Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería en Informática

PROYECTO FIN DE CARRERA

Design and Development of a Web-Based GUI for ScaleNet

Author: Víctor Pimentel Rodríguez
Tutor: Dr. José Ignacio Moreno Novella

October 2011

A mis padres y hermana: Inma, Julián y Sandra.

Resumen

Internet ha causado un tremendo impacto en muchos aspectos de nuestra vida cotidiana. A medida que la sociedad se va acostumbrando a las facilidades de trabajar en línea, los hábitos cambian de manera acorde. Aplicaciones que tradicionalmente se ejecutaban de manera nativa en la máquina del usuario se están, gradualmente, convirtiéndose en aplicaciones web ejecutadas remotamente.

Al mismo tiempo los navegadores han ido mejorando progresivamente hasta convertirse en potentes plataformas de desarrollo. Esta mejora ha dado lugar a la aparición de aplicaciones web de una gran complejidad basadas en HTML, CSS y JavaScript, distribuyendo una carga de procesamiento importante al cliente. A la vez, se obtienen interfaces flexibles capaces de adaptarse a dispositivos muy dispares.

En este proyecto se documenta el desarrollo de una aplicación web avanzada cuyo propósito es controlar la reproducción de contenidos multimedia en varios dispositivos. Esta aplicación se ha realizado en colaboración con *Deutsche Telekom AG*, durante un estancia de seis meses en Berlín como parte del programa *Erasmus Placement* en 2010.

Dicha aplicación se enmarca dentro del proyecto ScaleNet (2005-2009), una Red de Siguiente Generación (NGN) cuyo fin es un sistema que permita una integración escalable, rentable y eficiente de las diferentes tecnologías de acceso inalámbrico y por cable. El componente desarrollado, la *Interfaz de Administración de la Red Personal* (PNAI), es solo una pequeña parte de ScaleNet que sirve como ejemplo de aplicación sobre esta red.

Aunque la interfaz para estas operaciones ya existía, se solicitó un rediseño completo que integrara mayor funcionalidad y que ofreciera una experiencia de usuario más agradable. Además de la interfaz principal para ordenadores de escritorio, también se explica el desarrollo de una interfaz web para dispositivos táctiles modernos.

Abstract

Many aspects of our everyday life have been drastically affected by the Internet. As society becomes accustomed to the possibilities of working online, habits change accordingly. Traditional applications that are executed natively on the user's machine are gradually becoming web applications running remotely.

Meanwhile on the client, browsers are steadily improving to become powerful development platforms. This improvement has led to the emergence of highly complex web applications based on HTML, CSS and JavaScript, distributing significant processing loads to the client. At the same time, you get flexible interfaces able to adapt to very different devices.

This thesis documents the development of an advanced web application whose purpose is to control the playback of multimedia content across multiple devices. This application was completed in collaboration with *Deutsche Telekom AG*, during a six-month stay in Berlin as part of the *Erasmus Placement* in 2010.

This application is part of the ScaleNet project (2005-2009), a Next Generation Network (NGN) aimed at a system that provides a scalable, cost effective and efficient integration of different wireless and wireline access technologies. The developed component, the *Personal Network Administration Interface* (PNAI), is only a small part of ScaleNet that serves as an example application on this network.

Although the interface for these operations already existed, a complete redesign was requested to integrate more functionality and to provide a more pleasant user experience. In addition to the primary interface for desktop computers, this document also covers the development of a mobile web interface for modern touch devices.

Acknowledgements

I would like to thank my tutor José Ignacio for guiding me from Madrid, and Steffen, Dmitry and Hans for their warm welcome and constant help during my stay in Berlin. Also thanks to T-Labs and the Erasmus Placement program for making this experience possible.

Thanks to Carolina, Javi, Laura, Lucas, Phillip and the rest of the friends that I made in Berlin, six months flew by because of their company.

Since this project culminates my studies in Universidad Carlos III, I also want to thank all the friends, colleagues and teachers that I met and that help me during all these years. You made those endless hours working together in assignments seem bearable.

Finally, special thanks to my family for supporting me all these years and showing me all the patience in the world.

Me gustaría agradecer a mi tutor José Ignacio por orientarme desde Madrid, y a Steffen, Dmitry y Hans por su bienvenida y constante ayuda durante mi estancia en Berlín. También agradecezo a los T-Labs y al programa Erasmus Placement el hacer posible esta experiencia.

Gracias a Carolina, Javi, Laura, Lucas, Phillip y al resto de amigos que hice en Berlín, por su compañía seis meses pasaron volando.

Dado que este proyecto culmina mis años de estudio en la Universidad Carlos III, también quiero agradecer a todos mis amigos, compañeros y profesores que he conocido y que me han ayudado durante todos estos años. Habéis hecho fáciles de soportar las interminables horas trabajando juntos en prácticas.

Por último, un agradecimiento especial a mi familia por apoyarme todos estos años y por mostrarme toda la paciencia del mundo.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction and Objectives

Michael Scott : I enjoy having breakfast in bed. I like waking up to the smell of bacon —sue me— and since I don't have a butler, I have to do it myself.
So most nights before I go to bed I will lay six strips of bacon out on my *George Foreman* Grill.
Then I go to sleep.
When I wake up, I plug in the grill. I go back to sleep again.
Then I wake up to the smell of crackling bacon. It is delicious. It's good for me. It's the perfect way to start the day.
Today I got up, I stepped onto the grill and it clamped down on my foot. That's it. I don't see what's so hard to believe about that.

The Injury
The Office

1.1 Motivation

The current document describes the work done under a Erasmus Placement internship in the Deutsche Telekom Laboratories (T-Labs) of Berlin, developed during six months. This department is in charge of several research projects regarding, mainly, next generation networks.

The ScaleNet project was an attempt to converge several networks and services under the same system. These services may include voice and video telephony, mobile TV, video on demand, online gaming and internet access. Network convergence is seen as the migration of heterogeneous physical and logical network elements of fixed and mobile networks into one single (IP based) infrastructure.

This project was effectively finished in 2009, and one of the outcomes is an IP Multimedia Subsystem (IMS) demonstration environment. Using that demonstrator, service applications built on top of ScaleNet are shown to companies and organizations that are potentially interested in the technology.

One of those applications is the Personal Network Administration Interface (PNAI) application, a web based Graphical User Interface (GUI) from where the user can control running sessions in their devices. Sessions are simply instances of the previously mentioned services, and they could be from phone calls to video streaming.

The system can manage any number of devices for each user, so he can register his mobile phone, laptop, desktop or TV and he can manipulate the sessions running in any device. It has also some social capabilities implemented, so the user can also control a list of friends and can *buy*, start, transfer or duplicate sessions for any of his friends.

Additionally, another web application called IPTVplus is available to the user. The purpose of this application is the acquisition of video content: live streaming, video on demand, etc. The user can buy any content and a new session should automatically start on his default device.

The project motivation was improving the user interface by redesigning this PNAI, implementing some functional extensions and integrating the IPTVplus directly into the PNAI. With a little rethinking, the GUI could be modernized so that the demonstrator were stunningly effective.

Later in the project, this task extended to porting the same functionality to a mobile web application to modern devices such as the iPhone, the iPad or the Android devices. The system only supported *legacy* mobile devices like PDAs controlled by stylus, that were not really so sexy even back in their day.

1.2 Objectives

The initial objective of the work was creating a modernized version of the PNAI so that the interface were not only elegant but easy and fun to use. As this work is for demonstrating the underlying technologies, things like efficiency or complete stability were not so important.

Even then, quality is always a welcomed addition, so a generic objective was developing a considerable sized JavaScript application while maintaining a clear codebase. That is not very easy to do with JavaScript, so that impacted on the decision to use certain tools over others.

Also, the web application had to work on most browsers, but since we can control the demonstrator, we could reasonably drop some less used browsers if they could hinder the development. Because of that, compatibility with Internet Explorer (IE) 6 was dropped.

Giving that the PNAI application is a little part of ScaleNet, another goal was changing as less components as needed to avoid breaking other things. This translates in no changes in the database, no changes in the messages and, even if one component – the applet – was replaced, the rest of the system does not even notice it. Also, the deployment should be as simple as possible, the system already has a lot of dependencies.

As the project advanced, the goals evolved into filling other holes in the user experience. Between them, the most important goal was unifying all functionality regarding sessions in one interface. That is, to be able to buy content directly from the PNAI, without having to go to other page first.

Another new objective was building a mobile web application compatible with iPhone and other touch devices. A mobile web application was chosen instead of a native app because that way the application is

compatible with almost any device, and the web application will be faster to develop giving that most of the code can be recycled. However, native applications feel *better* than web pages, so that resulted in trying to mimic the experience of a native application in the mobile browser.

At the end of this document these objectives are evaluated to know if they were fulfilled, and at which level. Additionally, some future goals are delineated for a plausible iteration over this work.

1.3 Project Phases

Since the beginning, the project had fairly unambitious objectives, so the timetable kept expanding based on feedback from the finished work. According to the resulting development, the project can be structured in three phases, each of one taking a similar amount of time:

Initial phase To fulfill the original requirements, basically redesigning the PNAI interface and adding some basic functionality relating the device list. In this phase all JavaScript code was ported to MooTools classes.

Intermediate phase After that work was done, new requirements for the desktop version were added to implement more functionality into the PNAI interface. Apart from creating a sidebar with the content list, the applet was replaced by the Ajax Push Engine (APE) server. This also includes fixing all kind of bugs with IE, working with the PHP: Hypertext Preprocessor (PHP) files and porting the existing Java applet code to JavaScript.

Final phase The last goal in the project was creating the mobile version of the PNAI. Also, this marks the transfer of the static files from the Open Services Gateway Initiative (OSGi) bundle to the PHP server.

1.4 Document Structure

The work is mainly structured in four chapters, being this one the first. In this chapter motivation and objectives for this project have been discussed, and the project phases have been delineated.

The second chapter explains the state of the art, that is, the existing technologies this project is based on. This includes a somewhat detailed explanation of the ScaleNet project in its current state, although only the relevant parts for this work have been reviewed. Despite of the brevity of the existing documentation, some extra diagrams have been produced and added to help visualizing the system and more specifically the PNAI interface.

Besides that system description, all the basic technologies used for the work are discussed, stating their utility and place in this work. This sections talk about programming languages, platforms, libraries and techniques.

The third chapter concretes the actual development of this work, focusing on just the parts that have been modified. New requirements are compiled and analyzed, producing use cases for all the new situations that need to be handled. From that, the structure of the new system is described, starting from big components and going down to specific classes.

The development chapter continues with some implementation details that need further discussion or clarification. The last section compiles the deployment instructions for upgrading a previous ScaleNet installation to reckon the work in this project.

The four and last chapter evaluates this project, giving some conclusions to sum up the work made during those six months. Also, it provides future insight and advice to possible directions expanding this project.

The end of this document collects an estimated budget, the bibliography used in this project, the definitions of the acronyms and an index of the key works.

Chapter 2

State of the Art

Don Draper : Well, technology is a glittering lure.
But there is a rare occasion when the public can be engaged on a level beyond flash —if they have a sentimental bond with the product.
My first job I was in house at a fur company, with this old pro of a copywriter, a Greek, named Teddy. Teddy told me the most important idea in advertising is *new*. It creates an itch. You simply put your product in there as a kind of calamine lotion.
He also talked about a deeper bond with a product: *nostalgia*. It's delicate, but potent. Sweetheart.
[starts slide show featuring photos of Draper's family.]
Teddy told me that in Greek, nostalgia literally means the pain from an old wound. It's a twinge in your heart, far more powerful than memory alone.
This device isn't a space ship, it's a time machine. It goes backwards, forwards. It takes us to a place where we ache to go again. It's not called a wheel, it's called a carousel.
It lets us travel the way a child travels. Round and a round, and back home again.
To a place where we know we are loved.

The Wheel
Mad Men

2.1 Introduction

As the Internet evolves, Web development tools mature and multiply at an incredibly fast pace. In a few months the best choices becomes superseded by better and new tools. If we are dealing with a rewrite that is something to take into account.

This chapter describes the technologies used for this project. Since we are working on an existing system, most of them cannot be changed and are system constraints. And since ScaleNet includes so many different modules written in different languages and tools, it is wise to avoid adding even more layers of complexity.

A special case was the existing Java applet. Because of new requirements, an alternative had to be considered. Eventually it was replaced by a new module called the APE server. The other major addition to the system was the JavaScript Framework called MooTools. Both decisions are explained and justified in § 2.7 on page 96 and § 2.8 on page 103.

2.2 Existing System: ScaleNet

ScaleNet [1] is a research project developed between 2005 and 2009. Partly sponsored by the *German Ministry of Education*, several major corporations participated, including *Deutsche Telekom AG*, *Alcatel SEL AG*, *Eriksson GmbH*, *Lucent Technologies* and *Siemens AG*. T-Labs was specifically one of the departments more closely involved.

The aim of ScaleNet is to provide a Next Generation Network (NGN) that integrates different wireless and wireline access technologies. It is advertised as a scalable, cost effective and efficient Fixed and Mobile Convergence (FMC) solution.

Figure 2.1: ScaleNet logo

2.2.1 System Overview

ScaleNet addresses both service and network convergence. At the lower level, the system supports a multitude of heterogeneous physical and logical network elements of fixed and mobile networks into one single all-IP infrastructure. Figure 2.2 lists some of the protocols that could be used [2].

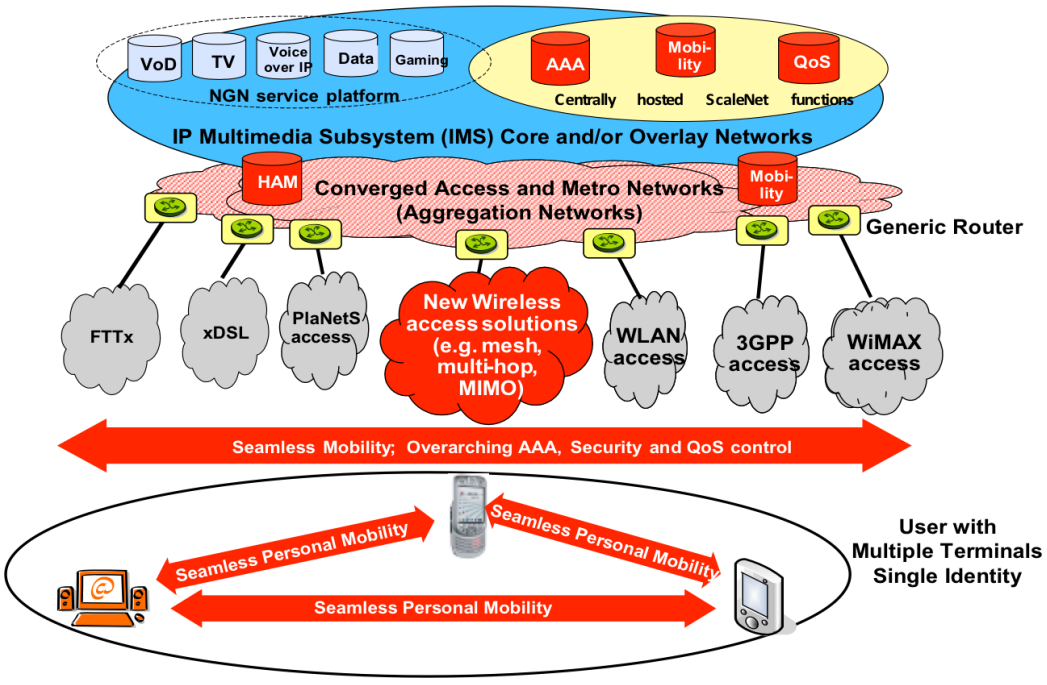


Figure 2.2: Structure of the system

At an upper level, multimedia services relay on the IMS framework for the delivery. Theoretically ScaleNet could support other protocols like Overlay Networks or Peer-To-Peer (P2P), but IMS is the one used by the current implementation.

It is important to notice that the network itself is user-centric, and transparently handles identities by using Session Initiation Protocol (SIP). This eases supporting users with multiple devices; therefore applications do not have to worry about that part.

It is also important to define what a session means in this system. A session refers to the current use of a service, so for every service that the user is enjoying a session is created. For example, if it is viewing a movie but also talking on the Internet Protocol (IP) phone, there are two sessions at the same time.

The creation of a session implies that a new service is created, but it goes the other way around too. If a session is deleted, that service must stop. If the user ends the service, the session must be deleted. That means sessions have to be synchronized with the actual services.

