard defined for IPTV. Several organizations have made attempts to establish their own standards, such as the Digital Video Broadcasting Project (DVB)[1], the Open IPTV Forum (OIPF) [2] and the Telecommunication Standardization Sector (ITU-T)[3]. The process of establishing a unified standard is ongoing, while the individual actors are promoting their directives about a standard as well. These different standards does not need to be taken into account since the application development very seldom is influenced by those and therefor not within the scope of this thesis.

4.1.2 Connected TV

While IPTV is a subset of all kinds of TV transmissions, Connected TV is the TV vendors' counterpart. Each vendor hosts a portal accessible by their TVs via the Internet. As opposed to IPTV, there's no dedicated network that will guarantee QoS or low latency, nor there are any TV shows in the sense of a regular TV channels broadcasts. Most commonly, each portal provides different applications that the TV user can download and use. The problem with mainting QoS is usually overcome by using Content Delivery Networks, such as Akamai.

Vendors

The main companies to which Accedo Broadband develops applications are Samsung, LG and Panasonic. Those three companies do have TVs with the

Connected TV feature on the market. The names of the services are distinct for each company. LGs market is named LG NetCast, Samsungs has a Smart TV with the portal named Smart Hub and Panasonic has recently launched the Viera Connect service.

4.1.3 Benefits of IPTV

Thanks to the fact that the technology of IPTV allows for bi-directional communication, services such as games, Video On Demand, TV with live interaction and time-shifting can be offered by the distributor.

Games

While waiting for a TV show to start, the viewer can access different games to pass the time. Examples of games are quizzes, sudoku, solitaire, poker and black jack. The games are controlled by the remote.

Video On Demand (VOD)

VOD is a service for distributing video and multimedia material requested by the viewer. The difference between VOD and a regular stream is that the material is played on demand. Examples of broadcasted material are live sports events, highlights of games, full movies and video trailers.

The service providers can offer a huge set of different videos and such, playable by the viewer. Everything is indexed in a catalog, which can be infinitely large. The viewer uses a screen based user interface or an Electronic Programme Guide (EPG) to get access to the catalog. What's exposed to the viewer is controlled by the provider and can be set on an individual basis. For instance, the provider has the possibility to offer a subscription based service where the subscriber pays a monthly fee for being granted access to a limited supply of material.

The beauty of this service is that applications can offer a set of related videos. An application providing news stories not only needs to rely on text and images, but can also provide video clips for each news entry. A sports application with information about upcoming and past baseball games can enable for the user to re-watch past games and pay for viewing live games.

There's no additional cabling required. In the case of IPTV, the service is distributed through the same medium as the regular TV transmissions. When it comes to Connected TV, all data is transmitted through the viewer's Internet connection.

Live interaction

Since the data transmission is bi-directional, people are allowed to send data to the server, which can re-distribute it do the other viewers. This allows for people to interact with an ongoing TV-show. This opens up for new ways of influencing the show. Let's say that it requires some kind of voting to proceed, such as American Idol where the contestants are voted on by the watchers. The viewers no longer need to make a phone call to cast a vote. All that's required is to use the remote and select what option they want to put their vote on. Another way of interacting is discussing an on-going TV-show. By bringing up an application that's running as a panel on the side of the screen, the user can watch the TV-show and have peoples' comments being displayed in parallel.

4.2 Smart phones and tablets

4.2.1 Smart phones

A smart phone is considered being an advanced cell phone. What makes it different from the regular cell phone is the backlit color display, possibility to advanced wireless Internet access, large memory and storage space and an advanced operating system. The screen is often with high resolution and a color depth with millions of colors[4]. The advanced wireless access involves technologies such as WiFi and 3G.

History

In 1992, IBM released a concept product called IBM Simon. Apart from supporting phone calls, it also provided a notebook, address book, calendar and e-mail. Everything was controlled by tapping with the finger or the stylus[5]. Nokia followed by releasing the Communicator in 1996[6] and Ericsson countered in 1997 when they released their GS88[7].

During the last decade, the release of smart phones really took off. Ericsson, Nokia, Microsoft and Palm Inc. submitted their contributions to the market. The phone that appeared clearly in the noise was the RIM Blackberry with the capability of wirelessly sending and receiving e-mail, targeted for business users[8].

During the third quarter in 2010 the two largest cell phone manufacturers were Open Handset Alliance (OHA) and Apple [9]. OHA is a union consisting of 79 actors on the cell phone market, whose purpose is to establish an open and effective platform for mobile devices[10]. Among their products, the Android operating system is the most famous one. Since OHA was formed by multiple manufacturers, there's a numerous different devices using the Android OS. In May 2010, the Motorola Droid had the most sales among those[11].

Models

Please see appendix for a list of some popular models. The phones were selected from the Wikipedia article "Comparison of smartphones". In the article, some phones were highlighted as being regarded the best for some attribute. Among those, six models were chosen randomly.

The purpose of the list is only to illustrate the supported features, such as OS, kind of CPU and screen properties. The phones do have quite large screens and are equipped with storage capacity far exceeding the space required for an application[12].

4.2.2 Tablets

The tablet is a unification between smart phones and a regular PC. It is a board whose surface is a touchscreen. The user interacts with the tablet through clicks (taps) and gestures using a finger or a stylus. Just like smart phones, it has a color screen, wireless Internet access and an advanced operating system. Some tablets run specific operating systems[13] while others run the same as regular PCs do (e.g. Windows 7[14]).

A tablet's scope of use is quite different from the smart phone. Since it has a larger screen it can be considered being a competitor to the eBook. The screen facilitates for reading books and newspapers, in contrary to the smart phone's relatively small screen. The iPad lack the feature of making phone calls and texting, while the Samsung Galaxy Tab works as a replacement of a phone.

History

The tablet concept has been around ever since E. Gray received a patent in July 31 1888 for her Telautograph. It was a device able to capture and transmit handwriting[15]. Later on, two US patents for handwriting recognition and touchscreen for user handwriting input were accepted in the 1940s.

Companies made several attempts throughout the years to release tablets aimed for the consumer market. Models such as the Microsoft Tablet PC (2002) and Apples Newton (1992)[16], were all interacted with using a pen or the finger. But none really took off as the iPad did.

When Apple released their iPad in May 2010, it was said to be a magical and revolutionary device. It was indeed something that made an impact on the market, since rumors about the Apple tablet had been around for quite a while and competitors made attempts to release their “groundbreaking” products before Apple’s announcement. According to IDC, the iPad had a market share of 87% in the second quarter of 2010[17].

Samsung released their competitor named “Galaxy Tab” shortly after the introduction of the iPad. As opposed to the iPad, it supported phone calls and ran the Android OS. Some people regarded it as a hybrid between the iPad and a smart phone — the features of a tablet but smaller and potential of a cell phone.

Models

Just like the smart phones, a list of several different models have been attached to the appendix. The list illustrates the different properties of the tablets. The CPU speed and used operating systems seem to be the same as for smart phones. An attribute that varies between the different devices, and is significantly larger than among smart phones, is the size and resolution of the screen. This shouldn’t be a surprising discovery since the tablet is intended to be a device at the size of a small book or a tabloid. When it comes to attributes such as GPS and accelerometer, they seem to be supported as well.

4.2.3 Applications

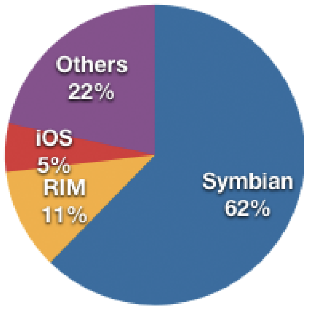
Operating Systems

There are numerous different operating systems, dedicated to smart phones and tablets, competing over the market shares. During 2009, Symbian had the largest market share. At second place came Blackberry, followed by Apple, Microsoft and Google (Android). The largest increase was constituted by Android (1075%) and Apple (83%)[18].

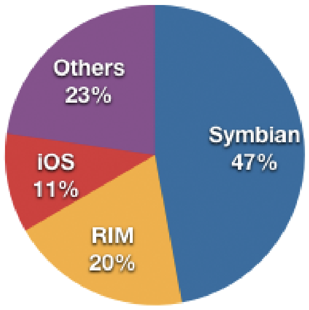
One can note by reading Gartners study of cell phone sales by operating system that Symbian’s share has declined from 62.3% (4Q07) to 36.6% (3Q10). Android has raised from 3.5% (3Q09) to 25.5% (3Q10). RIM did increase from 10.9% in 4Q07 to 20.7% in 3Q09, but has declined since then to 14.8% (3Q10) [19], [20], [21].

Many companies have as of today abandoned Symbian in favor for Android. SonyEricsson announced in the second half of 2010 that they had released a phone using Symbian for the last time[22]. The same announcement was made by Samsung[23] as well as Motorola[24]. The biggest reason for leaving the Symbian world was the fact that those companies actually were members of OHA and thus favoring Android OS. Nokia, the inventor of Symbian, is also discontinuing developing phones with Symbian and have directed itself towards Windows 7 Mobile[25]. The effect on the market will probably be that the largest actor is going to slowly phase out, leaving its top position in favor for more established operating systems.

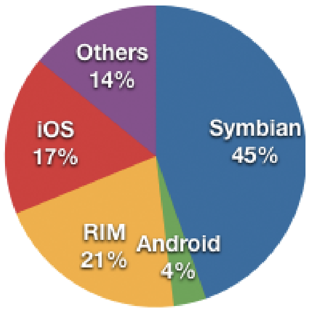
An upcoming competitor with iOS and Android is the latest operating system released by Microsoft, Windows Phone 7. It's reported by The NPD Group that it gained a share of 2% on the US market during its first two months. Windows Phone 7 was released in the US on November 8 2010. Microsofts previous OS is called Windows Mobile 6. Phones running that OS won't, unfortunately, be upgradeable to version 7[26]. Considering that fact, Microsoft will have to start all over on competing with market share, if the Windows Mobile 6 will no longer be maintained.



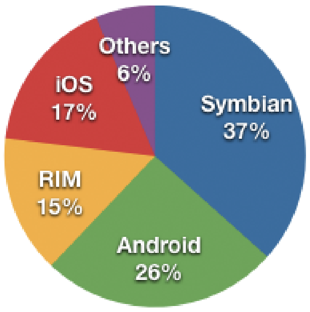
(a) 4th quarter, 2007



(b) 4th quarter, 2008



(c) 3rd quarter 2009



(d) 3rd quarter 2010

Figure 1: Distribution of used operating systems according to Gartner

iOS

Apple's operating system iOS is created specifically for their products, iPod, iPhone and iPad. It's based on the operating system Darwin, which is also developed by Apple and derived from BSD[27] and therefor UNIX mimicking. The user interacts with the system using the touchscreen which reacts to taps, sweeps and specific gestures. The units have no physical keyboard, forcing the user to rely on the virtual keyboard. iOS has support for technologies such as multitouch, accelerometer, gyroscope and compass.

Android

Android is the operating system developed by Google. It based on a branch of the Linux kernel tree[28]. In comparison to Apple iOS, it's open source and released under the Apache License. This means that manufacturers of cell phones don't need to develop their own operating system, since the Android is free for use. A benefit of the Android OS is that the applications created can run on different devices and aren't as specific as the ones designed for iOS[29]. The applications are mostly written in Java and are being run in a JVM called Dalvik[30].

Just as with the iOS devices, the user interacts with the system by tapping and making gestures using the finger. In addition, some devices have hardware keyboards and buttons.

Android also supports multitouch, accelerometer, gyroscope and compass.

Development

Applications for each operating system can be developed by any person with some knowledge in programming. There are plentiful of libraries and frameworks to facilitate the development process available.

iOS

Apple provides a free SDK for creating applications to the iOS devices. The SDK contains a simulator and an interface builder, which enables for the developer to create and run the applications without requiring a physical device. To have the possibility to move the application to iOS hardware, the developer must pay an annual fee of $99 to Apple[31]. All applications are written in Xcode, an IDE created by Apple, on Apple hardware. Windows or Linux machines can't be used, unless they run Mac OS X.

The applications are mainly written in Objective-C, a language developed by a company named Stepstone. The founders wanted to bring the features of Smalltalk to C, thus resulting in the language Objective-C. Apple promotes the use of Objective-C since the heavily used framework Cocoa is accessible through the language. Cocoa and Cocoa Touch are the main frameworks for developing mobile applications.

There's only one way of distributing applications to users and that is through Apple's App Store. It's a virtual store accessible from each device. When an applications is ready to be released, the developer submits it for review to the App Store. If the developer has paid the fee and Apple's review team accepts the application, it's published in the store. For each download, the developer receives 70%[32] of the applications price (unless the developer chose to share it for free).

Android

Android developers experience less restrictions on the development process. Google does provide a plugin (Android Development Toolkit) to the Eclipse IDE[33]. It contains Android libraries and a simulator engine. Since Android devices vary in size, several different kinds of devices can be chosen to be emulated. There's no need to pay a fee for testing on a regular device. The applications can be transferred from Eclipse to the device, or by downloading the application package to the device from a remote location. Since all applications are written in Java and Eclipse is a standard tool for development, Google suggests that using their Android Development Toolkit (ADT) is the most efficient way to build applications.

Applications don't necessarily have to pass a review (as in the iOS case) in order to be shared. The developer is allowed to distribute the application any way he or she prefers to. Google provides a store called Android Market, to which the developer can submit the application. Applications are downloaded from the Market using the Market Application on their device. There are other ways of distributing the product as well. The package can be installed on a device by downloading it off a web site, receiving it by e-mail or having it transferred from a computer. By using the Android Market, the developer has the possibility to publish paid apps, receiving a 70%[34] commission on the revenue.

5 Pre-study

5.1 Previous work

5.1.1 Yahoo! Connected TV's device communication platform

On January 25th 2011 Yahoo! announced the upcoming release of an update of their Widget Development Kit (WDK) allowing interaction between their applications and tablets and smartphones on their Connected TV blog (`http://www.yctvblog.com/`)[35]. The WDK would provide functionality for pushing data back and forth between the TV and the mobile device, thus enabling a richer interaction experience.