A session is also linked to the device that the user is using. The system allows the copy and transfer of sessions to other devices that he owns, wherever it makes sense. Since the current implementation has also basic social capabilities, that session can also be transferred or copied to a user's contact. In the context of this application a user's contact is called "buddy". Figure 2.2 on the previous page lists some of the services that can be offered:

- Voice & Video Calls
- Mobile TV & Video on Demand (VOD)
- Massively Multiplayer Online Games (MMOGs)
- Internet Access

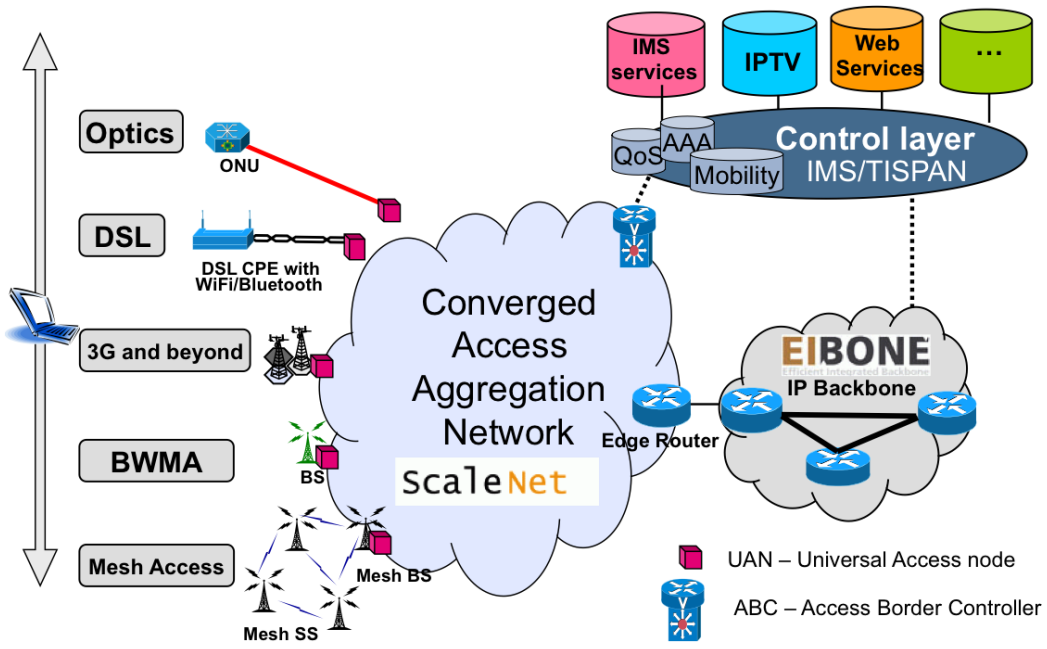
The work described in this document is primarily focused on the second application, i.e., video streaming. The idea is that the user can buy a video and play it anywhere using any supported device.

2.2.2 IMS Demonstrator

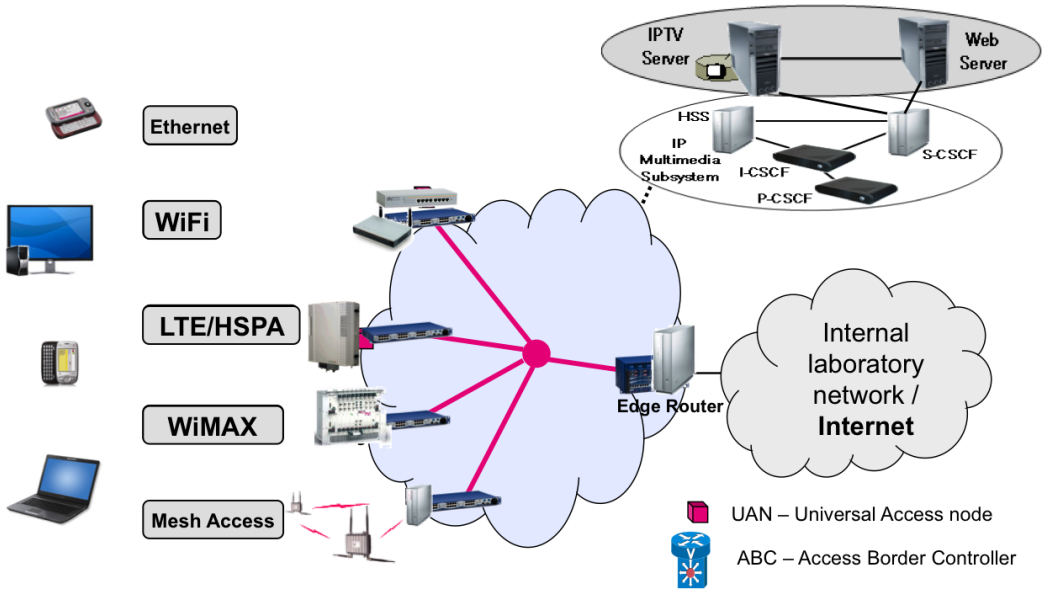
A logical view of the system is depicted in Figure 2.3(a) on the facing page, explaining the important nodes based on the capabilities needed. The information relevant to this project is contained in the upper right corner of the figure, the nodes behind the control layer.

In the offices of T-Labs in Berlin and Darmstadt there is a demonstrator with a working implementation of ScaleNet. That demonstrator is composed by several servers and a network infrastructure that enables access to the system using different network protocols and devices. In Figure 2.3(b) on the next page the actual network and hardware are exposed, replacing the same space as in the logical view (Figure 2.3(a) on the facing page).

Figure 2.4 on page 12 describes the setup in a better way and highlights the three different planes of the demonstrator. The developed web application is executed from the Web Server and the Application Server, since it



(a) System architecture



(b) Architecture of the demonstrator

Figure 2.3: IMS architecture

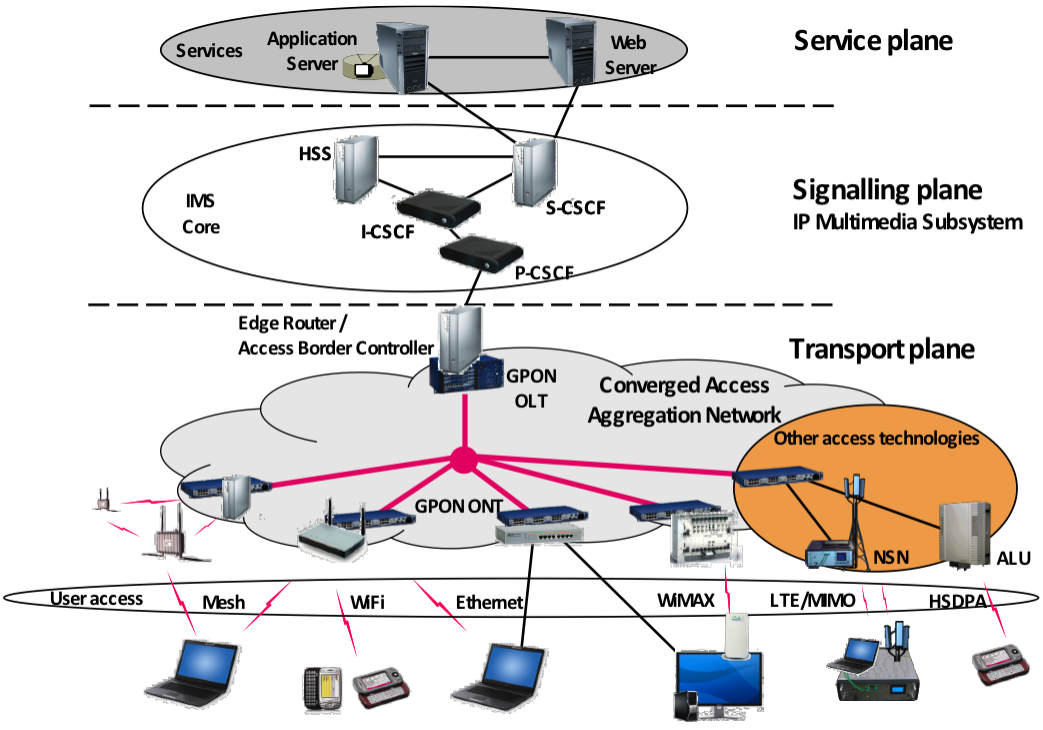


Figure 2.4: Setup of the demonstrator

belongs to the service plane. The signaling plane has also to be taken into account, because it communicates directly with the servers.

However, that is not the real deployment of the hardware used. Whether for convenience or efficiency, tasks are distributed between two main servers. This does not affect the logic of the system, since those tasks could be easily decoupled in an alternate deployment with more servers. Anyway, the interesting pieces of hardware for this project are:

IMS core This machine contains the IMS server*, but since the IMS load is not very high, it is responsible for other things. It acts as a Web Server (using Apache Web Server†) serving PHP applications. It is also the internal Domain Name System (DNS) server.

Application Server This is the Internet Protocol Television (IPTV) server, where the video content is streamed. It is also a Web Server, but it serves Java applications based on the OSGi framework‡.

*The IMS core is open source software from Fraunhofer FOKUS and it can be freely downloaded from: http://www.openimscore.org/

†http://httpd.apache.org/

‡http://www.osgi.org/

User Devices Devices intended for the user to access the services. There is a TV, a laptop and several phones. All of them run a custom IMS client that holds a connection to the servers, allowing the identification and adding IPTV and Voice over IP (VoIP) capabilities to those devices. In the last phase of the development, an iPhone was added for testing purposes.

This demonstrator contains several demo applications running. The interesting one for this project is the application that handles IPTV streaming.

2.2.3 Personal Network Administration Interface (PNAI)

The Web interface used for the management of sessions is called PNAI [3]. From this interface the user can obtain this information:

- All devices and registered in the system for that user and their online status.
- All buddies for that user and their online status.
- All multimedia sessions related to the user. This includes:
  - The sessions running on his devices, no matter who paid for that content.
  - The sessions running on devices from his buddies and started/-paid by that user.

Those are passive actions, but from that same view the user can initiate some operations to control the system. In Figure 2.5 on the following page all the available operations relating sessions are listed following a use case diagram.

In that diagram colors are used to differentiate the different kind of use cases covered. Also two visual marks (* and **) are added in case this is a copy in black a white. The meaning of the colors are explained according to this legend:

Green Available already in the main PNAI page.

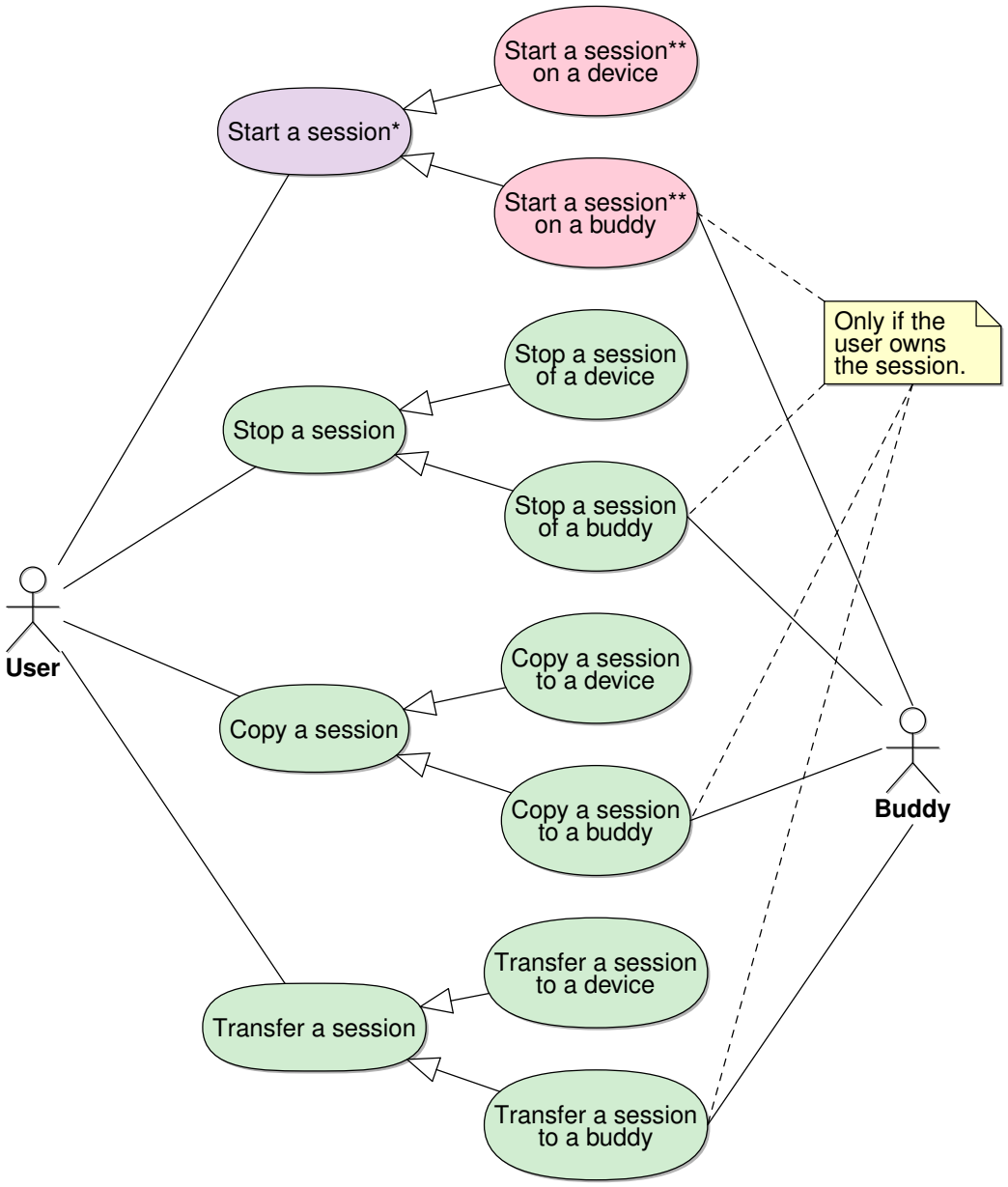


Figure 2.5: Use cases for the IPTV application

Purple (*marked with* *) Available in an individual page outside of the main PNAI page.

Red (*marked with* **) Not implemented.

As we can see, the main PNAI page has already a lot of functionality, but it can contain even more. Basically the actions available to that user in that page are:

- Terminate a session of a user device or of a buddy if the session is owned by that user.
- Transfer (handover) or copy (duplication) an existing session to a user device or to a buddy if the session is owned by that user. That is, if one buddy bought the content for us, we cannot transfer again that content to another buddy.

Beside of these session related operations, there are other management operations. For example, selecting which device is the default, adding/removing devices or adding/removing buddies. For this document they are not relevant since they remained untouched.

Figure 2.6 on the next page shows the old appearance of the main page for a logged user, before any work began. On the left side of the page there is the Device List, where the devices owned by that user are drawn. On the right side there is the Buddy List, where the user's buddies are listed. Finally, the trash bin is in the lower right corner of the Device List.

The devices that are offline are disabled and are drawn with a dimmed appearance. The buddies that are online are preceded by a green icon, while the ones that are offline are preceded by a red icon.

Devices or buddies that are online act as session containers. The reason for the devices to be so big is because inside them the current sessions are drawn. Besides the name of the content playing, session have an icon that changes depending on the type of session (video, audio, call, etc).

The user can interact with the sessions through the mouse using drag&drop. For example, the user can *grab* the icon he wants and drop it in another container to copy or transfer that session. That is a very visual and fast way to manage sessions.

Figure 2.6: Old PNAI page

When a user drops the session in another container, a menu will appear to ask the user if he wants to copy or transfer that session. The trash bin acts also as a container, but in a special way: when a session is dropped in the trash bin, that session is automatically deleted, with no menu involved.

Use Cases

In the following tables the current supported use cases are explained step by step. The first use case is detailed in Table 2.1, explaining the situation where the user wants to stop/delete the session in a device.

Table 2.1: Use case 1 – Stop a session of a device

Use Case 1	**Stop a session of a device**
Actor	System user
Precondition	A session is already running on a device, and it is showing in the PNAI interface inside of that device.
continued on next page	

Table 2.1: Use case 1 – Stop a session of a device (continued)

Use Case 1	**Stop a session of a device**
Postcondition	Session must terminate, i.e., the content must stop playing. The user must be notified with a popup and the session icon must be deleted from the PNAI interface.
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into the trash. **4.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **5.** The content stops playing. **6.** The popup disappears and the original session icon is deleted from the view.
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.
Alternate Path (A2)	**5c.** There is an error with the server and the content keeps playing. **6c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **7c.** Action is cancelled.

Figure 2.7 on the next page shows how the page looks when it is waiting for a response to the server for the previous use case. Since the user interface

Figure 2.7: Deleting a session in the old PNAI

does not block in the process, the communication between the front end and the back end must be asynchronous. The use case for terminating a session that a buddy is playing and that we own is very similar, as Table 2.2 exposes.

Table 2.2: Use case 2 – Stop a session of a buddy

Use Case 2	**Stop a session of a buddy**
Actor	System user
Precondition	A session owned by the user is running on a device, and it is showing in the PNAI interface near that buddy's name.
Postcondition	Session must terminate, i.e., the content must stop playing. The user must be notified with a popup and the session icon must be deleted from the PNAI interface. The buddy is *not* notified, the content stops without warning.
	continued on next page

Table 2.2: Use case 2 – Stop a session of a buddy (continued)

USE CASE 2	**Stop a session of a buddy**
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into the trash. **4.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **5.** The content stops playing. **6.** The popup disappears and the original session icon is deleted from the view.
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.
Alternate Path (A2)	**5c.** There is an error with the server and the content keeps playing. **6c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **7c.** Action is cancelled.

Tables 2.3 on the next page, 2.4 on page 21, 2.5 on page 23 and 2.6 on page 25 show how the user could copy or transfer a session to another device or buddy.

Table 2.3: Use case 3 – Copy a session to a device

Use Case 3	**Copy a session to a device**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface inside of that device/buddy. Also, there is another device online.
Postcondition	Session must be copied to that device, i.e., the content must be duplicated and played on that device. The user must be notified with a popup and the session icon must appear in the PNAI interface for the second device.
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into another device that is online. **4.** A popup menu appears where the user dropped the session, giving options to copy/duplicate the session, transfer the session or cancel the action. **5.** The user clicks on the copy/duplicate option. **6.** The popup menu disappears. **7.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **8.** The content starts playing on the destination device. **9.** The popup disappears and the same session icon appears inside of the destination device.

continued on next page

Table 2.3: Use case 3 – Copy a session to a device (continued)

USE CASE 3	Copy a session to a device
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.
Alternate Path (A2)	**5c.** The user clicks on the cancel option. **6c.** Popup menu disappears and action is cancelled.
Alternate Path (A3)	**8d.** There is an error with the server and the content is not duplicated. **9d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **10d.** Action is cancelled.

Table 2.4: Use case 4 – Copy a session to a buddy

USE CASE 4	Copy a session to a buddy
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface inside of that device/buddy. Also, there is another buddy online.

continued on next page

Table 2.4: Use case 4 – Copy a session to a buddy (continued)

USE CASE 4	**Copy a session to a buddy**
Postcondition	Session must be copied to that buddy, i.e., the content must be duplicated and played on the buddy's default device. The user must be notified with a popup and the session icon must appear in the PNAI interface near the name of that buddy. The buddy is *not* notified, the content plays without warning.
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into another buddy that is online. **4.** A popup menu appears where the user dropped the session, giving options to copy/duplicate the session, transfer the session or cancel the action. **5.** The user clicks on the copy/duplicate option. **6.** The popup menu disappears. **7.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **8.** The content starts playing on the buddy's default device. **9.** The popup disappears and the same session icon appears inside of the destination buddy.
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.

continued on next page

Table 2.4: Use case 4 – Copy a session to a buddy (continued)

USE CASE 4	**Copy a session to a buddy**
Alternate Path (A2)	**5c.** The user clicks on the cancel option. **6c.** Popup menu disappears and action is cancelled.
Alternate Path (A3)	**8d.** There is an error with the server and the content is not duplicated. **9d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **10d.** Action is cancelled.

Table 2.5: Use case 5 – Transfer a session to a device

USE CASE 5	**Transfer a session to a device**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface inside of that device/buddy. Also, there is another device online.
Postcondition	Session must be transferred to that device, i.e., playback must be stopped at the source and started at the destination device. The user must be notified with a popup and the session icon must appear in the PNAI interface for the second device.

continued on next page

Table 2.5: Use case 5 – Transfer a session to a device (continued)

USE CASE 5	**Transfer a session to a device**
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into another device that is online. **4.** A popup menu appears where the user dropped the session, giving options to copy the session, transfer/hand over the session or cancel the action. **5.** The user clicks on the transfer/hand over option. **6.** The popup menu disappears. **7.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **8.** The content stops playing on the source device. **9.** The content starts playing on the destination device. **10.** The popup disappears, the session icon is deleted from the view and created again inside of the destination device.
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.
Alternate Path (A2)	**5c.** The user clicks on the cancel option. **6c.** Popup menu disappears and action is cancelled.

continued on next page

Table 2.5: Use case 5 – Transfer a session to a device (continued)

USE CASE 5	**Transfer a session to a device**
Alternate Path (A3)	**8d.** There is an error with the server and the content is not transferred. **9d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **10d.** Action is cancelled.

Table 2.6: Use case 6 – Transfer a session to a buddy

USE CASE 6	**Transfer a session to a buddy**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface inside of that device/buddy. Also, there is another buddy online.
Postcondition	Session must be transferred to that buddy, i.e., playback must be stopped at the source and started at the buddy's default device. The user must be notified with a popup and the session icon must appear in the PNAI interface near the name of that buddy. The buddy is *not* notified, the content plays without warning.

continued on next page

Table 2.6: Use case 6 – Transfer a session to a buddy (continued)

USE CASE 6	**Transfer a session to a buddy**
Main Path (M)	**1.** User starts dragging the session icon. **2.** A copy of the session icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned session icon into another buddy that is online. **4.** A popup menu appears where the user dropped the session, giving options to copy the session, transfer/hand over the session or cancel the action. **5.** The user clicks on the transfer/hand over option. **6.** The popup menu disappears. **7.** A popup appears to notify the user that the action is in progress and the cloned session icon is deleted from the view. **8.** The content stops playing on the source device. **9.** The content starts playing on the buddy's default device. **10.** The popup disappears, the session icon is deleted from the view and created again inside of the destination buddy.
Alternate Path (A1)	**3b.** User drops the session into a blank space. **4b.** Action is cancelled.
Alternate Path (A2)	**5c.** The user clicks on the cancel option. **6c.** Popup menu disappears and action is cancelled.

continued on next page

Table 2.6: Use case 6 – Transfer a session to a buddy (continued)

Use Case 6	**Transfer a session to a buddy**
Alternate Path (A3)	**8d.** There is an error with the server and the content is not transferred. **9d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **10d.** Action is cancelled.

Figure 2.8: Duplicating a session in the old PNAI

Figure 2.8 shows how the popup menu is displayed to the user. It is a very simple menu with only three links, each of which correspond to an action.

Components

As explained in § 2.2.2 on page 10, the software is mainly executed in three machines: two servers and a client. Figure 2.9 on the next page shows

all the relevant components running inside those machines and how they interact with each other.

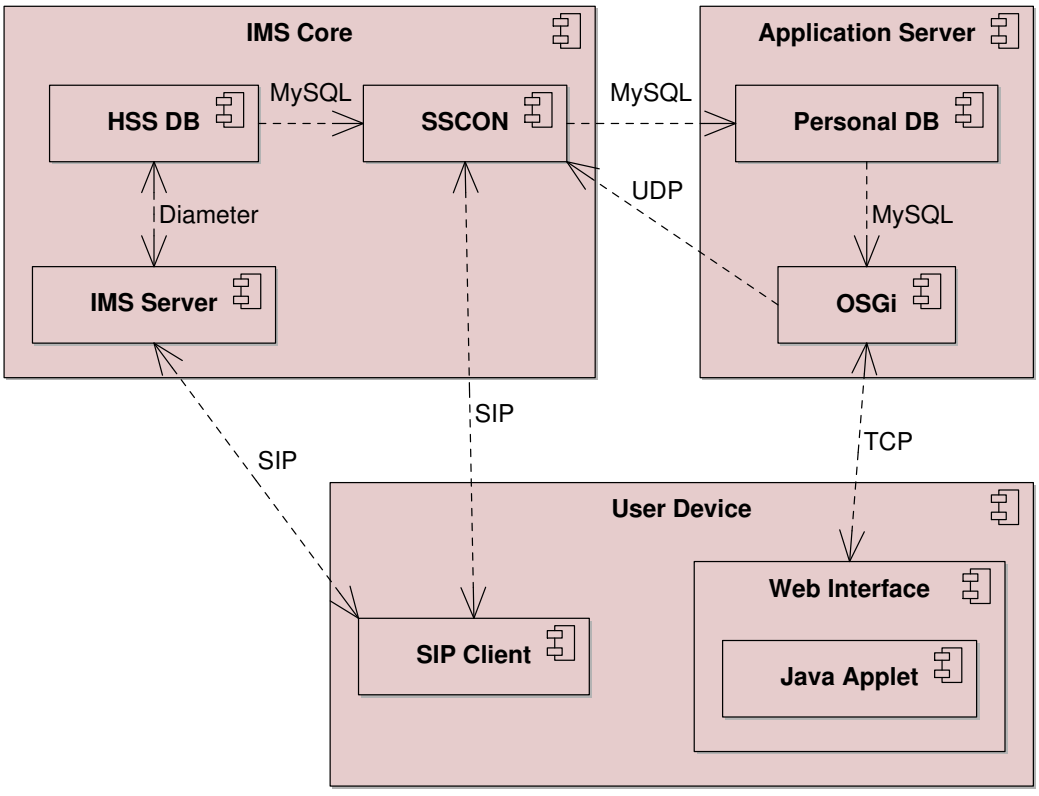


Figure 2.9: Component diagram for the old PNAI

That component diagram brings some interesting aspects about the system, although we are only interested in several of them:

- The SIP protocol is used for the IMS part, but this is not important for the work done, as both the server and the client were not touched.
- The Session Controller (SSCON) manages the sessions and acts as the bridge between the actual services and the web interface shown to the user. Actions from the web interface are notified through User Datagram Protocol (UDP) calls.
- In this diagram, though, there are two parts missing that are almost completely irrelevant for this document. These are the application server component that streams the content and the multimedia player installed in the user device. The protocols used between these two

components are not interesting for us, so for sake of simplicity, they don't appear here. The only important thing is that the SIP client controls the external player, so the web interface does not handle the video streaming.

- There are two MySQL databases in use:

  HSS DB This is the master database, and it is always up to date. It contains information about the user status, his buddy list and registered devices.

  Personal DB This is an additional database that depends on the HSS DB. This database gather all the information needed for the PNAI, since it contains all the relevant information from the HSS DB plus the session information obtained from the SSCON. Periodically, the SSCON polls information from the master database and then updates this slave database, so the information can be a bit outdated compared to the HSS DB. As stated in the diagram, the OSGi component grabs the information it needs from this database, by polling it periodically.

- Once the page is sent to the browser, the web interface continues talking with the server through a Transmission Control Protocol (TCP) socket without reloading the page. This goes both ways, since it is used for sending actions and receiving data following a push model (so no delays polling). Since, at the time of developing the original application, there was no way to get that using only basic web standards, it uses a Java applet to handle the socket communications.

Figure 2.10 on the following page shows the flow of the application in the scenario where the user wants to transfer a session from his device to one of his buddies. Blue lines belong to the main logical flow, while the yellow ones belong to the video streaming process. Other usage scenarios are very similar to this one, so they are ignored since this one explains well how and when they communicate.

As we can see, the web interface (written in JavaScript) talks back and forth with the Java applet using simple method calls, since all the code interface are directly available between them. Then the Java applet

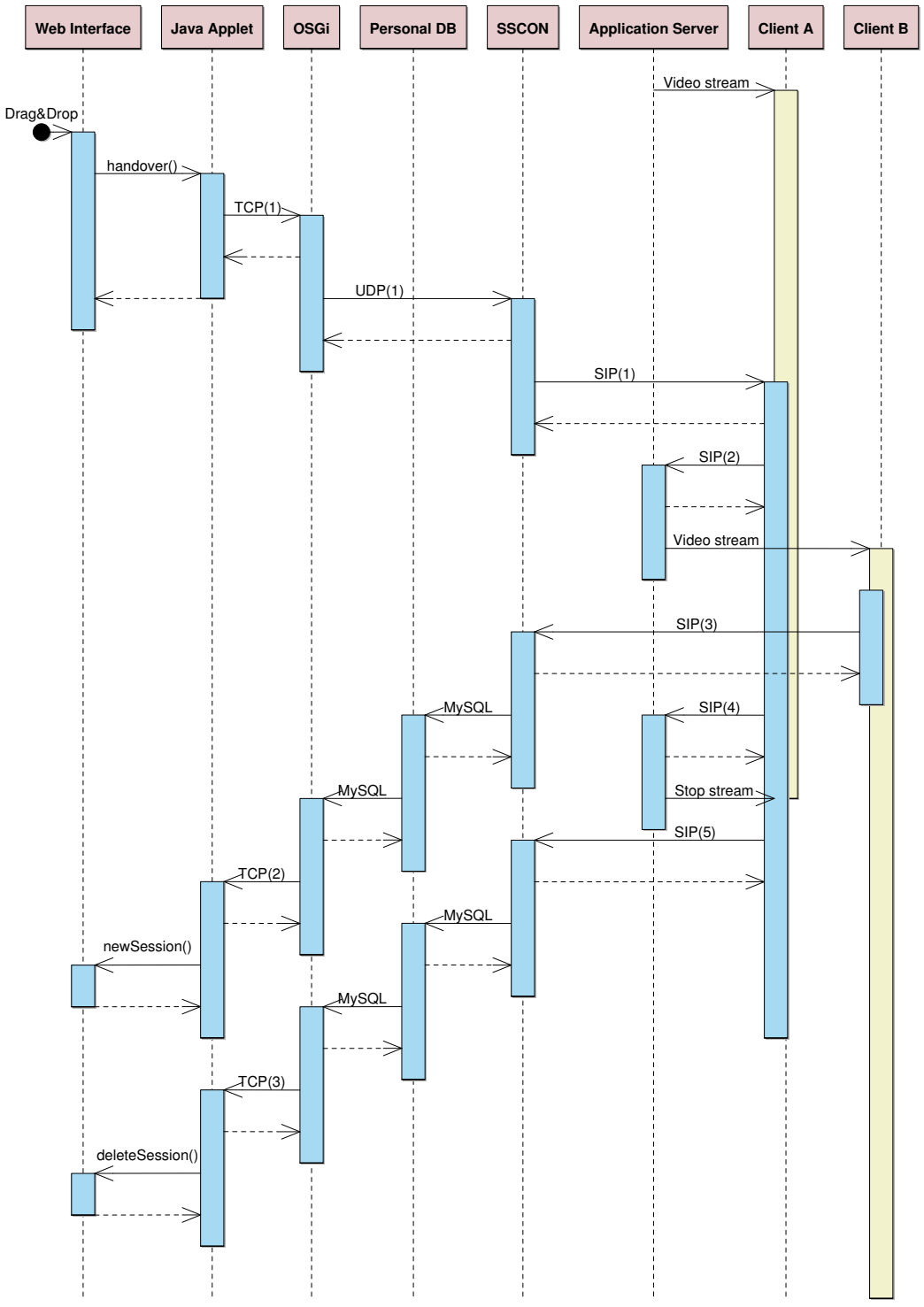


Figure 2.10: Sequence diagram for the old PNAI

translates those calls to strings with the method names and parameters and sends it to the OSGi using the TCP connection.

At the right part of the diagram, it is clear that the sessions are controlled using SIP messages. Given that ScaleNet is a very decentralized network by design, the SSCON delegates to the SIP clients running in the devices all the talking with the Application Server.

It is interesting to note that everything is mostly asynchronous, so there is not a lot of calls that block. Therefore the use of threads and callbacks is widespread in all components.

Personal DB

The database that is directly used by the PNAI is the Personal DB. The most important table is the `current_session` table, where is all the information about the sessions that are currently active. Table 2.7 explains all the fields in this table, and Table 2.8 on the following page details an example entry.

Table 2.7: Current session table architecture

Field name	Description
`id`	Identifier, auto-increment integer
`impi`	Private identity of user
`impu`	Public identity of user
`callid`	Unique number to identify session details
`partner`	Next party of session (AS identity if client to AS or impu of partner if client to client)
`as`	Application Server identity (client to AS) or NULL (client to client)
`ip`	IP address for impu
`initiator`	impu who sends the INVITE message

continued on next page

Table 2.7: Current session table architecture (continued)

Field name	Description
`owner`	impi who needs to pay for the session. Usually the impi who sends the `invite` message, but it can be the impi who sends the `refer` message in case of transfer/duplication
`session_name`	Name of the session
`type`	Type of the session (audio/video)
`bw`	Bandwidth of the session
`source`	Uniform Resource Locator (URL) of the source
`lov`	Type of video transmission (live/video on demand/tv)
`cid/did/tid`	Integers related to Quality of Service (QoS) parameters
`session_flag`	0 if it is a normal session 1 if it is a transferred session 2 if it is a duplicated session

Table 2.8: Current session table example

Field name	Example value
`id`	3
`impi`	deni@imusu.mobile.dtrd.de
`impu`	mda.deni@imusu.mobile.dtrd.de
`callid`	783457644
`partner`	as@imusu.mobile.dtrd.de or tv.hahn@imusu.mobile.dtrd.de

continued on next page

Table 2.8: Current session table example (continued)

Field name	Example value
as	as@imusu.mobile.dtrd.de
ip	19.168.5.92
initiator	mda.deni@imusu.mobile.dtrd.de or as@imusu.mobile.dtrd.de
owner	deni@imusu.mobile.dtrd.de
session_name	NASA
type	video
bw	5000
source	http://appserver:9000
lov	video on demand
cid/did/tid	3/5/12
session_flag	0

Other important table is the `user_status` table, that lists all the users in the system and some basic information about them. Table 2.9 explains all the fields in this table, and Table 2.10 on the following page details an example entry.

Table 2.9: User status table architecture

Field name	Description
id	Identifier, auto-increment integer
impi	Private identity of user
impu	Public identity of user
impi_id	Unique id to identify impi

continued on next page

Table 2.9: User status table architecture (continued)

Field name	Description
impu_id	Unique id to identify impu
status	Status of impu: 1 (online), 0 (offline)

Table 2.10: User status table example

Field name	Example value
id	1
impi	deni@imusu.mobile.dtrd.de
impu	laptop.deni@imusu.mobile.dtrd.de
impi_id	67
impu_id	35
status	1

Finally, there is a third table that handles the relationships between friends called web_buddylist. It is a very simple table modeling a classic many-to-many relationship, but anyway Table 2.11 explains all its fields, and Table 2.12 on the facing page details an example entry.