A mobile device pairs with the TV via the local network using a service discovery module. This relieves the device from being directed to a specific IP address. The Connected TV app starts its discovery module (based on Apples mDNSResponder) and binds itself to a specific port, with SSL-encryption support. When the service is started, clients starts the discovering process on the local network. Once the desired service is found, the client stores the host address and port for future use. If the session initialized contains a new actor, it needs to establish whether the client is allowed to pair with the TV. If the client key (application specific) isn't blacklisted, the TV will bring up a four digit pairing key. This key must be entered in the client application. If the correct key is entered, both parts store the instance ID, which is used for future pairing. Once the instance ID is created, there's no longer any need to redo the pairing.

All applications are driven by the Yahoo! Widget Engine. It uses an event based model where a device can subscribe to events for a specific service. These events are divided into three categories; input, service and widgets. The input category contains two subcategories — keyboard and navigation service. The keyboard service is basically what it sounds like, when a visible keyboard on the TV or the device changes state (such as key press), the other part is notified. The navigation service takes care of navigation events, such as buttons pressed and triggering widget callback functions. The service category supports advanced services such as broadcast and content awareness. There's not yet much information about this service available. Lastly, the widget category contains widget specific messages. These messages are passed from the widget to the device and are restricted to 2048 bytes, including header and end tags.

There's not much information published about this WDK version since it's only available as a private beta. According to Yahoo!, more information will be published when the WDK has been released, second half of 2011.

The inaccessibility of any material whatsoever (except the white paper) has made it impossible to perform a fair evaluation.

5.1.2 Samsung TV remote app

Although not much information has been provided, there is to be an app for iOS developed by Samsung available on the Apple App Store. An entry at the blog "Samsung Hub"[36], states that Samsung has published an application acting as a TV remote, communicating with the TV through the LAN connection. Supported models are the ones with the Internet@TV functionality, released in 2010 or later[37].

When starting the application, it provides a demo view where the controls can be interacted with without requiring access to a TV. There are six views available. The first one is called "TV Remote" and contains the same buttons as a regular remote. Secondly, a view called "Gesture Remote" utilizes the accelerometer to adjust volume and change channel. By tilting from side to side, the volume will increase or decrease. The channel number is increased or decreased by tilting backwards and forward. Other views are an on-screen keyboard and a gamepad equipped with the buttons that are used when playing games.

Pairing is carried out in about the same way as for the Yahoo! Connected TV-kit. The application locates the TV by using local network discovery. During the discovery process, each TV compatible with remote controlling is listed in the application. When the user selects a desired TV to control, a popup appears on the TV, asking to grant access to the device. The device will add the device to its Access Control List (ACL) for future reference. Whenever the device attempts to reconnect, the TV will verify whether the device is allowed or denied to connect, according to its ACL.

Since Samsung hasn't made any material, such as manuals and whatnot, available on their website, we can assume that this application is created for demonstration purposes and not meant to be widely used. The description of the application provided above is strictly based on testing and the author's conclusions. When testing, problems with the connection between the TV and the application were experienced. Unforeseen disconnections did occur every now and then, sometimes with the result of not being able to reconnect.

If the intention of the application is to be a proof-of-concept, it might be possible to further develop the source code in order to create a framework for interaction with Samsungs' devices. As there's no information to access, it's impossible to make this judgement.

5.1.3 Conclusion

The Yahoo! Connected TV module and the remote application from Samsung can be regarded as the most notable solutions for interaction between the TV and mobile devices. As Yahoo!'s framework seems to be something similar to the ecosystem that is to be developed in this thesis project, it is the one most suitable among those two. However, it's unknown whether the framework is Yahoo! Connected TV specific or have the feature of being used by other application frameworks as well.

5.2 Interaction between hand held devices and IPTV-apps

This project is aimed at defining the possibilities to somehow communicate with an IPTV application by using one's hand held device. The user should be able to utilize the technical features of the smartphone to direct the IPTV-app. With the limitation of not transmitting binary data in mind, the types of instructions passed back and forth to the user can be established.

A key aspect is to determine whether the application logic will be handled by the IPTV-app or the mobile device. Utilizing the first-named will result in a centralized model, which maintains a correct state of the session, is required to carry out all processing (communication-wise). The clients would only have to send and receive instructions, based on the result of the processing at the IPTV-app. If the client devices were to take responsibility for the application logic, the data processing would have the possibility to be distributed and only carried out at each affected device, thus relieving the IPTV-app from data processing. A distributed system, however, requires all devices to be synchronized and having agreed on being at the same state. By taking these facts into account, it has been concluded that centralizing the application logic would require less work and sufficient for this project.

5.2.1 Analysis of applications able to provide interaction

The following list contains examples of interaction with existing, and theoretical, IPTV-apps and handheld devices. In conclusion, it's possible to offer a rich interaction by sending instructions on how the IPTV-apps should behave with regards to controlling a game and replacing the shared remote in multiplayer sessions.

The list is only for illustrative purposes. The listed applications won't necessary be prototyped in a later phase.

Quizz Master

Quizz Master is a quiz, developed by Accedo Broadband, supporting one to four players. The procedure of the game is described in the following steps:

1. When starting, the number of players is entered by a user.
2. All player names are entered by moving a cursor among available letters.
3. The game is started and the first player is to choose the correct answer to the displayed question. Four alternatives are given.
4. When the answer has been given, a new view with information about the correctness of the answer is shown.
5. The next player is up, who repeats the procedure the first player went through, but answering a different question.
6. When every player has answered ten question, the game ends and the high-score chart is displayed.

Given that every player has a smartphone, it's possible to reduce the amount of tedious work of entering player names and passing the remote around. The game is started by choosing the number of players using the remote. Every player is prompted on his/her smartphones to enter the name. When all players have finished this step, the game is started. Each round starts in the same manner as the original version, a question along with four answers is displayed. The player who's up submits the answer by pressing one of four buttons on his/her smartphone. Next screen with the correct answer is shown when the IPTV-app has received the submitted answer. The next player will be prompted to signal that he/she is ready by pressing a "ready"-button. The game progresses until all questions have been answered.

Black Jack

This applications is developed by Accedo Broadband as well. It simulates the card game Black Jack. The rules for the game are available at `http://www.blackjackinfo.com/blackjack-rules.php#a21`. The concepts talked about in the next paragraph are explained at the website.

The player has access to, apart from the table, a group of buttons that will control the game. The buttons are related to game instructions, such as “Stand”, “Double”, “Hit” and “Deal”. These buttons might as well be removed from the screen and be shown on the smartphone instead. Apart from the buttons, the player’s cards can be displayed as well. This allows for a multiplayer session where each player controls its hand via his/her handheld device.

Max the Hungry Monkey

This game developed by Icareus has the objective to maneuver a monkey in the direction of fruits falling from the sky. The monkey, standing on a 2D platform, moves right and left when the user presses left and right button on the TV remote. In this case, the accelerometer or touch screen of the smartphone come handy. The accelerometer makes it possible to tilt the smartphone in order to steer the monkey. Using the touch screen will enable for moving the finger in the same direction as desired to move the monkey.

Youtube

A suggested application is one with the intention of displaying videoclips from Youtube that have been selected using the smartphone. The user searches for the clip using the smartphone. When the desired clip has been found among the hits, the user selects it for being submitted to the IPTV-app. Since the smartphone has a more convenient user interface for browsing a search list, but the TV will be more efficient at showing something to a large audience, the devices will complement each other.

Discussing ongoing shows

Utilizing the handheld device’s keyboard, one is able to offer an application where the audience is allowed to contribute to a debate about the ongoing show. The contributions will be displayed in a ticker, bordering the show. This functionality may be extended by creating an audience response system. If there will be a votum held, all the viewer is required to do is submit the answer through the handheld device.

5.2.2 Proposed instructions

Keeping the proposed functions in the previous chapter in mind, one is able to derive a set of instructions that will be passed between the IPTV-app and the handheld device. The set is describe in table 1.

Description	Functionality at device
Move object (up, right, down, left)	Swipe, accelerometer and button
Enter text	Keyboard
Use the menu	Button
Continuous movement	Swipe and accelerometer

Table 1: List of instructions derived from the application study.

5.3 System architecture

When it comes to requirements on the system architecture, the main objective to take into account is to seamlessly be able to pass messages back and forth between the mobile device and the IPTV application. Since the latter is a web application and thus not supporting sockets, other ways of pushing messages need to be established. Seamlessly in this sense includes properties as a relatively well defined and easily interpretable protocol and functionality allowing for passing messages at a high abstraction layer (JavaScript at IPTV application).

5.3.1 IPTV application

As Accedo highly regards the web browser platform as the undoubtedly most widely used, there's no point in targeting other platforms. Another key aspect to the exclusion of platforms is the restricted time-frame.

The advanced functionality of the web browsers extend to JavaScript. There's no low-level socket handling, as in the case of mobile applications. To achieve bi-directional communication with the server, the application must rely on the provided features of JavaScript. As of today, Connected TVs manufactured by the major companies do support XMLHttpRequest objects. This is about as far as the technology reach in complexity. To be able to respond to the requests, the other side must allow for HTTP connections (i.e. web server).

5.3.2 Server

Since a web application requires a HTTP server for communication, the ecosystem requires this kind of server. It can either be constituted by the mobile application or a separate actor. If the platform running the IPTV application and the mobile device are connected to the same network, it allows for low latency communication, when considering the first option. The drawback of this decision is that each application is required to run a complete web server, supporting the set of functions needed for the ecosystem to run. The web server might be quite rigorous, thus resulting in a heavy workload on the developer. It also brings up questions such as how a multiplayer environment is set up. The second alternative, running a standalone web server, will require less coding of the mobile application, no need to establish a discovery service (since the address of the server always is known) and facilitate for interaction with several users in one session. Accedo will also be able to implement a usage monitor and push updates without requiring users to update their mobile applications. The drawback will be increased latency, compared to a local network. For this ecosystem, it's concluded that a stand alone server is the preferred alternative. The main reason is to be able to focus on the framework rather than implementing a web server.

The server software must support different kinds of web connections and preferably be able to alternate between those depending on the support at client side. Since HTTP messaging requires a handshake session for each new message sent, it would also be nice to support regular sockets, to decrease the amount of transferred data.

5.3.3 Mobile application

The mobile application is the application ran on the smart phone, by request on the user. Its purpose is to somehow interact with the IPTV application by sending and receiving instructions. The user utilizes the features of the smart phones by touching the screen, tilting, moving around, etcetera. The application parses the users commands and sends the instructions to specified clients. Since the IPTV applications will require different ways of interaction, each must have a corresponding specific mobile application.

As opposed to a web browser environment, native mobile applications are compiled before being deployed to the running environment. As Objective-C and Java are rather developed languages, they support sockets and HTTP messaging. Using sockets facilitates for avoiding the communication overhead that is present in HTTP messaging. One can argue that all messages sent from the mobile application to the server are relayed to the IPTV application and thus experiencing a bottle-neck in the server the IPTV application communication. However, the mobile application would in the best of worlds only require a 3G connection, which has a rather high latency and varying speed.

5.4 Server and client communication

Below is a list of possible ways of carrying out communication between the client and the server. The list is ordered by the desirability in descending order. The definition of desirability is connection speed/latency, amount of overhead data and connection maintainability.

5.4.1 Internet socket

An Internet socket, or network socket, constitutes an end-point for communication between two devices over an IP-based network. It provides a two-way communication path, behaving differently depending on the socket type. The operating system provides the network communication through a file handle to each process. The behavior of a socket is described by the reliability of the transfer, whether the data is duplicated and/or sequenced and maximum data length[38]. The data is either transferred via TCP or UDP. For this project, we only consider stream sockets, since we want the messages to be sent in sequence, arrive in a certain order and utilize the bi-directional property. Due to the requirement on reliability, the stream socket is always implemented using TCP[39].

5.4.2 WebSocket

The WebSocket protocol is a newly designed protocol which enables a web browser to take full advantage of Internet sockets. By using the WebSocket API, it's able to set up a socket, supporting bi-directional communication, and connect to a WebSocket server. The connection is maintained throughout the session, relieving the browser from having to re-establish the connection each time it's passing or receiving data (see AJAX Long Polling). By maintaining the connection, only being required to pass the actual data, no overhead information, such as HTTP headers, is sent and therefore relieving the network load. It should be noted that this kind of socket is implemented in a layer on top of the web browsers layer, which in turn uses regular Internet sockets.

The first draft of the WebSocket protocol specification was published by IETF in January 9 2009[40]. As of today, WebSockets aren't fully supported by all major web browsers. Both Opera and Mozilla Firefox are concerned with the security of WebSockets[41] and have therefore disabled the feature, while waiting for a new version. The flaw discovered gives hackers opportunity to poison proxy servers' cache, by sending false HTTP headers and thus able to inject malicious JavaScript code[42].

5.4.3 The Comet model

The Comet model is an umbrella term for different solutions, predecessor to WebSockets, used to push and stream data to the web browser. It includes a lot of different techniques, all of which are mainly implemented using JavaScript. There's a wider support among browsers for the Comet model than WebSockets.

The model is implemented in a variety of different ways. The entities of the domain are the server and the client. The server side is made up of a single HTTP server (Apache (`http://httpd.apache.org/`), Jetty (`http://jetty.codehaus.org/jetty/`)) or a HTTP server running alongside with a comet module (Apache and Cometd (`http://cometd.org/`)). While the Apache web server isn't natively optimized for a huge load of requests, Cometd and Jetty are. By utilizing Java servlets, they handle requests asynchronously and therefor reduce the amount of unnecessarily allocated resources.

On client side, there are a couple of JavaScript solutions to choose from. As Cometd and Jetty have their own libraries, none of which used in this project, the methods are explained using generic code. The most popular, and efficient, way of pushing and pulling data is by using AJAX (Asynchronous JavaScript and XML). The browser registers an XMLHttpRequest object (XHR) with the Document Object Model (DOM) and sends request to the web server through this object. All requests are asynchronous, which lets the browser request information when the web page is loaded and publish whenever it's ready. Another benefit is that the entire web site doesn't need to be reloaded when the user requests new data. It can be noted that the data transferred doesn't necessarily need to be eXtensible Markup Language (XML) formatted. Many developers have chosen to replace the XML response with JavaScript Object Notation formatted data.

The main drawback is that a new connection need to be established for each request. The connection isn't maintained when the server has responded with the result. The client and server therefor need to negotiate a new connection passing HTTP headers back and forth. Overhead data is applied for each message and thus increasing network load. The size of a header is about 430 bytes when sending an XHR [1]. Another problem is the security feature "same origin policy". A JavaScript is only allowed to request resources from the same server as from which it was loaded. The origin is defined by the protocol and the domain. A web server running different subdomains may circumvent the problem by modifying the DOM's

[1] Tested at local web server. Philips: 480 bytes, LG 380 bytes

property `document.domain`. Setting it to the common domain among the subdomains (`a.foo.bar` and `b.foo.bar` becomes `foo.bar`) will enable for each subdomain to load data from each other.

The benefit of being able to publish whenever ready can be utilized to continuously request new data from the server. Whenever data is delivered, the script can instantly create a new request. This is called *Ajax Long Polling*. However, as more and more users request data, the server will be suffer from a high load of idle connections. Each connection requires some memory allocation and a socket.

```
/**
 * Function utilizing Ajax Long Polling to mimic receiving a
 stream of data.
 * Written using the jQuery library to illustrate the most
 important parts without involving customized cross-browser
 compatible code.
 */
function longPoll()
{
    jQuery.ajax({
        type: "GET",
        url: "resource.php",
        cache: false,
        timeout: 50000,
        success: function(data)
        {
            // Do something with received data

            // When finished, redo the request
            setTimeout('longPoll()', 1);
        },
        error: function(jqXHR, textStatur, errorThrown)
        {
            // Error occured, possibly a timeout
            // We retry the request
            setTimeout('longPoll()', 1);
        }
    });
};
```

Browsers that don't support XHR can utilize the *Forever iFrame* method. In the HTTP protocol, the length of data to receive doesn't need to be specified if the attribute `Transfer-encoding` is set to `chunked`. This allows for continuously filling the header with data. The web page loads an invisible iFrame with the source attribute set to a document that handles message receiving. As a new message will be passed to the client, the HTTP header is appended with the message, encapsulated in JavaScript code. By doing the encapsulation, the iFrame receives the code for handling the message. The benefit is that it's fairly easy to implement and supported by almost every browser. On the contrary, the browser can't natively detect the state of the transfer nor the amount of remaining data to transmit.

Browser not supporting XHR or web pages requiring cross domain-request (wants to avoid the "same-origin-policy") have the possibility to fall back to JSON with Padding (JSONP). The requested resource is loaded with a script-tag in the DOM, where the `src` attribute is the resource URL, with a callback ("padding") function in the querystring. In contrary with the regular XHR response where the client exclusively receives an XML or JSON object, the resource responds with the data wrapped in the callback function. As previously mentioned, the request can be issued to any domain, which is a great benefit when using third-party APIs. The major drawback, however, is the possibility for a malicious script to inject <u>any</u> code to the script-tag. The resource loaded must be fully trusted. In case of a malicious script is called, it can steal the site's cookies, redirect users or interfere with the DOM. As JSONP makes all requests through the querystring, there's no possibility to use other methods, such as POST.

```
<script type="text/javascript">
function getResponse(response)
{
    alert(response);
}

function requestData()
{
   // Define the resource URL
   var url = "http://foo.bar./taz.script?callback=
   getResponse";

   // Create a script element in the DOM
   var script = document.createElement('script');
   script.setAttribute('src', url);

   // Load the script
   document.getElementsByTagName('head')[0].appendChild(
   script);
}
</script>
```

Figure 2: Example of JavaScript doing a JSONP request

```
getResponse("This is a response");
```

Figure 3: Example of response (taz.script)

5.4.4 Conclusion

Among the listed technologies, some are more to prefer than others. The problem however, is that STB:s and Connected TVs don't support sockets, nor WebSockets. It's desirable to prepare for future support by using a communication library that conforms to the supported methods. The priority list must take some attributes into account. Security is a key feature that must be accommodated to. Overhead data is an important factor that influences the load on the server. Bandwidth consumption would benefit if it could be kept to a minimum.

5.5 Server software

The proposed solution for communication between the IPTV and mobile application will be an architecture with the server being a standalone unit, mediating all messages. As previously declared, implementing an HTTP server at mobile side would require tedious work, focus on a secure implementation and less time allocation for other parts. Possible solutions would be using server software allowing for either HTTP messaging and socket connections or just HTTP messaging. The software alternatives that analyzed in this project are Ajax Push Engine (APE) and node.js.

5.5.1 Ajax Push Engine

APE is a package, developed by Weeyla, including a comet server and a client side JavaScript framework. It's not indented for replacing a regular web server. Its only purpose is to work as a messaging platform for web services. It runs asynchronously, using event handlers for receiving messages at client and server side.

The comet server is a command line application written in C. The perk of this server software is that the server module is scripted with JavaScript. It's running the JavaScript engine Spidermonkey, the Mozilla foundation's C implementation of JavaScript, equipped with Mozilla's native-code compiler TraceMonkey. This facilitates for reusing code for some layers at the IPTV application. It's not restricted to HTTP messaging as it also features a socket API.

The JavaScript framework is loaded at client side (in this case the IPTV application) and provides the necessary messaging functions. Since different web browser support different ways of communication, the framework automatically determines what is the most appropriate technique for each browser to use. The highest ranked feature is Websockets and on the other end long polling. The third option is XHRStreaming, which falls between the other two. If the request will be made cross-domain, APE falls back to JSONP.

The system is quite well documented at the project page. Although the wiki is quite hard to grasp at first (it's quite tricky to find the content one is looking for since it's not well structured) it's exhaustive and provides all information necessary for implementing one's own server modules. The problem followed by the lack of structure is that a newcomer to APE will need some time to familiarize with the documentation. However, that's often the case for most software.

APE is released open source licensed under GPL. It's currently supported by Linux and Mac OS X. As most web servers at the largest web sites run Apache[43], it can be assumed that the services are hosted in a UNIX environment, thus being able to run APE.

5.5.2 Node extended with Socket.IO

Node is an event-driven framework based on Google's JavaScript engine V8. It's purely intended to be used for network applications. The main subject that Node addresses is scalability and performance. Just like APE, it doesn't replace a regular web server. Depending on implementation of the server module, it can act as a web server or a stand-alone messaging server.

As opposed to APE, Node doesn't come with a package for clients and servers. It's main purpose isn't to facilitate for writing comet environments, but rather any network application. This fact implies that it requires a little more work to reach APE's starting point. However, it does offer the developer to make more design decisions and optimize the code for the project specific environment. One example would be removing the overhead data that APE adds for each message packet. On the contrary, one can argue that APE is already as optimized as possible since it's only handling one kind of network service.

To extend Node with the web service communication feature there's a library called Socket.IO available. Just like APE, it facilitates for web browsers to load a communication framework. It's also event triggered and chooses the most appropriate method among the ones supported by each browser. By supporting Adobe Flash Socket and Forever iFrame, there are more techniques to choose from compared to the ones offered by APE. The client libraries can be bundled with the web application, not needing to be loaded from the server, as opposed to APE.

Just like APE, Node is scripted in JavaScript. It has an extensive API with support for query strings, stream sockets, HTTP communication, event listeners, and such. It runs an event loop as soon as the server starts and stops it when there are no more events. The model is based on the web browsers event model, where the developer doesn't need to bother about starting and stopping the loop.

Node is licensed under the MIT license. It does, however, rely on several external libraries, as V8, c-ares and libev. Those libraries are licensed under different licenses. Most common are BSD, GPL2 and MIT. Node is distributed open source, available for download from Github. It mainly supports Solaris, UNIX and Mac OS, but can also be compiled for Windows and other operating systems.

5.6 IPTV applications

IPTV-apps are applications and widgets ran on a set-top-box or a Connected TV. The architecture varies between the different manufacturers. The majority of those choose to follow a standardized solution, such as web and Flash applications[2]. Other chooses to have their own solutions. Microsoft's Mediaroom platform is supported among some STB:s and also Microsoft's products (smartphones and Xbox360)[44].

It is heavily regarded that the majority of today's modern STB:s and Connected TVs use an environment where the applications are expressed in the form of web applications (Accedo Broadband). A web application is an application that is accessible through a web browser. The popularity of such application can be derived from the following facts:

- The browser is a thin client. The users don't have to install the application and will be able to use it from whichever computer.
- The developer won't have to distribute new versions to the users as all that is required is uploading the new version to the web server.
- Regardless of platform, the browsers behave according to a standard. This removes the need to port the application to different operating systems and hardware.

5.6.1 Web applications

In the IPTV context, a web application is considered a website, constituted by a single or multiple HTML documents, driven by JavaScript and styled by CSS. Calls to external web services' API:s are sent through HTTP requests using AJAX.

As web applications runs in a web browser, which resides in an operating system, the performance won't be as good as for a native application, directly ran in the operating system. The fact that JavaScript is a scripting language also limits the performance of the application. If it were compiled, there wouldn't be as high runtime overhead. The tradeoff however is the relatively low requirements of memory handling and compiling time. JavaScript runs its own garbage collector, relieving for the developer to take

[2]Web applications might as well be constituted by Java or Flash. In this context, such applications are run in an engine separate from the browser.

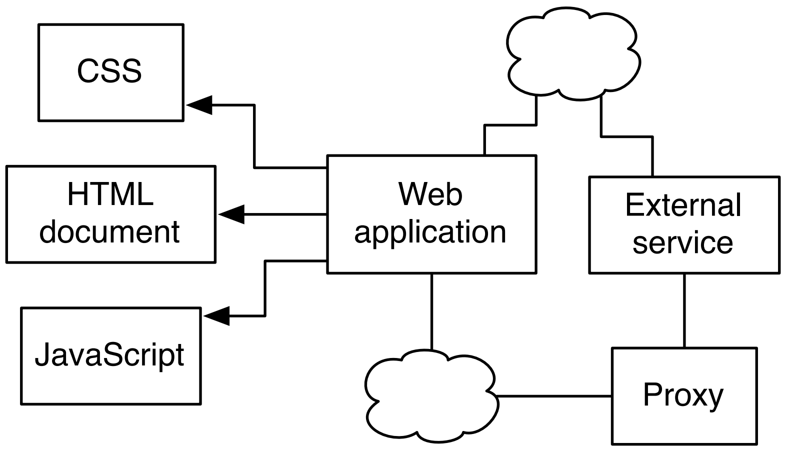


Figure 4: The components of a web application

memory management into account, which can be tedious work for heavy applications. As the scripts run in the browser, the developer can test the code by refreshing the the browser. Additionally, there's no, or very little, need to deploy and compile.

The markup language defining the contents of a web site is called *HyperText Markup Language* (HTML). Every element in the code defines an object, which in turn is interpreted by a web browser. The elements are building bricks that only can display static content.

The application would be rather static if it was only to be defined by the HTML. By modifying the markup's associated Document Object Model (DOM) using *JavaScript* (JS), the dynamism will be concretized. JavaScript is an object oriented scripting language that's been coupled with web sites with aim to enhance the interactivity. Apart from redrawing and modifying the DOM, IPTV-apps also use JS to catch input from the remote control. It's consequently JS that's allowing the user to operate the IPTV-app as desired.

Cascading Style Sheets (CSS) is the language of today that's used to define the appearance of the web site. By declaring class and id attributes in the markup, the CSS will define the specific appearance for each element. CSS can be expressed in a separate file or by modifying the DOM (using JS). The latter has its caveat of being able to dynamically change the custom appearance. For instance, the background image of a DIV-element defines the appearance of a character in a platform based game. As the JS catches the input from the remote, the character moves accordingly by changing the position of the DIV. At the same time, JS changes the background image to one that symbolizes a moving character.

The *rendering engines* available and used by STB:s and Connected TV:s are based on prominent engines. The HTTP request sent to a web server contains a header with a User-agent string, which is used to identify the engine. Some of those are describe in table 2. The problem with the claimed user agents is that they're not what they turn out to be. The developers have chosen to extend the functionality at the JavaScript engine with the feature of supporting platform specific features. Since the documentation of the API:s is confidential, those functions can only be described and not quoted. Generally, the functions are grouped into two sets. The first set contains hardware specific functions not directly related to the web application. Examples of such are opening menus for adjusting the volume, upgrading the firmware at the box, play a video stream with the built-in videoplayer and showing the browsers loading icon. The other category provides help functions, easing the developer from tedious work such as creating correct XmlHttpRequests or outputting text in specific containers.

Platform	Rendering engine	JavaScript engine
LG NetCast	Webkit 528.6+	V8 $>$ 1.6
Samsung Smart TV	Maple Browser 1.5	Maple Browser 1.45
Philips Net TV	Opera 9.70	Presto 2.5.33

Table 2: Sample of web engines

There are three cases when the application won't be able to carry out a regular AJAX request:

1. The domain at which the requested resource is located isn't the same as the application's host.

2. The response is of a structure containing more information than necessary. In this case, the data need to be filtered before JS begins its processing. The platforms at the STB:s and Connected TV:s are rather slow and will require lots of processing time.

3. The platform at which the application resides, isn't capable of formatting the request message according to the API.

The solution to this problem is to write a proxy through which all calls are sent. This proxy is a web service, custom tailored to every application, able to communicate with the external web server and the web application.

5.7 Mobile application development

As previously mentioned, mobile applications for iOS and Android are separately developed in Objective-C and Java. The languages do come, as always, with their pros and cons. To be able to reach as broad market as possible, the developer has to go through the programming process twice. This simple fact is a very central issue for mobile application development. In the case of trying to bring as many people as possible to interact with the IPTV application, Accedo would prefer to explore and assess what is the most suitable way of development. The more people in the party being able to join the game, the greater experience they will get. None should be left out, because one's phone, or tablet, is Android or iOS based.

The main requirement at Accedo is the possibility to reach a broad market of smartphone users and as well as reaching as many developers as possible. Development for Android and iOS, in a framework that most developers are familiar with, is therefor preferred.

5.7.1 Alternative development tools

As alternatives to regular application development, there's the option create cross-platform applications. This option is constituted by two sets of tools. The common denominator is the possibility to wrap the compiled code and make it appear as a native application. This won't require the platform specific knowledge which is needed when producing native applications. The first set is constituted by the web application frameworks where the distributor can choose from either publishing the applications in the regular app stores or on the web (as web applications). The second set consists of frameworks only being able to be cross-compiled into native applications. The key factor that must be taken into account when choosing tool is whether the app stores recognizes the applications for publishing.