Table 2.11: Buddy list table architecture

Field name	Description
id	Identifier, auto-increment integer
impi_id	Unique id to identify impi (identical to impi_id of user_status table)

continued on next page

Table 2.11: Buddy list table architecture (continued)

Field name	Description
buddy_impi_id	Unique id to identify the buddy impi

Table 2.12: Buddy list table example

Field name	Example value
id	2
impi_id	56
buddy_impi_id	67

These tables are not changed during the development explained in this document, neither the software that access those tables directly. However, it is interesting to know the kind of data they have because indirectly it is the same data we are going to process.

Messages

Of all the messages sent inside the system, the most important ones for us are sent though the TCP socket. These are processed and generated by the web interface and the OSGi backend, but they follow a different format depending which component sends the message.

Messages generated by the backend consist of serialized objects following a very simple format. There are three kind of data objects that can be sent over the wire: devices, buddies and sessions.

A serialized object is a string that starts with the type of the object (`device`, `buddy`, `session`), followed by the vertical bar character '|' as delimiter. Then the attributes for that object are appended one by one, separated by the same delimiter. If the values are not strings, they are converted directly, for example a boolean with value `true` will be passed as the string `"true"`.

The client does not know when a transfer or duplication happens, it is only notified of creation and deletion of things. When a status update happens, such as a device going online, it is notified as the creation of an object. Therefore the client must keep track of the objects received and realize that it is an update of a previously created object.

For example, as seen in Figure 2.10 on page 30, when a transfer happens the interface received two commands, first creating a new session and then deleting the original session.

To notify that a new object needs to be created in the view, the backend just sends the serialized object, without adding anything else. To notify that an existing object has to be deleted from the view, the string is the same but preceded by the text `"deleted|"` (that is, *deleted* and the delimiter).

Table 2.13 comprises all the different messages that can be sent from the OSGi backend to the Java applet with examples.

Table 2.13: Format of the notifications sent to the applet

Notification	**Format & Example**

Create/update device	`device|`*`impi`*`|`*`impu`*`|`*`online`*
	`device|hahn@imusu.mobile.dtrd.de`
	`↪|tv.hahn@imusu.mobile.drtd.de|true`

Delete device	`deleted|device|`*`impi`*`|`*`impu`*`|`*`online`*
	`deleted|device|hahn@imusu.mobile.dtrd.de`
	`↪|tv.hahn@imusu.mobile.drtd.de|false`

Create/update buddy	`buddy|`*`id`*`|`*`name`*`|`*`online`* `buddy|3|hahn@imusu.mobile.dtrd.de|false`

Delete buddy	`deleted|buddy|`*`id`*`|`*`name`*`|`*`online`*
	`deleted|buddy|3|hahn@imusu.mobile.dtrd.de`
	`↪|false`

continued on next page

Table 2.13: Format of the notifications sent to the applet (continued)

Notification	**Format & Example**

Create/update session	`session|id|type|name|owner|initiator|impi`
	`↪|impu|icon`
	`session|3|video|NASA`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|laptop.hahn@imusu.mobile.dtrd.de`
	`↪|http://imusu.mobile.dtrd.de/img/icon.png`

Delete session	`deleted|session|id|type|name|owner`
	`↪|initiator|impi|impu|icon`
	`deleted|session|3|video|NASA`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|hahn@imusu.mobile.dtrd.de`
	`↪|laptop.hahn@imusu.mobile.dtrd.de`
	`↪|http://imusu.mobile.dtrd.de/img/icon.png`

Going back to Figure 2.10 on page 30, the message sent in `TCP(2)` was a *Create/Update session* notification, while the message sent in `TCP(2)` was a *Delete session* notification. In both cases, the Java applet just parsed the messages and passed the arguments to the right JavaScript callbacks.

On the other hand, messages sent by the Java applet are requests from the user, for actions that he wants to complete relating sessions. These actions can be the deletion of a session, its transfer or its duplication.

It does not matter which kind of origin (device or buddy) or destination it goes, because dealing with unique SIP identifiers (`impi`) makes sure that every element is treated equally.

They follow a query string format, which is the usual way to pass data as part of a URL. This string is passed directly to the TCP socket, without any additional encoding.

Table 2.14 on the following page lists the different messages that can be

sent from the Java applet to the OSGi backend with examples.

Table 2.14: Format of the requests sent from the applet

Request	**Format & Example**

Copy session	`event=duplicate&uid=`*`uid`*`&source=`*`source`*
	`↪&sid=`*`sid`*`&destination=`*`target`*
	`event=duplicate`
	`↪&uid=hahn@imusu.mobile.dtrd.de`
	`↪&source=laptop.hahn@imusu.mobile.drtd.de`
	`↪&sid=458215`
	`↪&destination=steffen@imusu.mobile.drtd.de`

Transfer session	`event=handover&uid=`*`uid`*`&source=`*`source`*
	`↪&sid=`*`sid`*`&destination=`*`target`*
	`event=handover`
	`↪&uid=hahn@imusu.mobile.dtrd.de`
	`↪&source=laptop.hahn@imusu.mobile.drtd.de`
	`↪&sid=458215`
	`↪&destination=tv.hahn@imusu.mobile.drtd.de`

Stop session	`event=delete&uid=`*`uid`*`&source=`*`source`*`&sid=`*`sid`*
	`event=delete&uid=hahn@imusu.mobile.dtrd.de`
	`↪&source=laptop.hahn@imusu.mobile.drtd.de`
	`↪&sid=458215`

`TCP(1)` in Figure 2.10 on page 30 is a typical *Transfer session* request, generated by a Java applet after the JavaScript requested it upon a user's action. Additionally, `UDP(1)` follows the same format and `SIP(1)` contains the same information but in SIP format. Other messages from that figure are not explained with more detail because the pieces of code that deal with MySQL or SIP are in other modules apart from the web application.

Java Applet Codebase

The same codebase (classes, resources, etc.) is shared between the OSGi backend and the Java applet, taking advantage of the fact that they are written in the same language. Though this does not mean that the final executables are the same, since two bundles are generated, one for each purpose. More details about how OSGi and Java applets work in general are discussed on § 2.4 on page 53.

Figure 2.11 on the next page shows all the packages involved in the Java applet. All classes exclusively related to the OSGi backend are ignored, since they are not important for this work.

This diagram contains two packages: `Applet` and `Model`. The first one encloses all the logic needed to contact the socket in the backend (`BackendLink`), and the Java applet itself with the interface available to the JavaScript codebase (`ScalenetApplet`).

From those classes it is easy to identify the three external parameters needed by the Java applet: the user identifier (SIP format, email-like), and hostname and port used by the backend (where the socket is reached).

The `Model` package, shared with the backend codebase, comprises all the data model objects. This includes utilities to create objects from strings, following the formats explained in Tables 2.13 on page 36 and 2.14 on the preceding page. The attributes for each class object are the same used in those tables.

To create a `Buddy`, `Device` or `Session`, a factory pattern is used, calling the static method `fromString`* of the class we want to create an object from. For some reason, to create a `Request` first the object is created and then the string is parsed to fill all the attributes. In any case a string is the preferred way to specify the attributes of an object.

JavaScript Codebase

Finally, we get to the old JavaScript codebase, the main place where the effort of this work was focused. The code resides in the resources folder of the OSGi bundle, where all the static public code is thrown: HTML, CSS,

*For technical reasons the method `fromString` could not be underlined in the diagram, as the UML specification recommends for static methods/attributes. To bring attention to this important quirk, the method's name is surrounded by underscores.

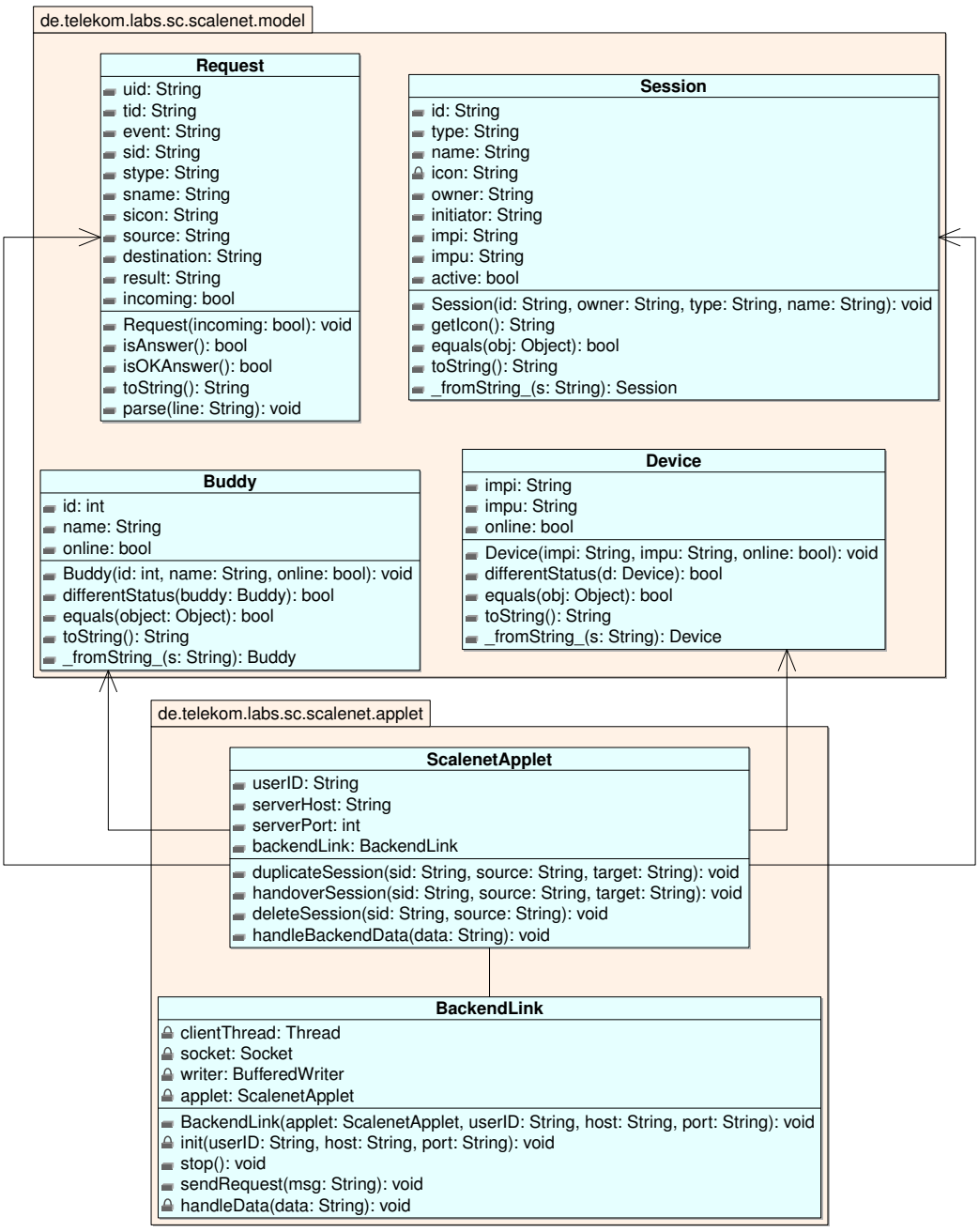


Figure 2.11: Class diagram for the Java applet

JavaScript, images, etc. When requested by a browser, these files are served by the OSGi-based server, not Apache.

JavaScript classes are organized in several source files, each file acting as if they were traditional packages. As discussed on § 2.6 on page 77, *class* is not the right term to refer to a JavaScript object, but it will be used through this document for simplicity's sake.

Additionally, in the old codebase the use of global variables is wildly used, and they are quite important to understand how the application works. Since the Unified Modeling Language (UML) language is designed for Object-Oriented Programming (OOP), there is no easy solution to specify those variables in the same diagram. However, a workaround is to directly copy how the Document Object Model (DOM) works and put all those variables under the global `window` object.

Figure 2.12 on the next page shows a diagram with all the classes involved and the relationships between them. Figure 2.13 on page 43 completes the picture adding the global object and the relationships between it and the custom classes.

The whole code revolves around two abstract classes: `Container` and `Session`. Technically, they are not *abstract* because the JavaScript language does not offer this construction. In reality, there are no objects created directly from these classes, but are created from subclasses that extend these classes.

A `Container` is anything that can contain a session, or more specifically, any element where the user can drop a session. All containers are stored in a `Hash` (`containerList`), where the key is the name of that container. Each container have several slots where the sessions can go, this means that the container can hold up to that number of sessions at the same time. The class provides methods to attach or detach session to those slots.

The DOM representation for a container is a `div` block. A background image with the real appearance of the container is drawn inside of this `div` in a `img` element. Each session the container *owns* is drawn inside of this image, so the `div`s for those sessions are children of the container's `div`.

Each slot stores the coordinates where the session can be drawn, so they need to be calculated when the container's `div` is created. They also keep track of the session they belong to and the DOM representation of that