The tools are equipped with several positive properties. As each tool uses a unified language, the platform knowledge is reduced from two to one. An obvious implication is the reduction of time required. If the chosen tool is intended for web application creation, there are several ways of distributing the final version. If the development process is incremental where each step results in a new generation of a prototype, the final version can easily be beta tested as a web application and finally wrapped into a native app. Without much knowledge on how much functionality each browser support at each device, one can assume that low functionality requirements open to a wider range of compatible devices.

On the contrary, only one of the investigated tools supports the native look and feel of iOS and Android. The native SDKs provided by Apple and Google do offer functionality for creating the user interface (UI) programmatically or using an interface builder. Each element in the UI come with a predefined layout, which the developer may modify if desired, but doesn't necessarily have to. Secondly, there's a great lack of hardware integration. Most web browser's don't support an API for accessing the hardware features of the devices. As it differs from browser to browser, and the web applications desire to support as wide range of browsers as possible, the utilization of the capabilities at the smartphones won't be as high as wished.

Tools

Here follows a list of popular tools and frameworks. The majority utilize the feature of being able to wrap a web application into a native application. Among the acquired benefits, the native web browsers tend to have a rich JavaScript API, enabling for hardware communication.

Sencha Touch is a JavaScript framework developed by Sencha with the purpose of developing mobile web applications. It's free to use, licensed under GPL v3, but does require a payment in case support is needed. The applications are written in HTML, JavaScript and CSS. User interface libraries to make the UI appear as native do exist.

Phonegap is a framework for building and deploying cross-platform applications. Just as with Sencha Touch, the development is carried out by web programming. The main difference is that Phonegap provides a template to be installed into the SDK, thus having the application to be deployed on the device as a native application, rather than a web application. Its JavaScript libraries have support for hardware integration. Although it's supposed to be experienced as a native app, with the benefit of only being required to know web programming, the native UI is not utilized.

Another framework for building native apps is the Ruby-based framework **Rhodes**. Just like Phonegap, the applications are originally web applications, cross-compiled for deployment to smartphones. According to the website, it does utilize a lot of hardware features at the devices. However, it does lack native support for the rudimentary functions such as tap, touch and swipe gestures. Those missing features can although be introduced by extending the application with the Sencha Touch framework.

The richest and most sophisticated tool is Appcelerator's **Titanium Mobile**. It's a cross platform framework for Android and iOS. Its outstanding properties are the wide support for hardware integration and high utilization of the devices features to natively run JavaScript ([45]).

This chart presents the support for the features required to create a rich application that interacts with the IPTV applications.

Feature	Sencha Touch	Phonegap	Rhodes	Titanium Mobile
Accelerometer	No	Yes	No	Yes
Compass	No	Yes	No	No
Gyroscope	No	No	No	No
Tap	Yes	Yes	Yes	Yes
Multitouch	Yes	Yes	Yes	Yes
Pinch	Yes	Yes	Yes	Yes
Scroll	No	Yes	Yes	Yes
Discrete movement	Yes	Yes	Yes	Yes
Continuous movement	Yes	Yes	Yes	Yes
File transfer	No	No	No	No
Network data transfer	Yes	Yes	Yes	Yes
GPS	No	Yes	No	Yes
Natively approved by App Store	No	Yes	Yes	Yes
Natively approved by Android Market	No	Yes	Yes	Yes
iOS look and feel	No	No	No	Yes
Android look and feel	No	No	No	Yes

Table 3: Chart of supported features

5.8 Prototype

Prior to designing the ecosystem, it was decided to develop a working throw-away prototype, which would act as proof of concept and verify properties on the connection, transmission of instructions and compatibility with some Connected TVs and STBs. Since the author was inexperienced with event based programming, the mobile platforms and IPTV applications, the prototype would also serve as a good starting point for learning how to develop applications for each target. The requirements on the prototype were the following:

Mobile application

1. The application should be developed for Android and iOS devices.
2. It should be developed in the native SDKs (Android SDK and Xcode).
3. The application should support the following input features:.
   (a) Touch/tap: When the screen is touched with one finger, it should display a symbol at the touched location, and output the coordinates.
   (b) Sweeping movement: When the screen is touched with one finger, which in turn is moved while still touching, it should continuously present a symbol at the finger's location.
   (c) Same as 3.3.b, but with support for at least two fingers.
4. The features described in 3. are to be presented in separate views.
5. When the application receives input, an instruction with the following parameters should be transmitted to the IPTV application: `<instruction number, input type, coordinate(s)>`.
6. The communication with the IPTV application will be through the server, as described in section 5.3.2
7. It should store the timestamp in each received acknowledgment packet from the IPTV application for statistics.

IPTV application

1. The application should be developed with support for WebKit based browsers.
2. It should be able to communicate with the mobile application through the server.
3. It should act on instructions sent from the mobile applications by drawing a rectangle at each coordinate.
4. When requirement 3. is carried out, it should send an acknowledgement, with a UNIX timestamp, to the mobile application: `<instruction number, ‘‘acknowledgment’’, timestamp>`.

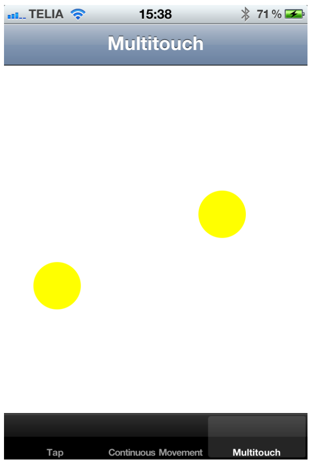


Figure 5: iPhone application prototype

Server

1. The server should be implemented in either APE or node.js.
2. It should allow for web and socket connections.
3. It should broadcast each received message.

5.8.1 Result

The prototype ecosystem worked as intended. The server and IPTV application was at first run on the author's computer, using the Chrome web browser to simulated an STB device. A slight delay (approximately 100ms) was noted when the IPTV application was run on a development box for LG devices. This was unexpected, as the representative of Accedo Broadband assumed the delay to be significantly larger.

As the socket connection between server and client was of type streaming, each message arrived in the correct order (which could be verified by outputting the instruction number in the console). There was no track of whether the delay between when message m_0 and m_1 were sent, was the same as the delay between arrival at the receiving side.

Given the result that the prototype worked as expected (with regards to functionality), it was decided to proceed with designing the ecosystem.

6 Design choices

6.1 Mobile application development

If the developer desires, it seems possible to reside to a unified framework for developing applications aimed at Android and iOS devices. Although some features aren't supported by all frameworks, one can choose the most suitable one based on the application's finished requirement specification.

The most appealing framework among the alternative ones is Titanium Mobile. It's the one with support for the most number of listed features. The big plus is the support for native look and feel at Android and iOS, which is a quite important factor for making the application appear as genuine as possible.

A problem though is the lack of information on what is the most popular framework. Any numbers comparing the listed frameworks with the native SDKs haven't been found. One can assume that native development is the most prominent, as of today. Although almost every developer at Accedo posses the knowledge in HTML, JavaScript and CSS, it's decided to go along with the native development tools. The main argument is to be able to utilize the devices' resources the most. Since the applications created by the third-party tools are more or less wrapped in a web browser environment, there is overhead computation time for rendering and code interpretation added.

6.2 Communication

Since studied server frameworks actually offer communication through Internet sockets and HTTP, the client can choose the most desirable option. Mobile applications do support Internet sockets, which have lowest connection latency. IPTV-apps, on the other hand, aren't capable of handling sockets and must reside to the libraries offering HTTP communication.

6.2.1 Communication options

The tail end of how the clients, including smartphones as well as IPTV-apps, communicate is through Internet sockets. Unfortunately, as previously mentioned, IPTV-apps must rely on layers higher up in the communication stack and are therefore referred to libraries provided by the server environment.

Smartphone

A previously mentioned, Internet socket seem to be the most desirable way to go for establishing a connection between a mobile device and a server. By choosing a layer deep down in the communication stack (TCP), one can achieves low latency and doesn't have to rely on third-party libraries, but is able to achieve the desired functionality through the native functions at Android and iOS.

Alternatively, it's possible sending HTTP request through POST and GET using third-party libraries. To iOS, there's the library ASIHTTPRequest[46] and Android can utilize org.apache.http.client.HttpClient[47]. There's no purpose in using these, since they can't demonstrate any performance gains compared to the traditional Internet socket solution.

IPTV application

Since the IPTV applications studied in this project are web applications, one is referred to HTTP connections. Both APE and node.js have libraries for handling the connection (built-in, using Socket.IO, respectively). The libraries choose the least resource demanding option, of the ones supported by the IPTV-device's browser. The highest prioritized option is WebSockets, which maintains the connection throughout the session. The least preferred option is the Forevere iFrame.

6.2.2 Messages

The messages being passed between the actors of the environment must take the fact that the future might offer possibilities to send data through sockets, without layers in-between, into account. There's also the requirement on the message being sent as a datatype, supported by JavaScript, Java and Objective-C. Java and Objective-C offer the possibility to serialize objects (such as Java's Hashtable[48] and iOS' NSDictionary[49]). These datatypes may therefore be utilized to send a message representation. The question how the different operating systems will be able to interpret each others representation remains. Alternatively, the data is sent as a string with layout of a generic message object.

Considering the data formats supported by JavaScript, one has two options - to either use XML or JSON. The main difference between these two is the representation of data. XML has a data model more complex than JSON's. Each data entry is surrounded by tags (see code example in 6.2.2). Since packets transmitted to a web application already are loaded

```
{
   name    : "John Doe",
   birth   : 1970,
   city    : "New York",
   contact : {
      email : "john@doe.com",
      phone : "1-555-123-4567"
   },
   friends : ["Jane", "Joe"]
}
```

Figure 6: Example of JSON object

with HTTP headers, it's desirable to reduce the amount of transferred data as much as possible. With the large data overhead in mind, JSON is chosen as datatype for message representation. Third-party libraries for Objective-C[50] and Java[51] are available. Those libraries have built-in classes, enabling for easily creating and parsing JSON objects. Serialization is also supported at each library, thus letting the object be sent and ready to be parsed by the receiving side, regardless language or operating system. The drawback of using JSON is the lack of a standard to formally define a valid JSON-object. XML support the use of Document Type Definition (DTD) documents, that can be used by the receiver to verify the correctness of the received message.

6.3 Protocol

Since the clients of the ecosystem communicate with each other, they must be able to comprehend each others instructions, thus requiring rules for how to function. These rules all together form the protocol of the ecosystem.

6.3.1 Requirements

The requirements on the ecosystems are partly defined by the set of instructions that a mobile user will send to the IPTV-app and partly by the requisites necessary to be able to handle multiple clients, several instances of interaction and authenticate users with sessions. Some instructions (mainly the ones that replace the remote) sent from a mobile users are listed in Table 1.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<person>
   <name>John Doe</name>
   <birth>1970</birth>
   <city>New York</birth>
   <contact>
      <email>john@doe.com</email>
      <phone>1-555-123-4567</phone>
   </contact>
   <friends>
      <friend>Jane Doe</friend>
      <friend>Joe Doe</friend>
   </friends>
</person>
```

Figure 7: Example of same JSON object expressed in XML

Every instance of interaction that is created is called a “session”. To every session, there are one/many user(s) that interact with one single instance of an IPTV-app. The session must have the property of being uniquely identified. Each client (users and IPTV-app) should be able to be associated and disassociated with the session. In the event of a client knowing that it won't be able to respond to network communication for a while, it should be able to propagate its status. Examples of cases are when the IPTV-app loads external resources to set up the view, computes the next move in Chess, etcetera. Lastly, clients are required to be uniquely identified, regardless of membership of any session.

6.3.2 Lifetime of a session

A session is created when the user chooses to run an IPTV-app in mode with support for handheld devices. The IPTV-app connects to the server, asking for creation a new connection with a certain number of players. The server responds with the session number, which also acts as pairing key. As the key is shown on the TV screen, the users starts the mobile application and enters the key in the pairing view, that's shown upon start. The server receives the key. If they key is correct, the user is associated with the session, the IPTV-app receives a list of associated clients (including itself and the server) and the mobile client receives a message with informing that the key is accepted. The user also receives a “codeOk” message, where the *receiver* attribute holds the user id, as the only element.

The IPTV-app and the user progress by sending application specific instructions to each other. In the case of Quizz Master, the users take turns answering questions and receive information about the process from the IPTV-app.

When the user wishes to end the session, the mobile application send a disconnect message. As the server receives the message, it removes the client from the session and notifies the other clients of the session about the disconnected client.

A UML diagram, describing the messages passed back and forth, is available in the appendix. See diagram in Appendix (section 11.2).

6.3.3 Messages

The structure of the messages that are being communicated in the ecosystem is rather primitive. They consist of common attributes and attributes specific to some instructions.

As a matter of fact, the only mandatory attributes are the id number of the message and the instruction. Every message sent must be equipped with a unique id. This is only unique at each client, meaning that the same id can occur multiple times in the ecosystem. The main point of the id is for the receiver to know in what order the messages were sent, thus knowing in what order to execute the instructions. In the case of high network load, messages might not be delivered and thus having to be resent. To know if the message was received and processed, the receiver can respond with an acknowledging message, attributed with the sent message's id. Secondly, the other mandatory attribute is the instruction. With no instruction, the message is rather superfluous.

In almost every case, the mandatory attributes are supplemented by some optional attributes. When looking at the overall message structure, they can be considered optional, but when it comes to specific instructions they might be required (depending on the application logic for each IPTV-app). The messages are in detail described in Appendix (section 11.3). Among the optional the attribute *session* identifies the session to which the message belongs. The server receives the message and delegates it to the correct session clients. If this attribute is left out, the message will be broadcasted to every client in the ecosystem. In the case of a client not being associated with a session but still sending a message (e.g. authenticating), the *receivers* attribute should be set to "0". This attribute has its counterpart *source*. Both contain a list of receivers and senders, respectively.

In most cases, the source is only one client. Receivers, however, can be multiple (all players in a multiplayer games, for instance) or just one single client (IPTV-app). The values in the list are each receiver's/sender's unique identifier. Fourthly, there's the *timestamp* attribute. Its datatype is an integer, representing POSIX time[52]. Whenever an application is required to execute the instructions with the same delay as inputted by the user, but arrived with a non-deterministic delay, this attribute becomes resourceful. An example of this case is when the the application wants to execute the instructions with the same delay as they were inputted by the user. In the event of an instruction which need to be executed before progressing, the attribute *requires-ack* is set to true. The receiver responds with an acknowledging message, notifying that the message has been received and the instruction successfully executed.

6.3.4 Server

The server is intended to act as mediator between the clients. All messages are sent through the server, maybe acted on and eventually passed on to clients specified by some rule or the message's receiver list. The messages shouldn't need to be altered by the server if being passed further on to the clients. For instance, the *disconnect* instruction, sent from a client, notifies the server that the client intentionally will disconnect from the session. The server will then disassociate the client with the session and distribute the same message to other clients, in order to make sure they take necessary actions (e.g. halts the running game).

There should be no specific application logic at the server. The server must support every kind of application, since there's no point in running one dedicated server per application at this early stage of the ecosystem. Therefore, only session and client handling, apart from mediating messages, will be carried out by the server.

Client handling

A client is either the IPTV-device or a user running a smartphone app. The server is responsible for assigning unique identifiers to each user. The id can be of any datatype, although a simple one (such as integer) is preferred. Other responsibilities are to take care of any state changes, such as disconnecting, connecting and changing from ready to not ready state (and contrariwise). The design is aimed for keeping a consistent status about each client and thus being able to notify other client about the status, without the need to repeatedly asking the targeted client. This property absolves the requirement to have the client in a mode where it's always ready to respond. If it's busy processing data and not able to answer to network requests, it won't be able to answer any inquiries about being ready either. Since this design doesn't cover timeouts, asking clients would then result in a forever waiting mode.

Session handling

The server takes full responsibility for maintaining the association of a session and the clients. It must therefore keep record of each connected client, the upper bound of number of clients and each session's unique identifier. The list of connected clients not only includes the mobile users, but also the IPTV-device and the server itself.

6.4 Library

In order to efficiently extend existing and applications under development, the solution will be to implement the properties of the ecosystem as a library. The aim is to have a highly modular library, where each part can be substituted without affecting the others. The developer should only be required to interact with the layer closest to the application. As development will occur for IPTV-apps and mobile application, the layouts should be as close to each other as possible. By achieving this, an IPTV application developer will easily be able to translocate to mobile application development.

The representative of Accedo Broadband wishes to use the Observer pattern[53] for layer communication, as it's very commonly used in web applications. The JavaScripts often register some functions for events occurring in the DOM (such as clicks, hovers and key presses). The layers will therefore be equipped with an event emitter, to which the closest overlaying layers will be registered. Whenever a message is received, an event is triggered at the listening layer and taking necessary actions.

6.4.1 Layers

The library is separated into multiple layers, with a one-to-one relationship between each. Each layer acts observer, subject or both. The Figure 6.4.1 describes how the layers are connected to each other.

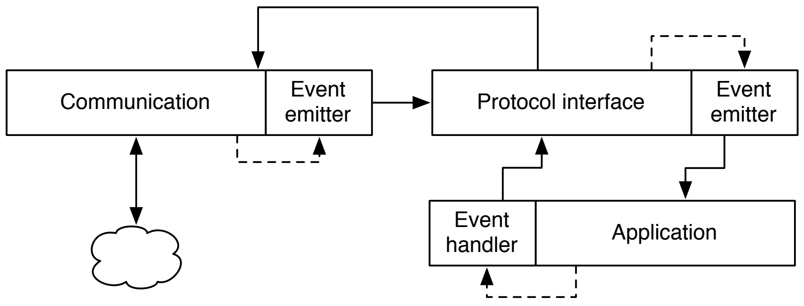


Figure 8: Overview of layers in a generic application

Communication

The communication layer is the connection to the OSI model's transport layer. It will open the bi-directional socket to the server. It's connected to the library's protocol interface layer, to which it propagates messages read on socket and from which it receives messages to send. The event emitter module extends the communication layer to enable it as a ConcreteSubject, allowing the protocol interface layer to listen to the events in Table 9.

Event	Triggering event
connected	Socket open and connected to server.
disconnected	Socket closed or disconnected from server.
gotmessage	Received message.

Figure 9: List of events listened to by the protocol interface.

Protocol interface

Next layer in the model is the protocol interface. Its main function is to filter the received messages and perform more specific callbacks. As the message is received from the communication layer, the protocol interface extracts the instruction and calls the registered observers methods. It also assures the maintainability of the connection by always listening to the *connected* and *disconnected* events. As the latter event is triggered, it emits the same event and also attempts to re-establish the connection. It's necessary to trigger the two events at this level as well since the listening application is responsible for the user interface and need to inform the user about the connection status.

Upon receiving a message, the layer is responsible for making sure that the listening layers are aware of the type of the message. This can be carried out in two ways. The first one is method overloading, as supported in Java[54]. Objective-C, however, does not support overloading[55], which adds the requirement of keeping record on the method callbacks. This also goes for JavaScript, which is a loosely typed language and isn't able to detect object types. The benefit of the first option is the slack off on the requirement on the developer to transform the received message into an object of a instruction specific class. It will be taken care of by the library.

Since this is the single layer connecting the application to the communication layer, it must provide the application layer an interface for sending message. This method should validate the messages tried to be sent by utilizing the protocol rules.

As the procedure for acknowledging messages is specified in the protocol, there's no need for the application layer to create those acknowledgments. The protocol interface layer should therefore supply a method callable by the application layer when the instruction has been executed.

The layer should have a method to generate a unique message id, to be tagged on to the message being sent. There should also be a method allowing for messages to be acknowledged.

With respect to the observer pattern, this layer takes role as observer and subject. It observes the events at the communication layer, while at the same time propagates messages to the application layer.

The events with which the application layer should be able to register callback methods are the ones presented in Table 10.

Event	Triggering event
swipe	Message with instruction “swipe” received.
continuous	Message with instruction “continuous” received.
tap	Message with instruction “tap” received.
string	Message with instruction “string” received.
disconnect	Message with instruction “disconnect” received.
ready	Message with instruction “ready” received.
notReady	Message with instruction “notReady” received.
code	Message with instruction “code” received.
ack	Message with instruction “ack” received.
codeOk	Message with instruction “codeOk” received.
invalidCode	Message with instruction “invalidCode” received.
invalidCmd	Message with instruction “invalidCmd” received.
clients	Message with instruction “clients” received.
customCmd	Message with instruction “customCmd” received.
disconnected	Disconnected from server.
connected	Connected to server.

Figure 10: List of events listened to by the application layer.

Application

This layer contains the application logic and shows the user interface. One of its responsibilities is to catch user input and transform it to an instruction in compliance to the protocol. It will also be responsible for observing the protocol interface layer and acting according to the received instructions. As this layer is specific for each application, there’s no need to specify detailed requirements, other than how to interact with the underlying interface.

Mandatory events to register with are the ones related to connection and pairing status. As the application connects and disconnects from the server, it must notify the user about the changes. It will also output information whether the user entered pairing key is correct or not. If the application involuntary disconnects from the server, it must re-authorize with the session by itself. This is carried out by re-using the pairing key submitted by the user.

7 Implementation

7.1 Libraries

Regarding libraries used by the client applications, they will share functionality, but be implemented in three different languages. As the smartphone either run iOS or Android, Objective-C and Java, respectively, are necessary. Along with these languages, JavaScript will be featured by the IPTV-apps. The language choice is based on the JavaScript libraries provided by the server frameworks.

The library layout follows the design in section 6.4.1. Three classes can be identified: Communication, Event Emitter and Protocol Interface. The Application and Event handler layers are to be specified by the developer at each application instance. Events occurring within the application are often being taken care of by built-in libraries (such as touch, swipe and button presses[56]).

7.1.1 Event emitter

The desired way to register with events is to pass a method reference to the emitter. After some investigation, it was determined that Java did lack support for an elegant way of referencing methods.

Objective-C has the selector conception, which supports passing references to the compiled methods[57]. By passing this reference, and the object on which it will be invoked, the event emitter acquires the information necessary for dispatching an event.

As for JavaScript, we can pass an anonymous method to be registered with the event emitter. Although the anonymous method might be defined as a class method, it can't refer to the context by using the `this` property. It must therefor refer to the object as a whole. This is seldom a problem, as most applications are driven by an engine (singleton), taking necessary actions on each object, determined by the input data.

The solution for Java utilizes a set of interfaces and an event emitter that identifies what interfaces are being implemented by the object. Every message in the protocol is a member of a set. The common denominator in the set is the metadata of the instruction. For instance, the instructions *ack*, *codeOk*, *invalidCode* and *invalidCmd* have the same metadata (either an id or a code). These messages will be members of the set *response* and thus there's an interface describing this message set. Each time an observer registers with the event emitter, the emitter will identify the implemented message interfaces. As an event is triggered, the emitter invokes the method `callBackForMessage(message)`, where the `message` object is an instance of the class `ResponseMessage`. The drawback of this solution is that the event, for each listening object, must identify the implemented interfaces at each object. This solution relies on the developer to identify the instruction in the set. It must perform a branch on the message's instruction property, thus requiring to ignore uninteresting instructions. By grouping instructions, several unnecessary callbacks might be performed, as the observer might only be interested in a subset of the group.

7.1.2 Messages

Since the messages are expressed as JSON objects, the datatype is natively supported by JavaScript. The developer isn't required to learn a new datatype. However, the JSON objects are supported by third-party libraries in iOS and Android. These libraries elegantly handle serialization and deserialization of objects. To maintain the abstraction of a message, not requiring knowledge about JSON, there is a set if message classes available. Each class represents a set of instructions, having getters and setters for each attribute. They inherit the generic class `Message`, which carries out the serialization and deserialization of the message objects. To send a message, the application only need to call the protocol interface's send-method with the message object. The interfaces guarantees the correctness of the JSON object before it's sent.

The caveat of using an internal representation of the JSON objects is that the classes can validate the integrity of the message. If the message doesn't conform to the protocol, they can throw an exception, notifying the developer that the message doesn't follow specification. As it is today, there's no validation performed when sending the message, since it's out of the scope of the thesis project.

7.1.3 Communication

What's most interesting to point out in the communication layer is how the messaging method is implemented. Apart from that, the layer behaves according to the design specification, where it triggers an event when the connection status changes and when receiving a message.

iOS and Android

The main requirement on the communication layer was that there should be no blocking when sending a message. The protocol layer should be able to pass a message to the layer, regardless of the socket being ready or not. As the messages received would trigger an event, it was also necessary to always monitor the status on the socket. By implementing the layer as a separate thread, it was possible to keep it always running and thus achieve the desired features.

As there is already a quite sophisticated library for Objective-C available, there was no need to write a new one. The used library was CocoaAsyncSocket[58], providing a class for asynchronously sending and receiving data (GCDAsync-Socket). The communication library was therefore constituted by this class. It fulfilled the necessary requirements. As it asynchronously received messages, it was able to trigger events at the protocol interface, without a intermediating communication class.

For the Java part, the communication class was implemented by hand. There might have been an suitable library available for download, but there was a curiosity on how to implement an efficient class. By extending the implementing the `Runnable` interface and creating a `run()` method, the object instantiated by this class would be threaded. The object runs an infinite loop, polling and pushing data on the socket. Each time the socket is available, it checks if there's any data available in it's local message buffer. This buffer is written to by the protocol interface layer. The loop is halted whenever the disconnect method is invoked, and started whenever the socket is prompted to be re-established.

JavaScript

When it comes to JavaScript, each client object at server side is coupled with a method used to send a message. All the server is required to do is identify whether the client was a socket or web client, in order to add the trailing `\r\n` characters. Those characters are used to mark the end of a message on a socket. At the web application, it instantiates a simulated socket object upon creating the connection. This socket object comes with a send-method, conveniently used for putting a message on the socket.

7.2 Server

The server properties are the same as the ones defined in section 6.3.4. It's implemented in APE as well as node.js. At first, it was only written for the APE framework, but as it later in the project was possible to allocate time for a node.js implementation as well, that was carried out. Since thhe framework runs a JavaScript engine, necessary modules would be written in the same languages as for the IPTV application. This opened up for reuse of some parts written for the client library. It was concluded, however, that only the event emitter was appropriate to reuse. The communication layer required support for socket and HTTP connections, which the existing library lacked. Protocol wise, there was no need to reuse the protocol interface layer, as the server wouldn't have to validate messages, nor listen to events other than incoming messages.

There's a little difference between the implementations of the modules at each framework. APE only requires one port to listen to, while node.js wants one port for socket listening and one for HTTP. This isn't much of a problem since callbacks are registered in the same manner at each framework. Secondly, the other main difference, implementation-wise, is the names of the methods used to register callback functions on socket and HTTP events. The communication part will be written as one class, specific for each framework, thus letting the other scripts be framework independent.

There are two catalogues maintained. One keeps track of connected users and the other of active sessions. A session is a JSON object with the attributes *code* (the session id, as well as pairing key), *numclients* (maximum number of clients, excluding IPTV-app and server), *hostid* (associated IPTV apps unique id), *clients* (list of associated clients). The list of active users also contain JSON objects, with the attributes *type* (connected via socket or HTTP) and *userobject* (framework specific representation of the user object). These catalogues must maintain the state of all connections and sessions. Therefore there are functions verifying users' session memberships, delete sessions and users, creates sessions and removes associations between users and sessions.

7.3 Proof-of-concept