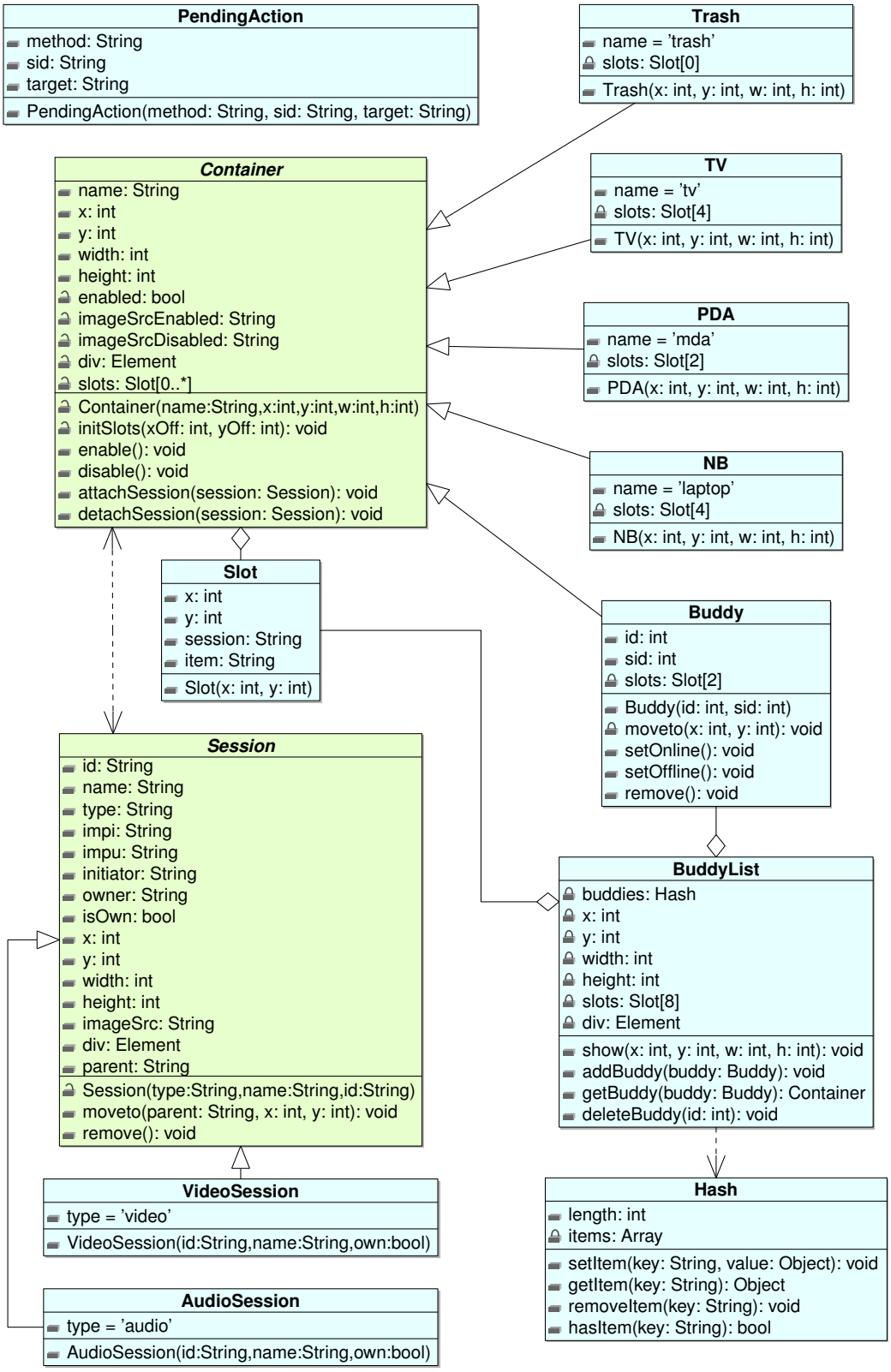


Figure 2.12: Class diagram for the old PNAI

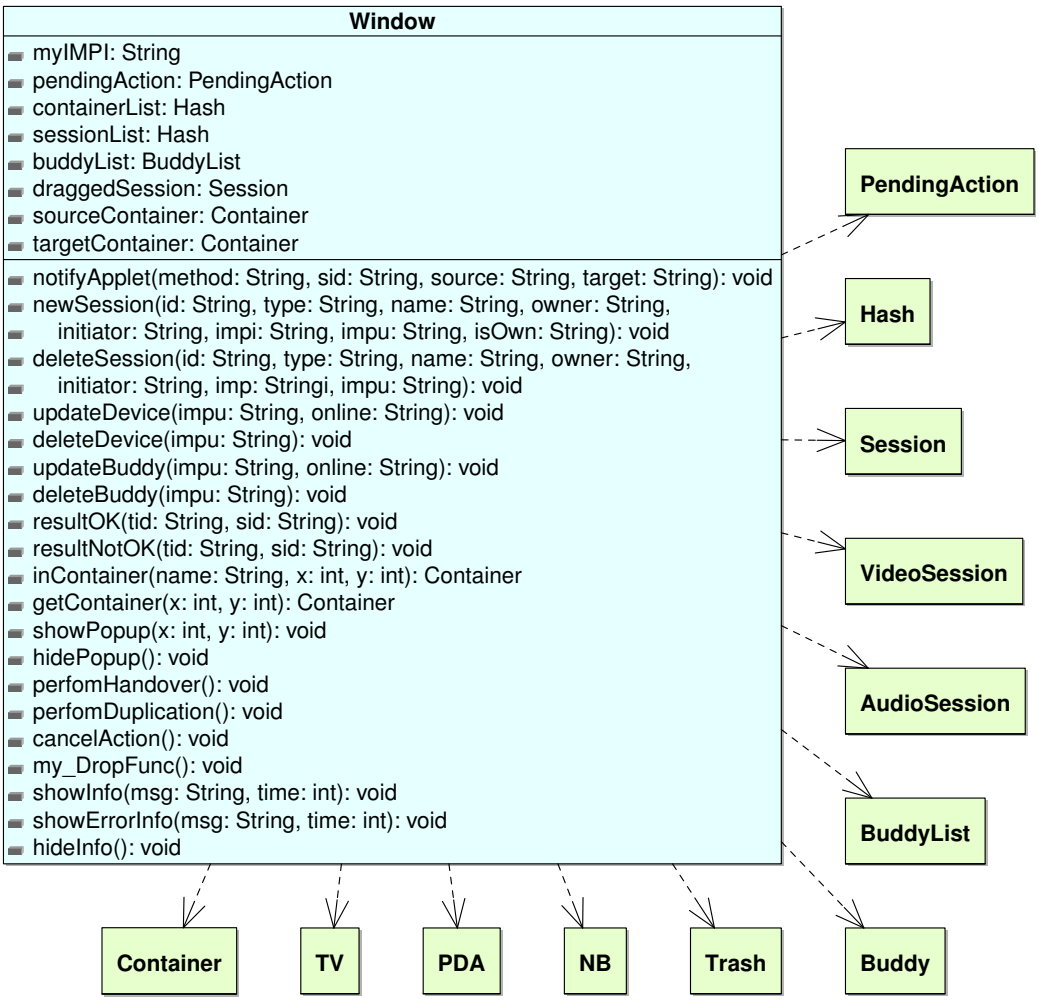


Figure 2.13: The global JavaScript object in the old PNAI

session (is the slot is not empty).

Every container can be enabled (online) or disabled (offline) at any time. If it is disabled, it cannot have any session, and the background image changes to a more grayish image to reflect this new state to the user.

There are three types of containers: devices, buddies and the trash. Since the number of devices is fixed in this interface for simplicity's sake, it is assumed that the user have a Television (TV), a Notebook (NB) and a Personal Digital Assistant (PDA).

Each one is linked to a realistic image icon used by the interface. The TV and the NB have four slots and the PDA has only two slots, but there is no such constrain in the backend so this is only a cosmetic issue.

The trash is also considered a container, because it can *receive* sessions. To delete a session there is not button, the user has to explicitly drag the session and drop it in the trash. Therefore, instead of storing them, the trash removes them from the interface.

A buddy is a special container, in that there can be more than one. In the interface there is an icon for that buddy, but it behaves different from the devices. That icon does not *contain* the sessions, they are displayed at the right side of the buddy's name. Each buddy can hold up to two sessions at the same time. For the rest, it is pretty much the same as a generic container.

All the buddies are stored in a `BuddyList` (`buddyList`), apart from the rest of the containers. The DOM representation for this object is the sidebar with the buddies, another `div` block.

A `Session` in this page is simply a current session in the system. Its DOM representation it is also a `div` with an image (a generic icon, not a thumbnail) and the name of the content playing. All sessions are stored in a `Hash` (`sessionList`). There are two kind of sessions: `VideoSession` and `AudioSession`, and they are mostly the same except for having different icons.

The `moveto` method is the one in charge, not only of moving the session to a new container, but also of creating the visual representation for that session. Unlike the containers, where the `init` method creates the `div`, sessions are not showed in the screen until the parent is explicitly set with this `moveto` method.

Besides this custom code, there is one external dependency for handling

drag&drop in a convenient way. The library is `wz_dragdrop`*, written by German author *Walter Zorn*. It comes from the early days of JavaScript, trying to provide a cross-browser solution for this problem. This library has been discontinued, but at the same time a lot of modern alternatives offer a better, simpler and cleaner approach.

To work with this library we have to explicitly set the items we want to be able to drag, in this case the sessions. Then, function callbacks are available for several events, and we can redefine those functions to decide what to do next.

The `my_DropFunc` function is the one called by this library when the user drops the session into something (or, simply, stops dragging the session). This is the only one that needed to be redefined, and basically it does this:

- Get the id from the object that user has dropped.
- Get the session with that id (`draggedSession`).
- Get the container where the session was before (`sourceContainer`).
- Calculate the container where the session has been dropped using its coordinates and the function `getContainer` (`targetContainer`).
- If the target container is the trash, pass the action to the Java applet using the `notifyApplet` function.
- If there is a valid target container, show a popup menu giving the user the option to transfer the session, copy it or cancel the action.
- If there is a problem with the target or no target is chosen, cancel the action and move the session icon back to the original container.

Later, when the user selects an option, three scenarios can happen:

Transfer session The `performHandover` function is called. In this function, the popup is closed and the Java applet is notified using the `notifyApplet` function. Until the backend answer, a panel with information (a `div`) is shown using the `showInfo` function, and the

*Original site seems broken, but a copy is available on:
http://gualtierozorni.altervista.org/dragdrop/dragdrop_e.htm

information about the action is stored in a `PendingAction` object (`pendingAction`).

Copy session The `performDuplication` function is called. This does the same as the previous case, but using another method name.

Cancel action The `cancelAction` function is called. Here, simply the popup menu is closed and the session icon is moved back to its original container.

After a while, the server will receive the request, process it and answer back to the Java applet. Then the Java applet will trigger the callbacks accordingly to the action, to create/update/delete a session. These callbacks (`resultOK` and `resultNotOK`) simply update the DOM according to the new information and the pending action, attaching/detaching the session to/from the correct containers and hiding the info panel.

Additionally the `newSession` function should be called when the user wants to duplicate a session. From the point of view of the web interface, a duplicated session is completely unrelated to the original one, since the id is different. So a full new session with new data and icon should be created.

There are other callbacks that the Java applet may called at any time (and a lot at the load of the application): `newSession`, `deleteSession`, `updateDevice`, `deleteDevice`, `updateBuddy` and `deleteBuddy`. Their names are very straightforward, and all of them update the stored data with the new information, changing the DOM accordingly. Since the devices in the screen are prefixed, the `createDevice` and `deleteDevice` functions act differently, only changing the online status of the device but never creating or deleting existing devices.

Finally, there is also some JavaScript code to setup the page, analogous to the *main* function in other languages like C or Java. Listing 2.1 on the next page shows this setup code.

In this code, first of all the id of the user is set. This is a piece of code written dynamically on the fly by the Java server, and it is obviously different for every user. Then the devices are created. As stated before, it does not matter the real devices really owns the user, for this demonstrator it is assumed that the user has three devices. By default the devices are disabled, because we do not know at the moment if they are online or not.

Listing 2.1: Setup code

```
var tv = new TV(40, 20, 380, 240);
tv.disable();
var laptop = new NB(40, 350, 380, 240);
laptop.disable();
var mda = new PDA(470, 20, 200, 300);
mda.disable();

var trash = new Trash(520, 430, 100, 150);
trash.enable();

buddyList.show(620, 20, 400, 600);
```

Then the trash and the buddy list (sidebar) are created. For each of the previous elements, their coordinates and dimensions have been specified statically. Those values were found by trial and error and hardcoded in the page.

2.2.4 IPTVplus and Other Pages

From the main PNAI page the user can control the sessions that are already created but, how can he create new sessions? Another page called IPTVplus lists all the multimedia services available to the user in categories, with thumbnails, descriptions, prices and buttons to buy that content. Figure 2.14 on the following page shows how the page looks.

Basically, the user clicks on the button *Start* to buy a content, then a popup appears to confirm the selection. If confirmed, the user is *charged* and the content starts playing in his default device.

From the user perspective this could be handled more elegantly, since the functionality is split between IPTVplus and the PNAI. One of the goals of this work is integrating that functionality directly in the main PNAI page.

This IPTVplus application resides in a directory called `scalenet` in the Apache public folder. The front page from where the user accesses all the ScaleNet applications, and therefore also the PNAI, is in that folder. All

Figure 2.14: Old IPTVplus page

these pages are written in PHP.

Figure 2.15 on the next page lists all the relevant PHP files in this directory. Some folders and files have been omitted because they are not relevant to this application.

The front page is in `index.php`, and it just contains several links to the sub-applications, that are in the `sub` directory. In the `includes` directory there are two files used by almost every application: `auth.inc.php` for authentication purposes and `db.inc.php` for connecting to the HSS DB.

The `IPTVplus.php` page is the main page for the IPTVplus application. This page includes `inhalt.php`, where most of the actual code is written. In short, the output of this subpage is just a list of all available videos, ordered by categories, as seen in Figure 2.14.

Internally `inhalt.php` gets the content in a very particular way. Instead of querying the database directly, it connects to an additional socket at `webportal.imusu.mobile.dtrd.de:7001` to retrieve the list. It does not matter which component really queries the database (the SSCON) or how, because that was not changed during this work.

```
scalenet/
├── index.php
└── sub/
    ├── includes/
    │   ├── auth.inc.php
    │   └── db.inc.php
    ├── iptvplus/
    │   ├── c2d.php
    │   ├── inhalt.php
    │   └── popup.php
    ├── IPTVplus.php
    ├── mobile/
    ├── mobile.php
    ├── personal/
    │   └── sessions.php
    └── personal.php
```

Figure 2.15: Old PHP directory

To that socket it sends the string `list` followed by a carriage return (`\r\n`). That command responds with the information of all content items, one by one. For each content it sends in order the following information, separated by a carriage return: `type`, `content`, `img`, `text`, `lov`, `priority`, `price`, `description` and `quality`. The end of the transmission is marked by the raw string `c2d`.

The very PHP code is rather inefficient, because it actually asks for the content four times, one for each category. That is, it opens the socket, send the command and receive the list four times. Each time it discards all the videos from the other categories, instead of only asking one and then ordering the results before *printing* the list.

As seen in Figure 2.14 on the preceding page, the list shows the title for each content (`text`), its icon, its description and its price. A button with the text *Start* allows the user to buy that content, by opening a popup to `c2d.php` if the content is free or to `popup.php` if it is not free.

`c2d.php` receives four `GET` parameters: the type of the stream, the desired quality, the id of the content and the impu of the destination device/user.

The link to this PHP file looks like this:

`.../c2d.php?type=`*`type`*`&quality=`*`quality`*`&content=`*`content`*`&`

`↪impu=`*`destimpu`*

This script sends a command to the SSCON to start playing that content in that device. It opens a socket exactly like in `inhalt.php` and sends the string `refer` followed by the parameters separated by spaces: the destination, the SIP id of the Application Server, the type, the content id and the quality. The end of the command is marked by a carriage return.

`popup.php` simply contains a confirmation page asking the user if he really wants to pay for that content, and redirects the user to the `c2d.php` if he confirms it. It receives two `GET` parameters: the price and the link to the proper `c2d.php` page (with the needed parameters already encoded in that URL). The link to this PHP file looks like this:

`.../popup.php?price=`*`price`*`&link=`*`linktoc2d`*

`mobile.php` and the files contained in the `mobile` folder have a mobile version of some of this applications. This is a very limited version and it does not contain a PNAI page, so this files where not even touched in this work.

`personal.php` is the interface that controls some things related to the user. In the `personal` folder there are several subpages that can be embedded in `personal.php` depending on a `GET` parameter. One of them is `sessions.php`, simply a wrapper for the PNAI page, and it consists of an iframe redirecting to the proper path in the OSGi server. Therefore the PNAI page is integrated in this portal.

There are other pages available from the same portal, some of them refers to other services (like controlling/monitoring a Mesh network) but others are configuration pages for the user. For example, in the `personal` directory there are pages to control the buddy list (add/remove), control the device list (add/remove/edit), etc. Since those pages were left untouched by this work, they are not explained.

2.3 Server Programming Language: PHP

PHP* is one of the languages used to build web applications in ScaleNet, and it is one of the most popular languages for building web applications in general. Not only most of the portal is written in this language but, eventually, the PNAI interface will be ported from an OSGi bundle to a PHP script.

2.3.1 History

Originally, PHP was created in 1995 by *Rasmus Lerdorf* as a set of Common Gateway Initiative (CGI) scripts written in C that parsed HyperText Markup Language (HTML) files. The goal was being able to call specific C routines and show its output when a page was visited, by directly embedding the code in the HTML source. In the next years this open source project became a full-fledged parser, creating a new generic language.

In 1997 *Zeed Suraski* and *Andi Gutmans* rewrote that parser to include more functionality and released it as PHP 3. Then, specially after PHP 4, the language started to be widely used. PHP has gained a lot of popularity for web development projects and it is used in big websites like Yahoo, Facebook and Wikipedia.

Figure 2.16: PHP logo

In 2004 PHP 5 was released, adding OOP capabilities to the language. However, it is compatible with PHP 4 scripts, so it is optional to work in an Object-Oriented way. This is the most recent version and it is the one installed in the IMS demonstrator.

2.3.2 Quick Overview of the Language

A PHP file is basically an HTML file with the `.php` extension that it is usually stored in the public folder of the server (Apache, Internet Information

*http://www.php.net/

Services (IIS) or other supported server). When that page is requested, the server calls the PHP module, that parses that file and returns a normal HTML page. Therefore it is a interpreted language, so no compilation is needed.

The interesting things happen within its delimiters (usually <?php and ?>), where the PHP code is written. Outside of these delimiters the text is treated as a normal HTML soup and therefore not processed. Listing 2.2 shows an example of PHP code embedded within HTML code and Listing 2.3 on the next page shows the resulting HTML output.

Listing 2.2: PHP code embedded within HTML code

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
  <?php
  for ($i = 0; $i < 5; $i++) {
    echo "<p>Hello World " . $i . "</p>\n";
  }
  ?>
  </body>
</html>
```

Its syntax is very similar to C, with equivalent constructions (if conditions, for and while loop, functions, etc). Some notable exceptions are that variables must start with a dollar sign character ($), and that a dot is used for concatenating strings instead of the most traditional plus sign.

Variables are dynamically typed, so the programmer does not need to specify types. A variable's type is determined by the context in which that variable is used, so any variable can hold different types during an execution.

One of the strengths of PHP is the wide range of utility functions bundled in the processor and in additional extensions usually included in distributions. Although objects are supported in PHP 5, it is not discussed here because the scripts in ScaleNet do not use them.

Global variables like `$_GET`, `$_POST` or `$_SERVER` offer access to informa-

Listing 2.3: Resulting HTML code

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
  <p>Hello World 0</p>
  <p>Hello World 1</p>
  <p>Hello World 2</p>
  <p>Hello World 3</p>
  <p>Hello World 4</p>
  </body>
</html>
```

tion sent from the browser, so it is commonly used to pass parameters to a PHP script (through the URL or forms in the page). On the other hand `$_SESSION` and `$_COOKIE` allow to store some data through the same *visit*, even across different scripts.

Since the main goal of the language is generating HTML content, the most common functions are ones that *print* text in the output, like `echo`. Because MySQL is quite popular in web development, there is an extension with functions to interact with those databases.

Other commons functions are `include` (to, ehem, include other PHP source files), `isset` (to know is a variable is set), `die` (to terminate the execution of the script at any moment), `header` (to set HyperText Transfer Protocol (HTTP) headers) and diverse string/array manipulation functions.

2.4 Server Programming Language: Java

Nowadays, Java is the most popular programming platform in corporative environments. In ScaleNet it is used in an OSGi module that it is always running in the background providing real-time communication with the browser, besides a Java applet. In the end, no Java code was written for this

project, since one of the goals was to move the files out of the OSGi bundle to the PHP scripts folder.

2.4.1 History

James Gosling originally developed the language at *Sun Microsystems* and released it to the public in 1995, within a initiative called *Green Project* started in that company in 1991.

Figure 2.17: Java logo

At first it was targeted at the digital cable television industry, but its true potential revealed to be the Internet, a much more dynamic platform. It became rapidly popular for its promise of running anywhere, and even more after the most used browsers added support for Java applets.