In order to verify the fulfillments on the requirements on the ecosystem, the libraries were used to extend an existing application and create a sandbox application as well.

7.3.1 Extending existing application

It was decided that the most prioritized application to be extended was Quizz Master. The application was described in section 5.2.1. The reason why to extend this application was its rather non-complex code and ability to support a multiple players.

The necessary steps to take were to define the application specific messages, extend the application with the JS library and develop mobile applications for iOS and Android. Application specific messages would involve sending player names, submitting answers, notifying about message answered, instructing about next player, signaling when ready to get next question and notifying that the game is over. These messages would be embraced by the `customCmd` instruction, unless no appropriate instruction was available.

During the extension phase, the application was tested using a web based client throwaway prototype. The prototyping relieved the developer from writing a mobile application that required overhead work for creating a user interface. The prototype was quickly assembled and could be equipped with the same messaging functionality as the native mobile application. The phase was successful in showing bugs and faults in the server implementation. There was a lack of proper session handling for disconnecting users and allowing the correct number of users to pair with the session.

When the application was flawlessly interacting with the client prototype, the mobile applications were developed. This phase was successful in discovering bugs in the provided libraries. The JSON library for Java was erroneously documented and the event handler class for iOS didn't unregister callback methods in a proper way. Overall, the implementation was successful, resulting in working applications for each operating system.

7.3.2 Sandbox

As the multiplayer feature was demonstrated in the previous application, the sandbox application was to focus on other properties. As there was a desired to have a demo application utilizing the accelerometer, it was concluded to create a clone of the popular Pong game[59]. The game would support only one player, were the player's paddle was to be moved up and down by tilting the smartphone. Due to time constraints, only the iOS devices would support playing the game.

The game specific messages were only instructions about moving the paddle up and down. The data read from the accelerometer, for each axis, varied between -1.0 and 1.0. The value specified the angle in which the device was tilted. It was used to define the position of the paddle.

A requirement on the game was the upper bound on response time. The user must get immediate feedback on the movement of the paddle. If the paddle wouldn't move when directed to, the ball would be missed and there would be no point in playing. Since there was no methods available for verifying response time and managing the IPTV-app at low level, one would just have to rely on the connection speed and hope for the best. It actually turned out to work as intended when using a smartphone connected to a remote server via 3G and running the game on at STB by LG.

8 Result

8.1 Overall functionality of the ecosystem

The final version of the ecosystem fulfills all requirements aimed at in the goal definition. It includes client side APIs for Android and iPhone, supporting Android v2.2 and iOS 4.2+. The libraries have no external dependencies and are bundled to facilitate for developers. The APIs are only tested for the following devices. It's assumed that the APIs support other devices as well, as it's more a matter of operating system, rather than device, when considering Android and iOS.

OS	Version	Device
iOS	4.3	iPad 1
iOS	4.3	iPhone 4
Android	2.2	Samsung Galaxy Tab
Android	2.1	Software emulator

Table 4: List of tested devices and operating systems

The protocol is efficient and secure. The efficiency is constituted by the low amount of overhead data, as it's dealing with JSON objects. Since all messages are addressed to specific users (unless broadcasted), it's somewhat secure as other users can't receive messages of which they're not receivers.

The implementation of the client library for JavaScript is tested at WebKit and Mozilla browsers, which seem to fully support the functionality. It also relieves the developer from identifying a proper communication method and initializing the connection. It is all carried out in the library, when creating an instance of the protocol interface class. When new versions of the server frameworks are released and provide support for other connection mechanisms, it's intended that the library will still be fully functional. The reason why is the utilization of the provided communication library, by the frameworks, which is assumed to have the same API as today's versions.

The server implementation supports session handling and device pairing.

Each running server supports an arbitrary number of users, which depends on the requirements on responsiveness. Regardless of which IPTV application requests to create a session, the server provides the same functionality.

8.2 Server stress test

As it would be desirable to know how much load a server would be able to carry, before the response time reaches an upper bound, a stress test was performed. The test was designed to set up a number of sessions, where each session included one IPTV-app client and a number of mobile users. Each IPTV-app would request the server to create a session, receive the pairing key and distribute it to the mobile users. The latter connected to the server, paired with the session and began sending messages with the swipe-instruction. To somehow mimic a user-scenario, the messages would be sent with a random delay between 1 and 5 seconds. Assume the “user” was swiping through a list of video clips or search hits. The server software was running on a remote server, which was responding to pings with an average time of 2.2ms. The server hardware was constituted by a Xeon 3050 @ 2.13 GHz 2MB CPU, 1GB memory and running Linux Debian 4.1 as main OS.

The timing was to be measured by tagging each message with the current timestamp and the *requires-ack* attribute set to `true`. As the messages arrived at the simulated IPTV-app, the time taken between sending and receiving would be reported to a shared list. As each acknowledgment was received at the mobile device, the time taken going from sending to arrival state was to be reported to another shared list. The mean value of both lists would be printed and thus reporting the average single trip time. The round trip time was not interesting, as it was desired to establish whether the messages were delivered faster in any direction.

The first test was implemented as a Java application, where the number of sessions, connected mobile users and running time were set in a GUI. The application worked fine for small amounts of concurrent clients. Each client was spawned in a separate thread. As the number of sessions reached 10, with one user per session, the messages sent from user to IPTV were delayed by 187ms. Something was obviously wrong, since that small amount of users was assumed to be handled with very low latency. It was concluded that the implementation of the test suite was poor or the socket handling mechanism of Java wasn’t able to handle 20 concurrent sockets. The test suite was scrapped in favor for another solution.

The second suite was implemented in node.js. As the framework was design towards serving multiple concurrent network connections, it seemed to be suitable. The implementation was basically the same as with the Java, except that all clients ran in a shared thread and the lack of a GUI. The test was successful as a delay of 200ms wasn't reached at a session count of 1000 and 2 users at each.

8.2.1 Result

The test showed that a server running with a maximum support for 300 sessions, each with five associated users, or 800 sessions, each with two associated users, only had maximum an average latency of 25ms. This value was reached running the node.js framework. Increasing number of sessions or users resulted in a latency of 200ms. It was however outnumbered by the APE framework, which only had a latency of 6ms for each direction. APE allowed for up to 3000 users when running 1000 sessions with two associated clients (+1 IPTV-app). The same parameters for node.js resulted in a latency of $\geq$ 163ms. See appendix, section 11.5 for the complete result.

8.2.2 Conclusion

It can be concluded that the most preferred server framework to is use APE, if the system is required to handle a high number of users. APE supports the most number of concurrent clients, with respect to low latency.

The test does not simulate reality with respect to web connections. All clients, the IPTV-app as well, were connected using Internet sockets. The latency might not be as low in a production environment. Performing this kind of testing was excluded since the most reasonable procedure would be to use as many computers as sessions, each running a browser environment. There was no time, nor equipment, available to manage carrying out this kind of test. Bearing this fact in mind, further investigation on the performance of the JS provided libraries must be done.

9 Future work

9.1 Future work

9.1.1 Testing with a significant number of web connections

There have only been two different stress tests carried out during this project. One that has been continuously ongoing, is with a single web connection and a maximum of two simultaneous mobile clients. The other test is the stress test with several hundred simulated mobile clients. As none of those tests determine how a real scenario with a large number of sessions, including both web and socket connections, would behave, it can't be concluded that this ecosystem is flawless considering response time and stability.

It's therefor strongly suggested to set up a test environment with features closer related to a production environment, than the environment specified in the stress test in section 8.2.

9.1.2 Exception handling for erroneous messages

Although there is a clear instruction set defined, an inexperienced developer might create erroneous messages, thus resulting in communication misses and undetected bugs. Since the message wrappers at the mobile application libraries are custom written, it's suggested that they are extended with algorithms for validating the messages created. By throwing exceptions at message creation, in case of invalid message, the application can take further actions before communicating erroneously.

9.1.3 Mobile application with built-in server

Another approach to the stand-alone server is the possibility to create an instance of a web server in the mobile application. One of the session clients would act server and let the other clients connect by notifying them about its existence. It's assumed that all clients are connected to the same network. The benefit would be the decreased latency as the data isn't routed to the Internet.

The biggest obstacle identified is the fact that each application need to run a fully functional web server implementation. It must support all different kinds of communication mechanisms, ranging from Forever iFrame to WebSockets. It must also handle connections from other mobile clients, which in this case is a regular socket server implementation.

The other problem to solve is the maintainability of the session, as the mobile client running the server might occasionally disconnect or die. It requires some distributed communication among the other clients, to determine who's to take responsibility for constituting the new server.

This alternative route is way beyond the scope of this project, but still an interesting option if it turns that the connection latency, for some reason, is unacceptably long.

9.1.4 Disconnected users

Something that's missing today is the tolerance for accidentally disconnecting users. As of now, they are required to reconnect and re-authenticate with the system. There's no way to distinguish between the accidentally and intentionally disconnected users. The other session members receive a disconnect message, but are unaware of whether the user will return, or not. If it was possible to distinguish between these cases, the other users would know how to respond to the disconnection - disconnect or wait for the user to return.

10 Bibliography

References

[1] Dvb - digital video broadcasting: Standards bluebooks: Internet protocol. http://www.dvb.org/technology/standards/#internet. Accessed 2011-06-06.

[2] The new iptv standard oipf. http://www.tele-satellite-global.com/TELE-satellite-1103/eng/oipf.pdf. Accessed 2011-06-06.

[3] Itu-t: Y.1991: Terms and definitions for iptv. http://www.itu.int/rec/T-REC-Y.1991-201003-I/en. Accessed 2011-06-06.

[4] LM Ni P Zeng. Spotlight: The rise of the smart phone. *Distributed Systems Online, IEEE*, March 2006.

[5] J Schneidawind. Big blue unveiling. *USA Today*, 1992-11-23.

[6] Nokia 9000 communicator detailed specs. http://pdadb.net/index.php?m=specs&id=879&view=1&c=nokia%209000%20communicator. Accessed 2011-06-06.

[7] Michael Simon. From 2-way to 4g: The complete history of cell phones. http://www.maclife.com/article/feature/2way_4g_complete_history_cell_phones?page=0,1, July 2010. Accessed 2011-06-06.

[8] Jim Davis. Short take: Blackberry wireless email device debuts. http://news.cnet.com/Short-Take-BlackBerry-wireless-email-device-debuts/2110-1040_3-220388.html, Januar 1999.

[9] Analysis: How smartphone analysis: How smartphone platforms compare. *InformationWeek*, 2007-01-20.

[10] Open handset alliance: Oha faq. http://www.openhandsetalliance.com/oha_faq.html. Accessed 2011-06-06.

[11] Admob. Metrics highlights. *AdMob Mobile Metrics*, 2010-05. http://metrics.admob.com/wp-content/uploads/2010/06/May-2010-AdMob-Mobile-Metrics-Highlights.pdf.

[12] Android market client update. http://android-developers.blogspot.com/2010/12/android-market-client-update.html, 2010-12. Accessed 2011-06-07.

[13] ios 4: The world's most advanced mobile operating system. http://www.apple.com/ipad/ios4/. Accessed 2011-06-07.

[14] Fujitsu. Fujitsu launches secure slate pc for business class. http://www.fujitsu.com/global/news/pr/archives/month/2011/20110224-01.html, 2011-02-24. Accessed 2011-06-07.

[15] Elisha Gray. Us patent: Us0386815: Telautograph. Technical report, 1888-07-31.

[16] Tom Hormby. The story behind apple's newton. http://lowendmac.com/orchard/06/john-sculley-newton-origin.html, 2006-02-07. Accessed 2011-06-07.

[17] Don Reisinger. Idc: Apple ipad secures 87 percent market share. http://news.cnet.com/8301-13506_3-20028801-17.html, 2011-01-18. Accessed 2011-06-07.

[18] Weiguo Ye Feida Lin. Operating system battle in the ecosystem of smartphone industry. *2009 International Symposium on Information Engineering and Electronic Commerce*, page 5, 2009.

[19] Gartner. Gartner says worldwide smartphone sales grew 29 percent in first quarter of 2008. http://www.gartner.com/it/page.jsp?id=688116, 2008-06-06. Accessed 2011-06-07.

[20] Gartner. Gartner says worldwide smartphone sales reached its lowest growth rate with 3.7 per cent increase in fourth quarter of 2008. http://www.gartner.com/it/page.jsp?id=910112, 2009-03-11.

[21] Gartner. Gartner says worldwide mobile phone sales grew 35 percent in third quarter 2010; smartphone sales increased 96 percent. http://www.gartner.com/it/page.jsp?id=1466313, 2010-11-10. Accessed 2011-06-07.

[22] Joel Åsblom. Sony ericsson slutar använda symbian. http://www.idg.se/2.1085/1.342282/sony-ericsson-slutar-anvanda-symbian, 2010-09-27. Accessed 2011-06-07.

[23] Thomas Ricker. Samsung terminates support for symbian. http://www.engadget.com/2010/10/01/samsung-terminates-support-for-symbian-development/, 2010-10-01. Accessed 2011-06-07.

[24] Joel Åsblom. Motorola byter symbian mot android. http://www.idg.se/2.1085/1.189278/motorola-byter-symbian-mot-android, 2008-10-30. Accessed 2011-06-07.

[25] Mats Lewan. Nokia ger upp symbian och väljer microsoft. http://www.nyteknik.se/nyheter/it_telekom/mobiltele/article3096279.ece, 2011-02-11. Accessed 2011-06-07.

[26] David Flynn. Microsoft: "no windows phone 7 upgrade for windows mobile 6.x devices". http://apcmag.com/microsoft-no-windows-phone-7-upgrade-for-windows-mobile-6x-devices.htm, 2010-03-01. Accessed 2011-06-07.

[27] Apple. Mac os x developer library: Mac os x technology overview: Kernel and drivers. http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/OSX_Technology_Overview/SystemTechnology/SystemTechnology.html#//apple_ref/doc/uid/TP40001067-CH207-TPXREF172, 2009-08-04. Accessed 2011-06-07.

[28] David Meyer. Zdnet: Linux developer explains android kernel code removal. http://www.zdnet.com/news/linux-developer-explains-android-kernel-code-removal/389733, 2010-02-03. Accessed 2011-06-07.

[29] Apple. Mac os x developer library: Darwin and core technologies. http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/OSX_Technology_Overview/SystemTechnology/SystemTechnology.html, 2009. Accessed 2011-06-07.

[30] Android developers: What is android? http://developer.android.com/guide/basics/what-is-android.html, 2011-06-06. Accessed 2011-06-07.

[31] Apple. Apple ios developer program. http://developer.apple.com/programs/ios/. Accessed 2011-06-07.

[32] Apple. ios developer program: Distribute your app. `http://developer.apple.com/programs/ios/distribute.html`. Accessed 2011-06-07.

[33] Android developers: Installing the sdk. `http://developer.android.com/sdk/installing.html`. Accessed 2011-06-07.

[34] Android market for developer help: Transaction fees. `https://www.google.com/support/androidmarket/developer/bin/answer.py?&answer=112622`. Accessed 2011-06-07.

[35] Yahoo! A new model for device communication from yahoo! connected tv. White Paper, 2011-01.

[36] Samsung tv remote control app for iphone, ipod touch and ipad. `http://www.samsunghub.com/2010/04/09/samsung-tv-remote-control-app-for-iphone-ipod-touch-and-ipad/`, 2010-04-09. Accessed 2011-06-07.

[37] Samsung remote by samsung electronics co.ltd. `http://itunes.apple.com/us/app/tv-remote/id359580639`, 2011-06-02. Accessed 2011-06-07.

[38] IBM. Ibm: Communications programming concepts: Socket connections. `http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.progcomm/doc/progcomc/skt_connect.htm`. Accessed 2011-06-07.

[39] Chuck David Jason Forrester Wei Liu Carolyn Matthews Nicolas Rosselot Lydia Parziale, David T Britt. *TCP/IP Tutorial and Technical Overview*. IBM, 2006.

[40] I. Hickson. The web socket protocol. Technical report, IETF: Network Working Group, 2009.

[41] Chris Heilman. Websocket disabled in firefox 4. `http://hacks.mozilla.org/2010/12/websockets-disabled-in-firefox-4/`, 2010-12-08. Accessed 2011-06-07.

[42] Adam Barth Eric Rescorla Lin-Shung Huang, Eric Y Chen and Collin Jackson. Talking to yourself for fun and profit. 2011.

[43] Netcraft. Netcraft: January 2011 web server survey. `http://news.netcraft.com/archives/2011/01/12/`

january-2011-web-server-survey-4.html, 2011-01-12. Accessed 2011-06-07.

[44] Microsoft. Microsoft mediaroom ¿ for you. http://www.microsoft.com/mediaroom/you/, 2011. Accessed 2011-06-07.

[45] Kevin Whinnery. Stackoverflow: iphone - what happens to javascript code after app is compiled using titanium mobile. http://stackoverflow.com/questions/4217551/what-happens-to-javascript-code-after-app-is-compiled-using-titanium-mobile/4798547#4798547, 2011-01-25. Accessed 2011-06-07.

[46] All-Seeing Interactive. Asihttprequest documentation. http://allseeing-i.com/ASIHTTPRequest/, 2011. Accessed 2011-06-07.

[47] Android developers: Reference: Httpclient. http://developer.android.com/reference/org/apache/http/client/HttpClient.html, 2011-06-06. Accessed 2011-06-07.

[48] Java 1.4.2 documentation: java.util: Class hashtable. http://download.oracle.com/javase/1.4.2/docs/api/java/util/Hashtable.html, 2010. Accessed 2011-06-07.

[49] Apple. ios developer library: Nsdictionary class reference. http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/Reference/Reference.html, 2011-01-18. Accessed 2011-06-07.

[50] Stig Brautaset. Json framework. http://stig.github.com/json-framework/. Accessed 2011-06-07.

[51] Douglas Rockford. Json in java. http://json.org/java/. Accessed 2011-06-07.

[52] M D Mcllroy B W Kernighan. *UNIX Programmer's Manual.* Bell Telephone Laboratories, Murray Hill, New Jersey, 1979.

[53] Microsoft: Exploring the observer design pattern. http://msdn.microsoft.com/en-us/library/Ee817669(pandp.10).aspx, 2002-01. Accessed 2011-06-07.

[54] Sun Microsystems. Java language specification 3rd edition: Chapter 8: Overloading. http://java.sun.com/docs/books/jls/third_edition/html/classes.html#227768, 2005. Accessed 2011-06-07.

[55] Apple. ios developer library: Cocoa objects. http://developer.apple.com/library/ios/#DOCUMENTATION/Cocoa/Conceptual/CocoaFundamentals/CocoaObjects/CocoaObjects.html, 2010-12-13. Accessed 2011-06-07.

[56] Apple. Ios developer library: About events in ios. http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Introduction/Introduction.html, 2010-09-29. Accessed 2011-06-07.

[57] Apple. Mac os x developer library: Selectors. http://developer.apple.com/library/mac/\#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocSelectors.html. Accessed 2011-06-07.

[58] Robbie Hansson Luke Steffen. cocoaasyncsocket. http://code.google.com/p/cocoaasyncsocket/.

[59] Arcade Museum. Pong. http://www.arcade-museum.com/game_detail.php?game_id=9074. Accessed 2011-06-07.

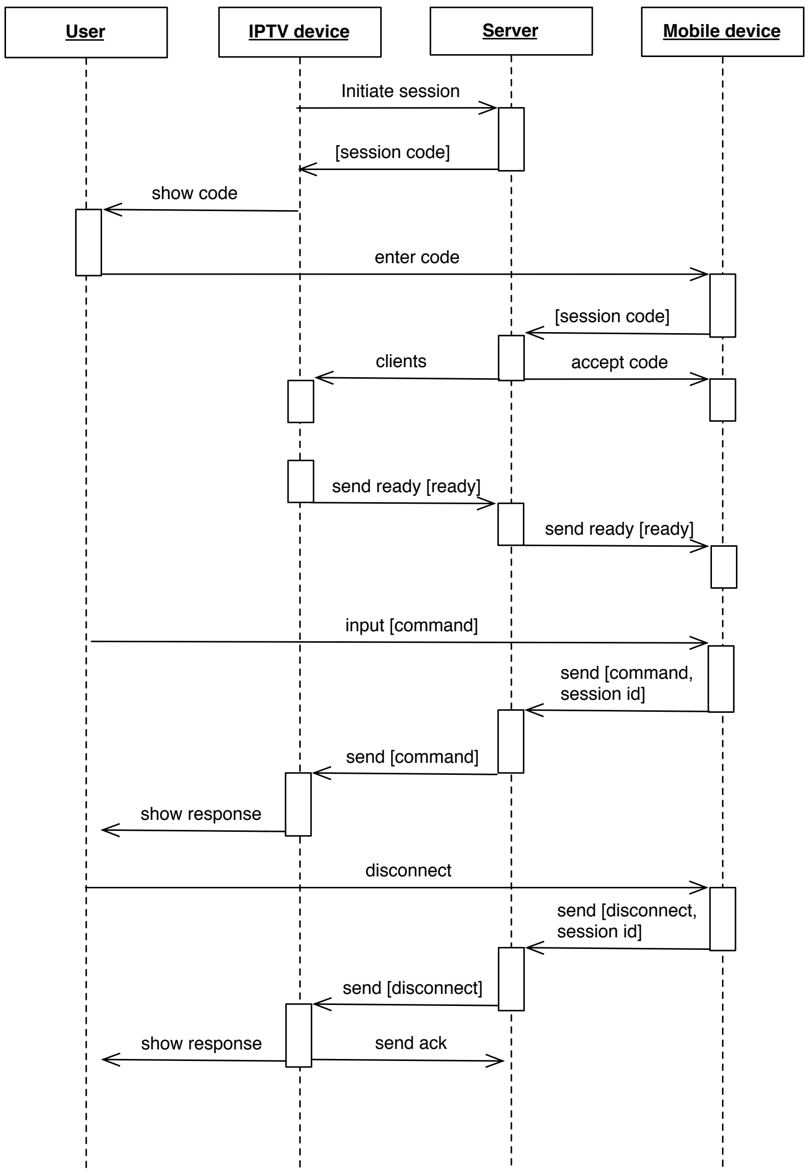
11 Appendix

11.1 List of popular smart phones

Smartphones						
	HTC Desire HD	**HTC Nexus One**	**SE Xperia X10**	**Samsung Galaxy S**	**Apple iPhone 4**	**Nokia N8**
Release date (US)	Oct 19, 2010	Jan 5, 2010	Aug 15, 2010	June, 2010	Jun 24, 2010	Oct 30, 2010 (Finland)
OS	Android 2.1	Android 2.1	Android 2.1	Android 2.2	iOS 4.2	Symbian^3
CPU	QSD8250 1GHz	QSD8250 1GHz	QSD8250 1GHz	S5PC110 1GHz	ARM Cortex-A8 Apple A4	680MHz ARM 11
Screen	3.7”, 480x800 WVGA	3.7”, 480x800 WVGA	4.0”, 400x854	4.0”, 480x800	3.5”, 960x640	3.5”, 640x360
Storage	ROM 512MB, RAM 576MB, microSD	ROM 512MB, RAM 576MB, microSD	1GB, RAM 384 MB, microSD	8/16GB + microSD, 378MB RAM	16/32 GB, 512MB RAM	16GB, 256MB RAM
GPS	Yes	Yes	Yes	Yes	Yes	Yes
Accelerometer	Yes	Yes	Yes	Yes	Yes	Yes
Gyroscope	No	No	No	No	Yes	No
Compass	Yes	Yes	No	Yes	Yes	Yes

Tablets						
	Motorola Xoom	**Dell Streak 7**	**iPad**	**Samsung Galaxy Tab**	**Blackberry Playbook**	**Notion Ink Adam**
Release date (US)	Feb 24, 2011	Feb 2, 2011	Apr 3, 2010	Nov 10, 2010	Apr 10, 2011	Jan 21, 2011
OS	Android 3.0	Android 2.2	iOS 4.2	Android 2.2	Blackberry Tablet OS	Android 2.3
CPU	Nvidia Tegra Dual-Core 1GHz	Nvidia Tegra Dual-Core 1GHz	ARM Cortex-A9 Apple A4	ARM Cortex A8 1.2 GHz	Dual Core Cortex-A9	Dual Core Cortex-A9
Screen	10.1”, 1280x800	7”, 800x480	9.7”, 1024x768	7”, 1024x600	7”, 1024x600	10.1”, 1024x600
Storage	32GB	16/32GB	16/32/64GB	16/32GB + microSD	1GB RAM	8GB
GPS	Yes	Yes	Yes	Yes	Yes	Yes
Accelerometer	Yes	Unknown	Yes	Yes	Yes	Yes
Gyroscope	Yes	Unknown	No	Yes	Unknown	No
Compass	Unknown	Unknown	Yes	Unknown	Unknown	Yes

11.2 Use case diagram describing the life of a session



11.3 Protocol description in Backus-Naur Form

```
(*
   json-object is not defined in this BNF. For the full
   specification, please consult http://www.json.org/
*)

swipe-command ::= 'swipe'
position-command ::= 'continuous' | 'tap'
input-command ::= 'string'
session-command ::= 'connect' | 'disconnect' | 'ready' | '
notReady' | 'newSession'
code-command ::= 'code'
response-command ::= 'ack' | 'codeOk' | 'invalidCode' | '
invalidCmd'
clients-command ::= 'clients'
generic-command ::= 'customCmd'

json-swipe-command ::= '"command" : ' <swipe-command>
json-position-command ::= '"command" : ' <position-command>
json-input-command ::= '"command" : ' <input-command>
json-session-command ::= '"command" : ' <session-command>
json-response-command ::= '"command" : ' <response-command>
json-code-command ::= '"command" : ' <code-command>
json-generic-command ::= '"command" : ' <generic-command>
json-clients-command ::= '"command" : ' <clients-command>
command ::= '"command" : ' <command>

digit-non-zero ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' |
'8' | '9'
digit  ::= {<digit-non-zero>} | '0'
digits  ::= {<digit-non-zero}+ {<digit>}*
float  ::= '0.' {<digit>}+ | {<digit-non-zero>}+ {<digit>*} '.'
 {<digit>}+
alpha-char ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' |
'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's'
 | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' ;
char-digit ::= <alpha-char> | <digit>

(*
 A direction is the direction in which a swipe can be carried
 out
*)
direction ::= 'up' | 'right' | 'down' | 'left'
json-direction ::= '"direction" : ' direction

(*
   A coordinate describes a point on a surface.
*)
```

```
coordinate ::= <float>
json-coordinates ::= '"coordinates": [' <coordinate> ', ' <
coordinate> ']'

(*
   The code is used by mobile device client for pairing with
   IPTV app.
   It must be unique for all existing sessions.
*)
code ::= {<char-digit>}+
json-code ::= '"code" : ' <code>

(*
   Unix timestamp describing time when the message was sent
*)
timestamp ::= {<digits>}+
json-timestamp ::= '"timestamp" : ' <timestamp>

(*
   An id respresents a unique identifier of a message.
*)
id ::= {<digits>}+
json-id ::= '"id" : ' <id>

boolean ::= 'true' | 'false'

(*
   The session datatype uniquely identifies a session.
*)
session ::= '"' <alpha-char>}+ '"'
json-session ::= '"session" : ' <session>

(*
   Defines whether the sent message must be acknowledged by
   receiver.
   It might be complex to demand that a broadcasted package
   must be acknowledge.
*)
json-requires-ack ::= '"requires-ack" : '<boolean>

(*
   A client is some device involved in the session
*)
client-type ::= 'iptv' | 'mobile' | 'server'
client ::= <id> <client-type>
clients ::= {client}+
json-client ::= '"id" : "' <id> ', "type" : ' <client-type>
json-clients ::= '[' <json-client> [{', ' <json-client>}*] ']'
```

```
(*
   A receiver is a client identified by a unique string.
*)
receiver ::= {<alpha-char>}+
receivers ::= '"' <receiver> '"' {', "' <receiver> '"'}*
json-receivers ::= '"receivers" : [' <receivers> ']'

(*
   A source is a client identified by a unique string.
*)
source ::= {char-digits}*
sources ::= '"' <source> '"' {', "' <source> '"'}*
json-sources ::= '"sources" : [' <sources> ']'

(*
   A source view is the specific view that triggered the event.
*)
source-view ::= {char-digits}*
json-source-view ::= '"source-view" : ' <source-view>

json-message-head ::=
   '"id" : ' <json-id>
json-message-options ::=
   [<json-session>', '][<json-timestamp>', '][<json-receivers
   >', '][<json-sources>', '][<json-requires-ack>', ']
json-message-body       ::=
   <json-swipe-command> ', "value" : {' direction [', ' <json-
   source-view> ] '}' |
   <json-position-command> ', "value" : {' <json-coordinates>
   [', ' <json-source-view>] '}' |
   <json-input-command> ', "value" : { "text" : ' {<char-digit
   >}+ [', ' <json-source-view>] '}' |
   <json-session-command> [', "value" : { "numclients:"' {<
   digits>}+ ' }'] |
   <json-code-command> ', "value" : {' <json-code> '}' |
   <json-response-command> ', "value" : {' (<json-id> | <json-
   code>) '}' |
   <json-clients-command> ', "value" : {' <json-clients> '}' |
   <json-generic-command> ', "value" : {' <json-object> '}'