The main difference with the existing languages was the Java Virtual Machine (JVM). It allowed to compile a program in an intermediate byte code that can run on any Java supported device, independently of the underlying architecture.

In 1998 Java 2 was released, with a Java plugin and multiple versions targeted at different kind of scenarios (mobile devices, enterprise applications, limited devices, etc). Since then every two years a new version was released until reaching Java 6 in 2006, adding each time new capabilities to the language.

Starting in 2006, *Sun* published Java's source code under the General Public License (GPL). Now, for the most part, it remains as free software and *Oracle*, the current owner of the trademark, seems to be continuing the same strategy.

2.4.2 Quick Overview of the Language

According to Sun*, there were five primary goals in the creation of the Java language:

- It should be *simple, object oriented, and familiar*.
- It should be *robust and secure*.
- It should have *an architecture-neutral and portable environment*.
- It should execute with *high performance*.
- It should be *interpreted, threaded, and dynamic*.

The syntax itself is very similar to C, the main difference is that the code is organized around classes following OOP. By design it does not have any remarkable syntax anomaly, and the usual suspects are all there (if, for, while, etc).

It is strongly typed, and every variable needs to be declared with its type before using it. Except the primitives types, everything is an object, which incidentally leads to an increased verbosity.

As said before, all code needs to be compiled. The resulting byte code is not linked to any specific hardware, but to the JVM. This means that the compiled code is compatible with every supported platform, without any additional work, from a computer running Windows to a mobile phone.

The entry point for a Java applications is the `main` method of the main class. The main class is usually declared outside of the Java code in a manifest file. The most common way of packaging a program is using a Java Archive (JAR) file, essentially just a ZIP file containing the compiled code and a manifest.

All classes are organized in packages, each one focused in a different issue. Like in PHP, there is a lot of useful libraries already built in the JVM, covering all the basic needs. Due to its popularity, third party libraries are available for almost any imaginable problem. To use any class outside of a package, the code needs to specifically import all the packages, even the bundled ones.

*http://java.sun.com/docs/white/langenv/Intro.doc2.html

There is support for handling common problems and techniques, like threading, exceptions, user interfaces, security, networking, etc. One of the most important features is its garbage collector, freeing the programmer from memory management tasks.

2.4.3 OSGi

The OSGi framework is a module system and service platform for Java applications. The main goal of this framework is providing a dynamic component model. It offers a series of basic APIs to develop services, like logging, a HTTP server and the Device Access Specification (DAS), that eases the discovery of the device's capabilities.

An application or component developed for OSGi is called a bundle, and it is a self-contained package with the compiled byte code, additional resources and a manifest. According to that manifest, any bundle can be remotely installed, started, stopped, updated or uninstalled without requiring a reboot. Figure 2.18 shows how bundles can interact with the framework and with each other.

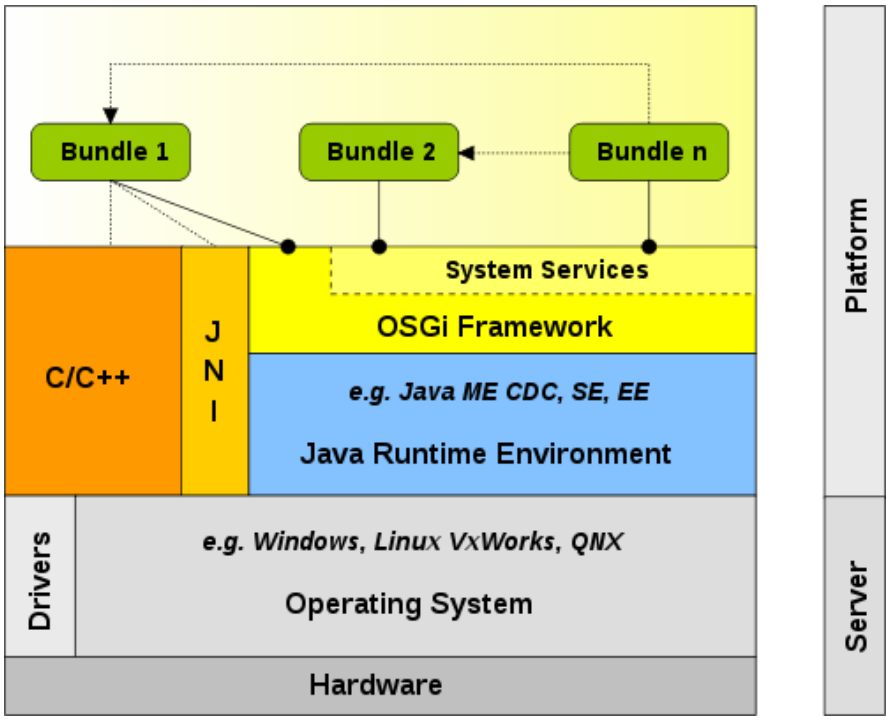


Figure 2.18: OSGi layering

The OSGi specification is maintained by the OSGi Alliance, backed by more than 35 big corporations and where *Deutsche Telekom AG* acts as a full member. There is also a vibrant open source community, having several open source implementations, for example this project uses Knopflerfish OSGi. OSGi R4 is the version used by this bundle, released originally in 2006 and currently the latest version available.

Once the framework is loaded, a console is available to manage all bundles. Table 2.15 lists the most used commands for that console. This console will be running in the background all the time, so it may be useful to attach the process to a `screen` process.

Table 2.15: OSGi commands

Command	**Description**
`ps [-l | -s | -u]`	List installed bundles.
`refresh [<id> ...]`	Refresh packages.
`install <URL> [<URL> ...]`	Install bundle(s).
`uninstall <id> [<id> ...]`	Uninstall bundle(s).
`start <id> [<id> <URL> ...]`	Start bundle(s).
`stop <id> [<id> ...]`	Stop bundle(s).
`update <id> [<URL>]`	Update bundle.
`shutdown`	Shutdown framework.

In this framework modularity and portability are the key concepts. This has proven useful in a wide range of fields like mobile phone, automobiles, grid computing, etc. In this case it is used for building an application server, taking advantage of the built HTTP server.

2.4.4 Java Applets

A Java applet is a small application written in Java designed to be executed inside of a web browser, i.e., in the client machine. An `applet` HTML tag (or the more recommendable `object`) is embedded in the code specifying

where the package is located, the dimensions that applet will occupy in the rendered page and the parameters that will receive. Then the browser downloads it (distributed as a JAR package), and executes it in the scope of that web page.

This enabled the development of interactive web applications with a much better performance than JavaScript and more capabilities, while maintaining full compatibility within the supported platforms. It was often used for graphic and resource-intensive applications, like plotting or basic gaming before Adobe Flash was popular.

Since it provides access to most of the JVM classes, some web applications used them for functions not natively possible in a browser. For example in this project a Java applet is used solely for its native socket capabilities.

Security is critical in a platform like this that allows running code from the web with near-native privileges. To alleviate this, by default an applet has very restricted access to the local filesystem and web pages. The solution is to sign the applet with a trusted entity's certificate, relaxing those limitations. In practice, this is an extra step that slows any iterative development and hardens deployments in corporative environments.

However, the rise of new standards and unsupported devices, the huge popularity of Adobe Flash, the qualitative improvements in JavaScript performance and the disruptively poor user experience crippled its future and nowadays *native* web alternatives are preferred over Java applets.

2.5 Interface: HTML and CSS

The PNAI interface was designed from the ground up following two basic standards for the web, HTML and Cascading Style Sheets (CSS). As a matter of fact, there are two versions completely different created for diametrical purposes: one for desktops to be compatible with every major browsers, other for mobile touch devices based on only one browser engine, Webkit.

2.5.1 HTML

Since the very beginning, HTML has been the soul of the World Wide Web (WWW). Created by *Tim Berners-Lee* at *CERN* around 1990, this markup language was aimed at providing an hypertext digital solution. The beauty of an hypertext document is that it can contain references (hyperlinks) to other documents, so a person or computer can navigate easily though the documents in a non-linear way, normally from a remote computer.

The most popular way to view these documents is using a Web Browser. There are several vendors that offer competing products, but the main ones have been evolved following closely the growth of the web. These browsers interpret the code and generate a visual (or audible) representation suitable for human consumption.

Typically HTML pages are delivered using HTTP, the protocol specifically designed for this purpose. Additional resources are also delivered this way, e.g., images, scripts or stylesheets. Every document has a unique address (URL) that allows the browser to find in which remote machine is located (usually a server), which path has to ask for and which further parameters are needed.

The language itself is based on Standard Generalized Markup Language (SGML), father of the more popular (and realistic) Extended Markup Language (XML). Documents written in these languages are composed by elements consisting of *tags* within the page content in a plain text file.

The name of these tags are enclosed in angle brackets (like `<div>`), and normally they come in pairs: one at the beginning (the opening tag) and one at the end (the closing tag, like the opening tag but with a backslash, e.g. `</div>`).

Each tag can contain different attributes that will affect that particular tag. The actual content goes between those two tags, and that content can be a combination of plain text and other tags (children of that element). Therefore a document can be represented as a tree, with each element acting as a node and having `<html>` as the parent node.

The following line shows an example element. It has two different attributes, one called `id` and other called `class`, and each one has a different value. The meaning and usefulness of these attributes will be discussed

later.

```
<div id="myid" class="myclass">Text</div>
```

A document consists of two sections: the head and the body. The head contains the metadata of the document (title, language, encoding, styles, scripts, etc) and its content is not displayed in the browser. By contrast the body is where the content goes.

Over the years a diverse range of HTML tags have been supported, either promoted by a standards body or unilaterally by browser vendors. Now there are tags for embedding multimedia content (images, video, audio), tables, forms, headings, paragraphs, comments, lists, quotes, code, etc. Of course, also links and even other full pages using frames.

Besides these content related tags, there are also tags to specify the appearance and layout of the page (CSS) and its behavior (JavaScript). Since this code does not relate to the content itself, it is best to put it in external files and just link them from the HTML. Table 2.16 shows a comprehensive list of the most used elements in current web pages. On the other hand, Listing 2.3 on page 53 shows how an HTML document looks like.

Table 2.16: HTML elements

`html`	`head`	`body`	`title`
`meta`	`object`	`script`	`p`
`h1...h6`	`ul`	`ol`	`li`
`blockquote`	`pre`	`div`	`span`
`a`	`em`	`strong`	`code`
`br`	`hr`	`img`	`form`
`input`	`select`	`textarea`	`iframe`
`table`	`tr`	`th`	`td`

Each of these elements may specify a different set of attributes, name-value pairs separated by a '=' symbol written within the start tag of the element after the element's name. The most important attributes are `id` and `class`, and they can apply to every element. Most other attributes are

useful only for one or two elements.

The former specifies a unique identifier for that element, so it could be easily modified by CSS rules and JavaScript code. Additionally, it allows to link directly to that element rather than to the full page putting its identifier at the end of address after the '#' character.

The latter indicates that the element belongs to one or more classes. Classes are used to classify and group similar elements for semantic or presentation purposes, something very useful for CSS but also for JavaScript. Multiple class values can be assigned by separating their names with spaces (`"class1 class2 class3"`).

This standard has been traditionally maintained in the WWW Consortium (W3C), the primary international organization for the WWW. The most popular version is HTML 4 (and its subsequent minor revision HTML 4.01), the only version that it is also an International Organization for Standardization (ISO) standard [4]. A parallel version with the same capabilities but based on well formatted XML was released as Extensible HyperText Markup Language (XHTML) 1.0 and later XHTML 1.1 [4].

After several years of stalled development making the XHTML 2.0 specification, some browser vendors got tired of waiting and founded the Web Hypertext Application Technology Working Group (WHATWG). Through this new standardization body HTML 5 was born, and later the W3C dropped XHTML 2.0 and declared HTML 5 to be the official evolution of HTML. More information about this new standard is stated in § 2.5.3 on page 73.

Every version is mostly compatible with the previous one, but some subtle differences make browsers need a way to distinguish between them. For that a well formed HTML document must start with the `doctype`. This is a tag that needs to be put at the very beginning of the document, defining which standard the document follows. Listing 2.3 on page 53 showed the standard HTML 5 `doctype`.

When the page is parsed by a browser, it starts believing that the document complies with this `doctype`, and the rendering is relatively predictable according to the specification. However, given the large quantity of malformed pages*, in many occasions it sadly backfires to a quirks mode.

*This vicious cycle is also the browsers' fault, because their parsers have been historically

As opposed to the standard mode, this mode is mostly unpredictable, and it usually leads to strange bugs and inconsistent states.

This situation gets worse because traditionally different browsers provide contradictory support (or no support at all) of some parts of the specification, especially CSS. IE is the worst offender in this aspect, not only for the engine itself, but because it took more than five years until Microsoft finally released a new version. During that time, every non security bug remained unfixed in the default browser that shipped with almost every computer.

Due to the huge market share held by IE (see Figure 2.19(b) on the facing page)*, this has been the primary frustration for every front end developer, and a tremendous drawback for web applications. For instance, in this work a considerable amount of development time was spent bypassing IE bugs, and from the beginning it was decided to drop IE6 support, a browser in clear decline not even fully supported by Microsoft.

New IE versions are slowly reverting that situation with a more modern and standards based engine, and for developers rejoice the market share of IE6 is rapidly declining (see Figure 2.19(b) on the next page). However, it is probable that as long as IE6/7/8 and Windows XP retain any substantial market share, web developers will continue supporting effectively 3 or 4 ostensibly different engines only from Microsoft, plus the competition.

2.5.2 CSS

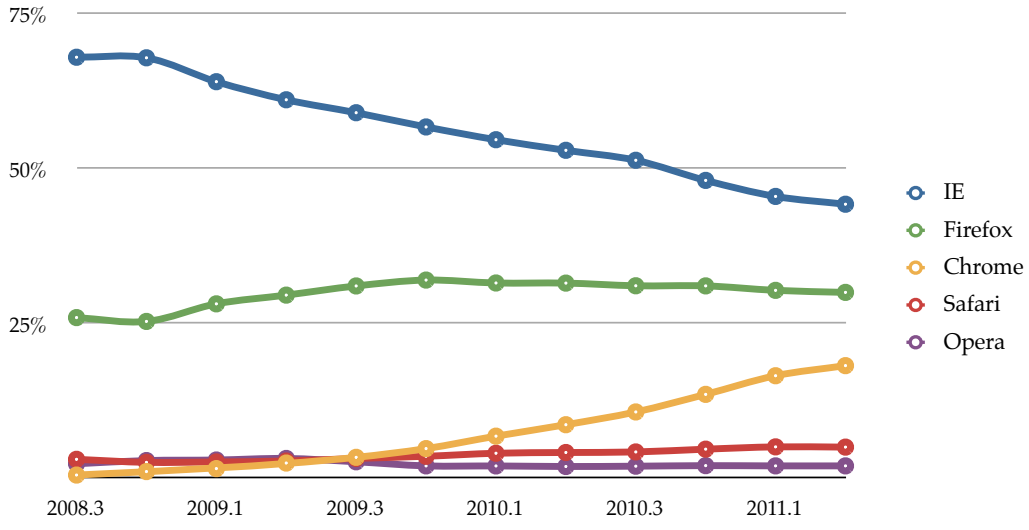
In the early years of the WWW, as it was starting to hit the mainstream, web developers were asked to build visually attractive web pages. HTML was not prepared for this and the only way to do it was using a lot of tables and cropped images. Inevitably, this always ends in a tag soup impossible to understand and even more difficult to maintain.

The W3C, with *Robert Cailliau* (WWW co-founder) at its head, wanted a solution to separate the structure from the presentation. That way the HTML file would gather only the content and external style sheets would specify how to show that content (layout, colors, fonts, etc).

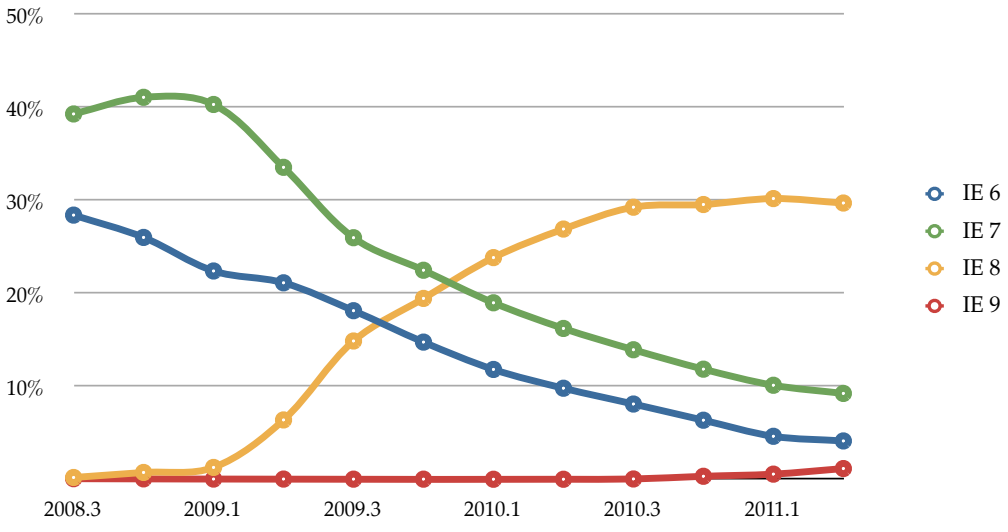
---

very lenient about HTML errors in order to be compatible with more pages.

*Data extracted from *Statcounter GlobalStats*: `http://gs.statcounter.com/`. It may be a little biased towards modern browsers but trends match other sources.



(a) Market share of the top five browsers



(b) Market share of current versions of IE

Figure 2.19: Current browser market shares and trends

This separation is aimed at improving the accessibility of the page independently of how it is consumed (on screen, in print, by voice, etc). Incidentally it provides more flexibility and control, avoiding repetition by sharing the same stylesheet between pages. In most browsers, there are also options to override certain aspects of a page style, e.g. this is very useful for a visually impaired person.

After several proposals in this direction, *Håkon Wium Lie* and *Bert Bos* started working in the specification of CSS, releasing the first version in 1996. The second recommendation CSS 2 was released less than two years later in 1998, and more than thirteen years later its successors (CSS 2.1 [5] and CSS 3) are yet to reach the *Recommendation* status [6]. Almost full support for CSS 2.1 is provided in every major browser and partial support for CSS 3 is provided in modern browsers with uneven results.

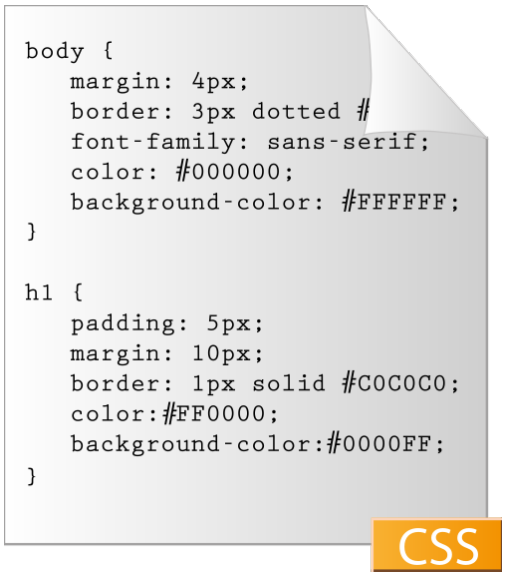


Figure 2.20: Example CSS source

A CSS file defines a set of rules with properties to be applied to the elements of the page. Each rule can affect several elements, and each element can be affected by several rules at the same time, so a priority system is needed to predictably decide with rule shall prevail in this *cascade*.

Every rule consists of one or more *selectors* and one or more properties stated inside the declaration block. A selector is a pattern matching elements in the HTML source. Table 2.17 on the facing page contains all the available selectors in CSS 2.1*.

Those selectors can be mixed and put together composing a more complex selector targeting specific elements. When several selectors share the same declarations, they may be grouped into a comma-separated list to avoid repetition. When there is a collision between two rules, the browser

*The information depicted in Table 2.17 on the next page has been extracted from the official specification documentation, specifically from here: http://www.w3.org/TR/CSS2/selector.html

follows this simplified selector precedence: the selector with more ID attributes, the selector with more classes/attributes/pseudo-classes, the selector with more elements/pseudo-elements. If two rules tie, the last declared rule is the winner.

Table 2.17: CSS 2.1 selectors

Pattern	**Type**	**Meaning**
`*`	Universal selector	Matches any element.
`E`	Type selector	Matches any E element (i.e., an element of type E).
`E F`	Descendant selector	Matches any F element that is a descendant of an E element.
`E > F`	Child selector	Matches any F element that is a child of an element E.
`E + F`	Adjacent selector	Matches any F element immediately preceded by a sibling element E.
`E:first-child`	First child pseudo-class	Matches element E when E is the first child of its parent.
`E:link` `E:visited`	Link pseudo-classes	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited).

`E:active`
`E:hover`
`E:focus`
	Dynamic pseudo-classes	Matches E during certain user actions.

`E:lang(c)`	Lang pseudo-class	Matches element of type E if it is in (human) language c.
`E[foo]`	Attribute selector	Matches any E element with the "foo" attribute set, whatever the value.

continued on next page

Table 2.17: CSS 2.1 selectors (continued)

Pattern	Type	Meaning
`E[foo="bar"]`	Attribute selector	Matches any E element whose "foo" attribute value is exactly equal to "bar".
`E[foo~="bar"]`	Attribute selector	Matches any E element whose "foo" attribute value is a list of space-separated values, one of which is exactly equal to "bar".
`E[lang|="en"]`	Attribute selector	Matches any E element whose "lang" attribute has a hyphen-separated list of values beginning (from the left) with "en".
`E.foo`	Class selector	Matches any E element with a class name equal to "foo".
`E#foo`	ID selector	Matches any E element with ID equal to "foo".

After those selectors, the declaration block is written between curly brackets. This block contains a list of declarations, and each declaration is composed by a property, a colon acting as separator (':'), a value and a semi-colon (';'). Those declarations change the properties for the matched elements. The number of properties is just too large to be specified in this document, however it is important to explain some of them. Listing 2.4 on the facing page sums up all the explained syntax.

Positioning

Positioning is arguably one of the most important and powerful capabilities in the CSS specification. Unfortunately, it is also the least comprehended and the most error prone. The main properties for specifying how to place an element are `position`, `display`, and `float`. In CSS 2.1 there are three

Listing 2.4: CSS example code

```
selector [, selector2, ...][:pseudo-class] {
  property: value;
  [property2: value2;
  ...]
}
```

positioning schemes defined:

Normal flow By default, every element has its `position` property set to `static`. This means that the element is integrated into the normal flow of the page. However, with this value the element will not be affected by the `top`, `bottom`, `right` or `left` properties. If we want to use these properties to align an element referencing other elements we should set the `position` property to `relative`.

Depending on the element type, its `display` property can be set by the browser *user agent stylesheet* to:

`inline` Inline elements are laid out in the same way as the letters in words in text, one after the other across the available space until there is no more room, then starting a new line below. E.g., `span`, `em` or `strong` elements.

`block` Block elements, on the other hand, are stacked vertically, like paragraphs and `div` elements. So independently of their width, they always cause a *line break*.

`none` Elements that will not be taken into account when rendering the page, so they will not be shown. There are not a lot of elements with this value by default (`script`, `style` and similar), but it is very used in conjunction with `block` to hide and reveal elements dynamically, i.e., using JavaScript.

Floats A floated element is taken out of the normal flow and shifted to a side. The property `float` can be set as `left` or `right`, to push the element to those sides, and any value triggers the element to have the

`display` property set to `block`. Following elements in the normal flow are wrapped alongside the floated element, as shown in Figure 2.21.

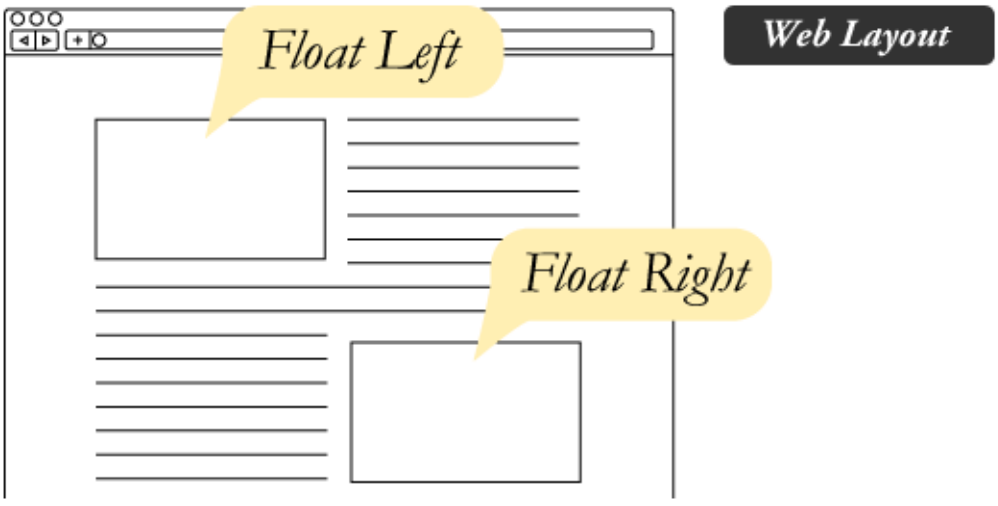


Figure 2.21: CSS floats

The `clear` property allows to specify not to wrap the element around another floated element; instead it is placed below those elements. It supports `left` and `right` values to *clear* all elements floated at that side, or `both` to *clear* any floated elements.

Absolute positioning An absolutely positioned element is taken out of the normal flow and effectively ignored by other non-descendant elements, so it has no effect on other items positions. Its position is calculated using the `top`, `bottom`, `left` and `right` properties with respect to the first ascendant (closer) that has a `position` value set to other than `static`. Since the `relative` value is almost exactly as the `static` value, it is often used to mark the ascendant as the reference.

There is another type of absolute positioning: the `fixed` value works as `absolute`, but it retains its position even when the document is scrolled. This is used to achieve a sticky element effect in a page.

Those properties are enough to place an element in the two-dimensional surface that is the document. However, CSS 2.1 introduces another dimension to decide which box must be drawn last when boxes overlap visually. The order is determined by the position of the boxes in this *z-axis*.

The `z-index` property can be set to any integer (including negatives) to manually set the stack order of that element. When the page is rendered, elements with greater stack order are drawn in front of other elements with a lower stack order. However, this `z-index` property is only taken into account when the element is positioned*.

Usually this property is not needed, but in complex cases it can be handy. By default, elements are rendered in order of appearance in the HTML source file; i.e., if any overlapping occurs, the last element should be drawn on top. So the `z-index` property is mainly used to revert that behavior.

One side effect of this property is that when an element is positioned, a new *stacking context* is created, completely independent from the its siblings. This means that the stack order of all child elements now refers to the parent element instead of to the whole document.

Moreover, a child element can only be positioned between its own siblings; respect the parent siblings a child element always have the parent stack order. For practical purposes, this limits the `z-index` usefulness: if an element is positioned on top of another element, children of the latter can never be positioned on top of the former element.

The Box Model

In CSS each element is treated as a rectangular box. That box consists of several components, so the final looks and dimensions are defined by the resulting composition of those components. Figure 2.22 and Figure 2.23 on the next page† break down a box into its different components.

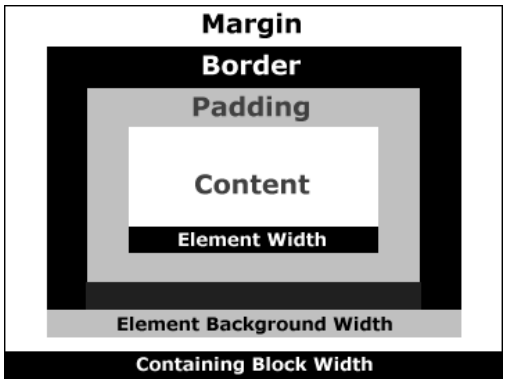


Figure 2.22: CSS box model

Each component completely encloses the preceded component like a matryoshka doll. Therefore, its dimensions must be equal or greater than

*A positioned element is any element which its `position` property is set to other than the default `static` value.

†Original image from: http://www.hicksdesign.co.uk/boxmodel/.

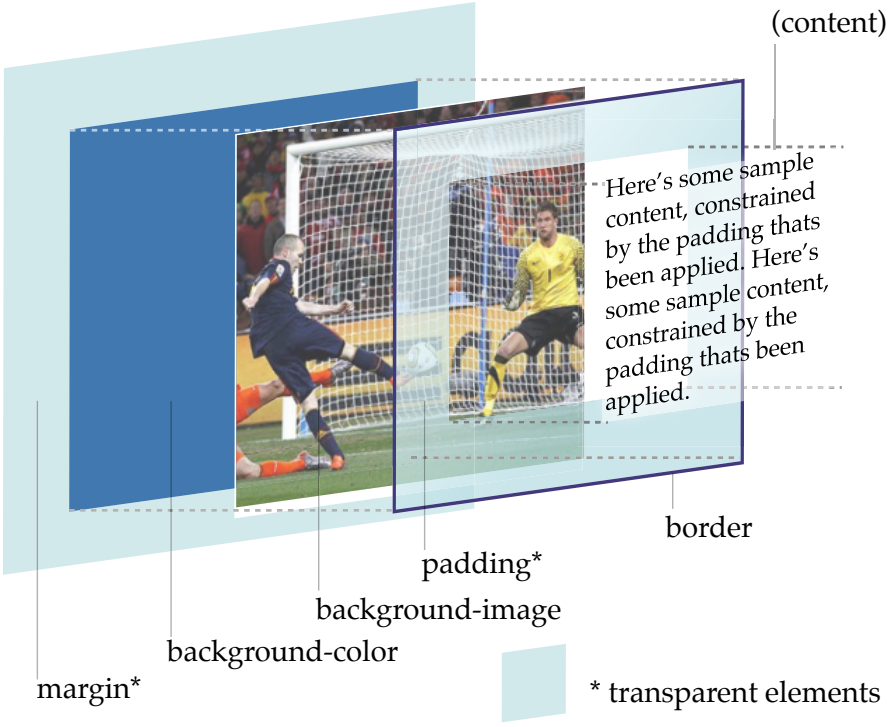


Figure 2.23: CSS box model in 3D

the enclosed component. These four components can be explained as:

Content is the actual content for that element, including the text and other child elements. The `height` and `width` CSS properties set these dimensions.

Padding is the transparent space surrounding the content, making room between it and the border. Different values can be specified for the top, bottom, left and right spaces.

Border is the delimiter that marks off the visual boundaries of the box, and it can have several styles and colors. As with `padding`, different values can be specified for the top, bottom, left and right borders.

Margin is the transparent space surrounding the rest of the elements, and it is usually reserved to provide a visual separation between other elements. As with `padding` and `border`, different values can be specified for the top, bottom, left and right spaces.

Any of those elements can have zero dimensions, in that they do not increment the box size. The `margin` property can even have a negative value, in that case the box is pushed to that direction. An element's background is painted between the border boundaries, including *behind* the border itself, and excluding the margin space.

Traditionally, working with CSS dimensions has been a painful experience because in IE the `width` property did not set the content width but the background width. Newer versions of IE (IE6+) still exhibit this behavior bug for compatibility reasons, but only if the page triggers the Quirks Mode. The best solution is to ensure that the browser use the Standards Mode by writing correct HTML source code, instead of adding a bunch of empty elements trying to circumvent this issue.

Sometimes the content of an element is bigger than the allocated size for that element. The `overflow` property defines what to do in those occasions. By default it has a value of `visible`, allowing the content to be drawn outside the box, but it also accepts `hidden` (clipping it and hiding the overflow content), `scroll` (giving a scroll bar to show the overflow content) and `auto` (like `scroll` but only showing the bar when there is overflow content).

Finally, it is important to note that, although those components work as intended in elements with its `display` property set to `block`, it is quite erratic with inline elements. The `height` and `width` properties are completely ignored: the height is calculated by the `line-height` property and the width by the content itself.

The `padding` and `margin` properties work, but only affecting the surrounding elements in the horizontal direction, not the vertical direction. Though, `border` works as intended. In any case it is clear that inline elements should only be used for text.

Values and Units

Most numeric values are composed by a float number and the identifier for the chosen unit. Several units are available to specify lengths, some of them are absolute and others are relative. Absolute units include centimeters (cm), millimeters (mm), inches (in), points (1 pt = 1/72 in) and picas (1 pc = 12 pt).

However relative units are preferred, because they will more easily scale from one medium (device) to another. The most widely used are px (pixels) and em. Pixels are relative because the actual size depends on the size of the pixel in the screen device, and it is the most popular way of specifying component sizes.

In turn, em it is defined as the font size of the element, but when it is used in the `font-size` property it refers to the font size of the parent element. It is therefore preferred for setting font sizes and other typographical properties, like the line height.

The last way of specifying sizes is using percentages (a number plus the '%' symbol). However, those percentages are defined upon different properties depending on which property are applied.

Most of the time it refers to the value of the same property on the parent (equals 100%), e.g., when setting `height`, `width` or `font-size`. Some properties use another property value as a reference, e.g., spacing properties like `top` or `margin` refers to the `width` and `height` depending on their affecting dimension. In some typographical properties it refers instead to the current font size, e.g., when setting `line-height`.

Hence, percentages can be used in fluid layouts for almost everything. Several reasons make it the recommended unit. Accessibility is the main issue, because the user browser can define different sizes. But it has other advantages too, such as adapting dynamically to disparate screens or resizing windows.

Colors are usually specified following an hexadecimal Red Green Blue (RGB) notation: first the '#' character, then three hexadecimal numbers between `00` and `FF` defining the *amount* of each primary color. For example, `#FFC0CB` represents the pink color, and it is formed by 255 (`FF`) red, 192 (`C0`) green and 203 (`CB`) blue.

Some popular colors can also be specified using a keyword, such as `white` or `black`. Other useful keyword is `transparent`, that declares that the element property has to be completely transparent.

In CSS 3 another notation called Red Green Blue Alpha (RGBA) is available, with an additional value that declared the *opacity* of that color. In that notation, though, amounts have to be in decimal notation, e.g. a pink color at 50% opacity is defined as `rgba(255,192,203,0.5)`.