(*
   A json-message is what to be transmitted to the receiver(s).
    It is composed of a head; options definining session id,
   timestamp and receivers; and a body.
*)
json-message ::=
   '{' <json-message-head>[',' <json-message-options>][',' <
   json-message-body>] '}'
```

11.4 Explanation of messages

Protocol definition

Introduction

This document, along with the BNF, describes the communication protocol that's intended to be used along with the [NAME] library.

Messages

Several different messages constitute the protocol. Every message is expressed in JSON format since the IPTV application and the server are supposed to run software that can parse the messages without need to reformat.

Session

The protocol defines a session as a regular session related to communication protocols. It is created when the IPTV application wants to support mobile devices. Every client is member of a specific session. When the server decides to close all connections, the session is invalidated and destroyed.

The session identifier must validate against the regular expression [a-z0-9]+. It's up to the programmer to define its length and come up with a good algorithm for generating a unique identifier.

Setting up the connection

The sequence for setting up the connection and pairing one device with the IPTV applications is as follows:

1. The IPTV application connects to the server.
2. When connected, the IPTV application sends the message *newSession* to instantiate a new session.
3. The server responds with the pairing code wrapped in the message *code*.
4. The code is shown to the user, which inputs the code in the related application on the mobile device.
5. The mobile device transmits the code to the server with message *input*, which responds with either *codeOk* or *invalidCode*. The *invalidCode* response is sent if the code is invalid or maximum number of clients is reached.
6. If *invalidCode*, tell user to resubmit code. Else, the *codeOk* message contains the session id and the mobile client id, which is to be stored.
7. The server sends an updated list of clients via message *clients.
8. When the IPTV application has determined that sufficient number of clients have connected, it sends *ready* to all clients in the session.
9. From here on, it's up to the developer to determine how interaction should be carried out.

Changelog

2011-02-08: First draft 2011-04-12: Changed the sequence for setting up connection.

Messages

Introduction

Id

The id is a mandatory property that need to be set for all messages, regardless they will be acknowledged or not.

Options

Each message can consist of several options. Some of those are requires by some messages. They are optional in most cases though. The messsages that explicitly require any of these options reveal that information in the "Mandatory values" section.

Session

The session identifies the current session. It's made up of a **json-session** object. For every message that is related to a specific session, this object is required.

Timestamp

In the case that we want to send messages and be specific about the duration between to consecute messages, we set the **json-timestamp**. It can either be expressed as a UNIX timestamp or relative time. It's up to the programmer to decide upon that and the time unit (e.g. seconds, milliseconds, etc.)

Receivers

Some messages are only intented to be sent to a specific subset of all clients that are involved in a certain session. This object contains the identifiers of those clients. By specifying the clients, the sender will be sure that no one else receives the message.

If the receiver isn't defined, the server will take care of the message and make sure that it's delegated to the correct receiver.

Sources

In the case of a stakeholder wanting to inform the receiver about a group of sources, this object is filled with their unique identifiers.

Requires ack

Some messages need to be acknowledged in order for client to proceed to next step. A possible scenario is when the user wants to select an object, which he/she will interact with. The selection must be finished before anything else can happen.

By setting **json-requires-ack** to true, the message is guaranteed to be acknowledged by the receiver(s).

Types of messages

Swipe

Messages members of this group share the denominator of being discrete gestures. There are only four gestures supported at the moment.

swipeUp

Intention

When the user has swiped the screen from the bottom to the top, this message will be sent. It only communicates the discrete movement of swiping the screen, no speed or detailed direction. It should be based on the device's gesture recognizer and therefore be defined by the device itself, rather than relying on the developer to implement the correct calculations.

Mandatory values

There are no mandatory values, apart from the *id* and *commmand* parameters.

Optional values

There are no optional values

Constraints

There are no constraints.

swipeDown

Same behavior as *swipeUp*, except that the direction of the swipe is from the top to the bottom of the screen.

swipeLeft

Same behavior as *swipeUp*, except that the direction of the swipe is from right to left side of the screen.

swipeRight

Same behavior as *swipeUp*, except that the direction of the swipe is from left to right side of the screen.

Position

Whenever something is intented to change position, or the device wants to send a coordinate determined by the users interaction with the screen, messages from this group are sent.

tap

Intention

The *tap* event is intended to be used when a user has tapped the screen and expects some kind of response. The message must contain the coordinates defining the position for where the tap has occurred. There's no distinction between a single tap and multi taps.

Mandatory values

The **value** object must contain a **coordinates** object, identifying the tap's coordinates.

Constraints

There are no constraints

continuous

Intention

The message is sent for each coordinate that is registered as the devices recognizes an arbitrary gesture. The gesture does not necessarily need to be recognized as a certain one, the sole notion of the movement being any gesture is enough.

If a multitouch gesture is recognized, it must be decomposed into single gestures, each triggering each event. This condition doesn't need to be fulfilled if the gesture is taken care of somewhere else.

Mandatory values

The **value** object must contain a **json-coordinates** object, identifying the tap's coordinates.

Optional values

The **value** object can contain a **json-source-view** to identify the object identifying the gesture.

Constraints

There are no constraints.

Input

Input other than swipes and position changes is handled by members of this group.

string

Intention

A string is a set of characters that constitue a meaning. It doesn't require any detailed description.

Mandatory values

The **value** object must contain a **{<char-digit>}*** object, which constitutes the string.

Optional values

The **value** object can contain a **json-source-view** object to identify the object sending the string. This can be used to identify what kind of input we are dealing with.

Session

All functions handling the connection between the clients (IPTV and mobile device) and the servers are gathered in this group. The members are quite rudimentary.

Disconnect

Intention

This message is bi-directional, with two different meanings depending on sender and receiver.

1. Server sender and client receiver: The connection has been cancelled by either server or another client. If the message contains a **json-source** object, it can imply that one client has left the party in a multiplayer game, or something else. If the **json-source** object is null, the session will end and the client must close the connection.
2. Client sender and server receiver: The client will close the connection and notifies the server.

Mandatory values

There are no mandatory values.

Optional values

There are no optional values.

ready

Intention

Whenever a client or the server is ready, it broadcasts the message to notify the other stakeholders. When a client has sent *notReady*, it must send *ready* in order to be able to receive messages. Therefore, other clients need to wait for it to be ready. What's happening to messages being send during the *notReady ready*-phase is unknown.

When establishing connection to the server and setting up necessary properties (such as views and calculating data), the client is in the "not ready" state. The server can't start the initial comunication unless the client has issued the *ready* message.

Mandatory values

The **json-source** must contain the sending client's identifier.

Optional values

Unless the message shall be broadcasted, the receivers are specified by pushing each's id to **json-receiver**.

notReady

Intention

As a client is occupied by heavy calculations, waiting for some data to arrive, etcetera, it may broadcast this message. The clients receiving this message, must be aware of the fact that the messages that they send will not arrive properly. Whenever the client once again is ready, it will communicate the *ready* message.

Mandatory values

The **json-source** must contain the sending client's identifier.

Optional values

Unless the message shall be broadcasted, the receivers are specified by pushing each's id to **json-receiver**.

newSession

Intention

Send by IPTV application to request instantiation of a new session. Server will respond with a *code* message containing the session code.

Mandatory values

The **json-source** must contain the sending client's identifier.

Optional values

Unless the message shall be broadcasted, the receivers are specified by pushing each's id to **json-receiver**.

Code

code

Intention

The code that is needed to set up pairing is distributed by this message. Each client sets up pairing by having the user inputting the code shown by the IPTV app and send to the server. There should be no need to send the code to clients other than the IPTV app.

Mandatory values

The **value** object must contain a **json-code** object, where the "code"-property is not equal to an empty string.

Since the code mustn't be broadcasted, the **json-receiver** must specify the id of the IPTV client.

Optional values

There are no optional values.

Response

These message don't contain any valueable data. They are only responses to messages that need to be acknowledged in some way.

ack

Intention

Each message received with **json-requires-ack** set to true, must be acknowledged.

Note that the **id** object isn’t globally shared. Two clients can send different messages with same id at the same time. It’s therefore important to make sure that the correct receiver is specified.

Mandatory values

The **value** must contain a **json-id** identifying the acknowledged message.

Optional values

There are no optional values.

codeOk

Intention

Whenever a client wants to pair with the IPTV app, it sends the pairing code to the server. The server responds with this message if the code is correct.

Mandatory values

The **value** must contain a **json-session** identifying the session the device is connected to.

Optional values

There are no optional values.

invalidCode

Intention

This is the antagonist of *codeOk*. If the client sends an invalid code, the server responds with this message.

Mandatory values

The **value** must contain a **json-id** identifying the message containing the submitted code.

Optional values

There are no optional values.

invalidCmd

Intention

If a client sends a message that’s not valid, this is the response. An invalid message is a message that’s not identified by this protocol or a generic message.

Mandatory values

The **value** must contain a **json-id** identifying the message containing the submitted code.

Optional values

There are no optional values.

Clients

clients

Intentions

The messsage informs the receiver of the clients connected to the system, including the receiver as well.

Mandatory Values

The **value*** must contain a **json-clients** containing at least two elements (the receiver and the server).

Optional values

There are no optional values

Examples

Note that the JSON objects in these examples occur in two version, one unwrapped and one that flies with the BNF definition.

Swipe

Apart from the mandatory values, **id** and **command**, it also requires receiver to acknowledge the message and informs that the swipe occurred on object named “picture001”.

Unwrapped:

```
{
    "id" : 1,
    "session" : "af91e",
    "requires-ack" : true,
    "command" : "swipeDown",
    "value" :
    {
        "source-view" : "picture001"
    }
}
```

Wrapped:

```
{ "id" : 1, "session" : "af91e", "requires-ack" : true, "command" : "swipeDown",
"value" : { "source-view" : "picture001"
} }
```

Continuous

The message has id 2 and is sent to the clients with identifier 10 and 12. It doesn't require an acknowledgement, which it explicitly defines (not necessary). It's communicating that the user moved the finger over the coordinate (24.0, 13.0) on the object "drawingpad002".

Unwrapped:

```
{
    "id" : 2,
    "session" : "af9e2",
    "receivers" :
    [
        "10",
        "12"
    ],
    "requires-ack" : false,
    "command" : "continuous",
    "source-view" : "drawingpad002",
    "value" :
    {
        "coordinates" :
        {
            "x" : 24.0,
            "y" : 13.0
        }
    }
}
```

Wrapped:

```
{ "id" : 2, "session" : "af9e2", "receivers" : [ "10", "12" ], "requires-ack" :
false, "command" : "continuous", "source-view" : "drawingpad002", "value" : {
"coordinates" : { "x" : 24.0, "y" : 13.0 } } }
```

11.5 Server stress test result data

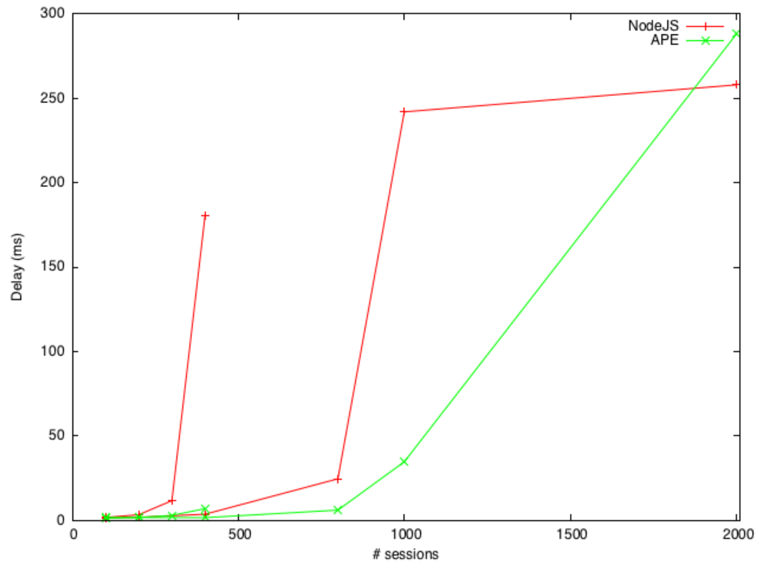


Figure 11: Mean latency from user to IPTV

# sessions	# users per session	node.js (ms)	APE (ms)
100	5	1.78	1.30
200	5	3.37	1.73
300	5	11.7	3.02
400	5	180	6.98
100	2	1.46	1.75
200	2	1.87	1.73
400	2	3.67	1.63
800	2	24.65	6.11
1000	2	242	34.75
2000	2	258	288.12

Table 5: Mean latency from user to IPTV

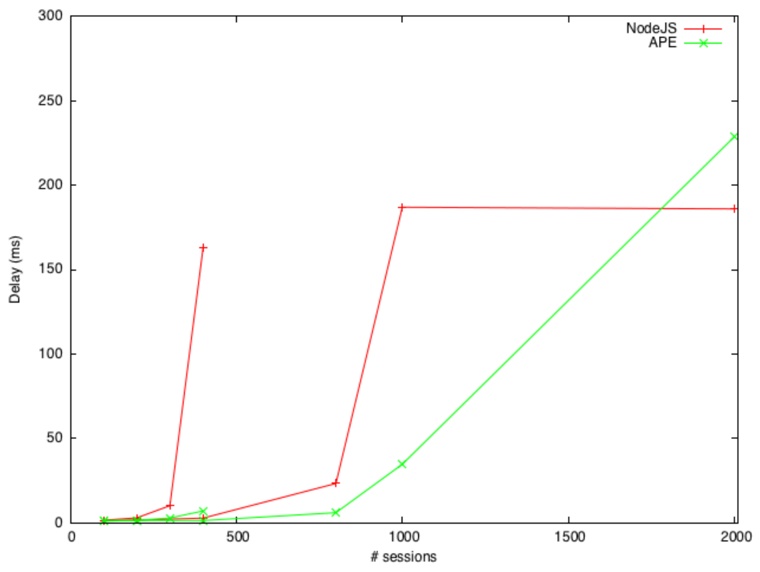


Figure 12: Mean latency from IPTV to user

# sessions	# users per session	node.js (ms)	APE (ms)
100	5	1.60	1.01
200	5	2.98	1.52
300	5	10.2	2.89
400	5	163	7.07
100	2	1.30	1.57
200	2	1.63	1.52
400	2	2.84	1.4
800	2	23.4	6.06
1000	2	187	34.8
2000	2	186	229

Table 6: Mean latency from IPTV to user