2.5.3 HTML5 and CSS3

HTML 5 is the next generation of HTML, and it includes a lot of new functionality[7]. Unlike its predecessors, it is not a great big monolithic specification. Indeed, the term HTML 5 encompasses the latest round of web standards and specifications, including HTML 5 itself, CSS 3 and other extensions that define additional APIs for HTML documents.

Since it is mostly backwards compatible, it can be reasonably used today. Those browsers which supports that part of the specification should work fully or partially, while the rest should fallback to an acceptable rendering. The important thing is that, even if the specification is not finished, a web application can choose to use a slice of it to enhance the user experience.

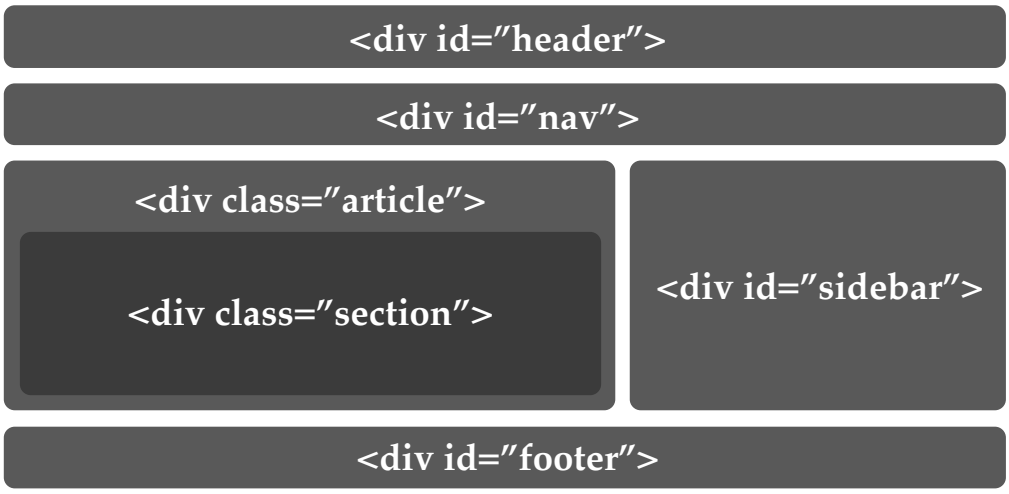
One focus of HTML 5 is to improve the structure of a document by adding new semantic elements, primarily to avoid the `div` soup. Figure 2.24 on the following page introduces the new `article`, `aside`, `header`, `footer` and `nav` elements and compares it with the traditional HTML 4 approach. At the same time, a lot of non-semantical elements have been deprecated.

But probably the most popular new feature in HTML 5 is the new built-in multimedia integration. By using the new `video` and `audio` elements, a browser can play natively video and audio files. And with the `canvas` element a page can draw anything in a page. These additions make proprietary plugins dispensable in most scenarios, such as Adobe Flash, Microsoft Silverlight or Java applets.

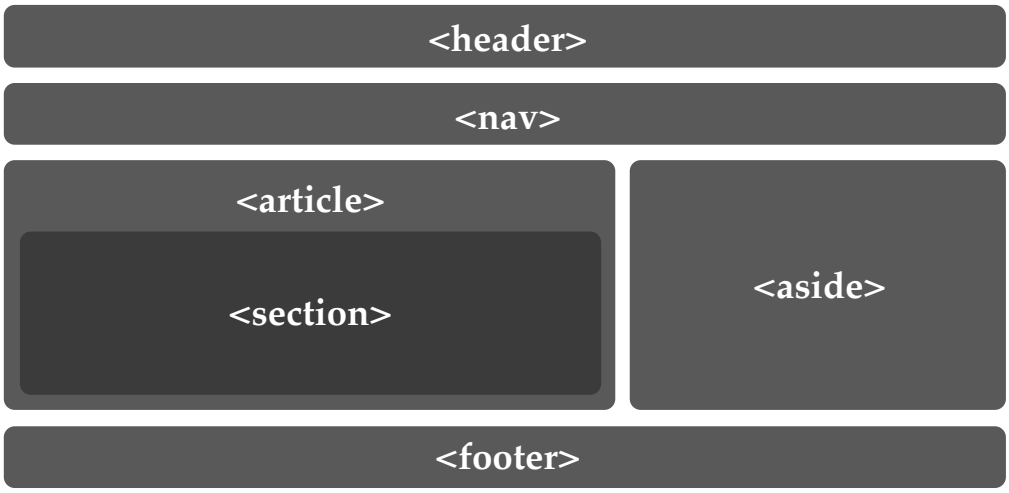
A lot of new attributes have been added and, in some occasions, deprecated. Forms have got a well-received update with more specific input elements (to write an email or an URL, to choose a date or a color, etc.) with auto-validation capabilities. All elements can be *draggable* or *editable* with the flip of an attribute.

Additional modules to the main specification give web applications the ability to store information in the browser in a local database (see § 2.6.3 on page 83) and caching resources, allowing the development of offline web applications. Other modules give new APIs to manipulate and process local files right in the browser, with no need of sending it to the server.

Files from the computer can be dropped in the browser (in mass or individually) so the web application can read them directly, and files with a certain Multipurpose Internet Mail Extension (MIME) type can



(a) HTML 4 Structure



(b) HTML 5 Structure

Figure 2.24: Structure of typical HTML 4 and HTML 5 documents

also be linked to a specific web application. For example, images can be configured to be open by default in a web image editor, instead of with a local application.

Another possibility is to directly manipulate the history of the browser, very useful for Asynchronous JavaScript and XML (AJAX)-based applications. Other new features are geo-localization (to know where the user is physically located), notifications (to notify events in the browser) and Web Sockets (to open a socket right from the browser).

Given the huge web applications that can be developed on top of these new features, performance could become critical very quickly. Web Workers are available to run heavy tasks in the background in parallel with the main loop. If JavaScript code is too inefficient for the task, there is an HTML 5 extension to run compile native code (written in C/C++). Even with the use of Web-based Graphics Library (WEBGL), the development of interactive native 3D applications is possible.

Many, if not all of these new capabilities, are closely linked with the use of JavaScript. There is an Application Programming Interface (API) for every feature, to modify the state and integrate it with the web application. Additional discussion over the DOM takes place in § 2.6.3 on page 82.

On the other hand, CSS 3 is also considered to be a big step in web development. In order to speed up the tedious standardization process, the CSS 3 specification effort is much more modular than the HTML 5 specification. Currently there are more than 50 micro-specifications in work, with uneven progress. Some of the most important new features are:

Media queries Now media queries are more important than before, allowing device targeting. For example, some rules or stylesheets can be specifically designed to only affect a mobile device when it is oriented in portrait, or a generic device when it has more than 960 pixels of width.

Selectors New selectors are introduced, mostly to target dynamic pseudo elements. Two new pseudo selectors, `::before` and `::after`, are more special, because they generated new visual elements in the page, very useful to recreate artifacts.

Borders In CSS 3 borders are incredibly powerful: it can have rounded

corners with different radius for each corner, and images can be used to fill the border no matter the size of the box.

Shadows There are two kind of shadows: box shadows and text shadows. Box shadows are applied to the elements boxes, while text shadows are applied to the text itself. Both shadows can be either inner or outer, and they have to specify a color, opacity, direction, size and blur. A shadow value can be composed by a undetermined number of shadows.

Gradients For backgrounds, borders or any property that accepts an image, a gradient color can be generated on the fly by the browser. A gradient can be composed by any number of color stops, it supports opacity changes and it can be either linear or radial.

Multiple Backgrounds Going further, backgrounds can be dynamically composed with several images, gradients and colors. This is very useful when the sources have opacity, since it allows complex compositions, e.g., mixing textures and several shadows.

Transforms An element can be easily rotated, skewed, translated or scaled. Additional 3D transformations are defined in another CSS module, with the same transformations in 3D and a perspective function.

Transitions CSS 3 allows to gracefully transition an element from a state to another, for example when the class is changed or when the mouse is hovering the element. In those cases two different declaration blocks are defined for each class (or pseudo-class), and instead of a sharp change of values the element will slowly morph into the new values. To set a transition it is needed to specify the altered properties (or all), the timing function (the speed curve of the transition effect) and the total time to spent in the effect.

Animations Similar to transitions, an element can be animated using only CSS rules. Two or more keyframes need to be defined, with the set of properties and values that the element will take each keyframe. Then, according to its parameters (duration, timing function and

iteration count), the styles are smoothly interpolated from keyframe to keyframe.

Web Fonts Although the `@font-face` rule was introduced in IE4 (1997), until recently the use of custom fonts in web pages was not widespread. One of the reasons is that there is no single format compatible with all major browsers. Even now, to target the maximum number of browsers five different files have to be deployed, each with a different font format. In the near future the Web Open Font Format (WOFF) is supposed to be the standard, simplifying this process.

When a browser implements some of those properties, since most of these modules are a work in progress or directly they were proposed by that browser, their names are usually preceded by a vendor prefix. For example, `border-radius` was called `-webkit-border-radius` in Webkit browsers, `-moz-border-radius` in Firefox and `-o-border-radius` in Opera. Later, when the standard was more settled, they all changed it to `border-radius`. For this reason any modern web application that wants to use such a modern property now have to specify redundant rules targeting each browser engine.

2.6 Client Programming Language: JavaScript

The only programming language available natively in browsers is JavaScript, so any modern interactive web application should embrace it. Given that the web has become such a popular platform and JavaScript has a steep learning curve, the language is getting more attention lately and all indications point to a very bright future. Most of the coding in this project was done in JavaScript.

2.6.1 History

Brendan Eich originally developed the JavaScript language working at *Netscape Corp.* under the name of *Mocha*, renamed to *LiveScript* and then

again to finally JavaScript.

First implemented in Netscape Navigator 2.0 in 1995, and contrary to whatever the name may lead to think, it has little to none to do with Java. Indeed, the name only served more marketing purposes. The competitor, Microsoft, added support for the essentially the same language in IE 3 the next year, but called *JScript* to avoid trademark issues.

Then Netscape delivered the language to the European Computer Manufactures Association (ECMA) for standardization. The effort culminated in 1997 with the first edition of a new open standard called *ECMAScript*, named as a compromise in the dispute between Netscape and Microsoft. For some reason, that name was never popularized, instead the original term JavaScript is used in almost every situation.

Figure 2.25: Since there is no official JavaScript logo, here is a rhino

Being closely linked with a markup language that even non-programmers and *amateurs* could code, JavaScript was initially disregarded as a toy by many traditional developers. Other apparent limitations of the language, a very lenient parser, performance issues and compatibility problems between browsers did not help its popularity either.

However, the advent of AJAX, its ubiquity in browsers, better/faster parsers and the need for more complex web interfaces put JavaScript back in every professional web developer toolbox. Since then a multitude of framework and libraries have been released to ease the development, with its usage growing broadly not only in browsers but even in server applications.

2.6.2 Quick Overview of the Language

At first glance the syntax may seem a simplified version of Java (or other C-based language), but much more direct (no classes, packages, etc.). Make

no mistake, under the hood JavaScript hides a lot of powerful constructions that makes it flexible enough to suit diametrical programming paradigms. Some of the most important features are:

Scripted The browser receives the plain source code, and then it is interpreted and run. The big advantage is that it does not needed to be compiled beforehand, the traditional disadvantage is that it is slower than native code or even Java byte code.

Nevertheless, a lot of improvements have been deployed lately in browser engines, and new techniques like Just-in-time compilation (JIT) nearly close that gap. On the other hand, similarly to other script-based languages, it is easy to end up with a low-quality codebase, since inexperienced programmers can quickly write working code.

Imperative statements It supports the basic imperative paradigm with the typical if statements, while and for loops, global functions, etc. It is not required to think about objects to make a script, with those things it can be possible to write a full-fledged application if wanted.

Object based Internally, in JavaScript everything is an object, not only variables but even functions or undefined values. However, there are no classes like in Java, to create a new object it can be specified in a literal notation or using a constructor (a plain function).

Prototype based The language is classless, but inheritance between objects is possible through the use of *prototypes*. A prototype is just an object used as a template from which to get the initial properties for a new object. If an object is asked for a property (also known as *attributes* and *methods* in Java jargon) it does not have, its parent is asked, and then again until the property is found or until the root object is found (the end of the *chain*). Listings 2.5 on the next page and 2.6 on the following page show how inheritance works in JavaScript compared to Java.

It is easy to demonstrate that this abstraction can be more powerful than classes because a similar class-based OOP behavior can be achieved with prototypes but the opposite is not true.

Listing 2.5: Inheritance in JavaScript

```
function Employee() {
  this.name = "";
  this.dept = "general";
  this.hello = function() {
    console.log("Hello, I'm " + this.name);
  };
}
function Engineer() {
  this.dept = "engineering";
  this.projects = [];
}
Engineer.prototype = new Employee;

var john = new Engineer();
john.name = "John Doe";
john.hello();
```

Listing 2.6: Inheritance in Java

```
public class Employee {
  public String name;
  public String dept;
  public Employee() {
    this.name = "";
    this.dept = "general";
  }
  public void hello() {
    System.out.println("Hello, I'm " + this.name);
  }
}
public class Engineer extends Employee {
  public String[] projects;
  public Engineer() {
    this.dept = "engineering";
    this.projects = new String[0];
  }
}
Employee john = new Engineer();
john.name = "John Doe";
john.hello();
```

First-class functions As said before, JavaScript functions are objects, so they have properties and methods and can be assigned to variables as every other object. Moreover, these functions can be passed as arguments, returned as other function values and invoked in any moment and scope using multiples ways (such as the `()` operator).

Functions do not need to be defined in the global scope, they can be defined inside other functions. Nowadays the use of closures (specifically anonymous functions) is considered a very good practice to avoid cluttering the global scope with variables.

As stated repeatedly, every variable in JavaScript contains a reference to an object. However, that does not mean that values do not have a type, and there are several primitive types available in the language. In any case, even literals of these types have accessible properties like any other object.

Undefined and Null Before assigning some value, a variable holds the value `undefined`. This `undefined` is the only object of, pun intended, type `undefined`, a full-fledged object that can be even used in assignments. On the other hand, `null` is never used by JavaScript, so if some variable holds this value, it must have been set programmatically by the developer. Although the type `null` is apparently Object, internally it is also the only value of type `null`.

Number In JavaScript there is no external distinction between integers, floating point values and others, every literal number is treated equally and has the same properties. Internally they are all doubles, floating points numbers with an accuracy nearly 16 significant digits, so some real numbers cannot be represented.

String Strings are delimited by double quotes and single quotes indistinctly, the only difference is which quotes need to be escaped inside of the string.

Boolean As usual, there are only two values possible in this data type: `true` or `false`.

For all of these types (except undefined/null), there are object constructors, but the preferred way to initialize them is through plain literals.

Besides these primitives data types, there are also several native objects built in the language:

Array As other C-based languages, arrays are available to store a collection of values in a variable using integers as keys. To create an empty literal array (also preferred to the Array constructors), two square brackets can be used. This bracket notation can also used for accessing object properties, so it can be confused with associative arrays while the language does not support this directly.

Date A simple object to store and manipulate dates down to a milliseconds.

Error Similarly to exceptions in other languages, this kind of objects can be thrown in case of error (using the throw clause) and caught for error handling (using the try/catch block).

Math This object cannot be *instantiated* and it only contains math-related constants and functions.

Regular Expression Literals between backslashes represents regular expressions, and they hold some useful methods to deal with search and replace in strings. The `RegExp` constructor can be used but, again, it is not recommended.

Function Functions are declared by using the keyword `function`, an optional name, the arguments and the body block. The object constructor, though also available, is very ill-advised, as it relies on strings.

Object The rest of the values are of type Object or indirectly inherit from this type. Objects literals are represented by curly brackets and a comma-separated list of keys/values separated by colons. This literal syntax is the base for JavaScript Object Notation (JSON), a notation used for data exchange that can be natively used in JavaScript.

2.6.3 The DOM

Apart from the native capabilities of the JavaScript language, browsers provide a very useful tool to deal with the HTML elements of the page, the DOM. It consists of an API accessible from JavaScript and it allows the

dynamic retrieval, creation, modification and deletion of HTML elements within the page scope.

This DOM is a fully object-oriented representation of the web page, in which every element is depicted as an object with properties that responds accordingly to different methods. Those properties determine the document structure, style and content. Actually, this representation is not bound to JavaScript or even browsers, it can be accessed with other programming languages and applications. But since JavaScript is the most popular one in browsers they are usually delineated as partners.

The basic idea behind the DOM is that the page is composed around a tree of elements (nodes). Starting from the root node, every element may have an indeterminate number of children. The DOM provides some methods to traverse that tree either by depth (between a parent and its children) or breadth (between siblings). Visually, children are drawn inside their parent bounds, but that behavior can be subverted with the use of CSS properties.

This API forms a standalone standard maintained by the W3C. But, as other web standards, its support varies from browser to browser, each one suffering from either partial implementations, vendor extensions or, commonly, both.

Another chronic burden is that modifying visible elements in the page is a very expensive operation. Since the DOM is linked to the visual representation of the page, every modification triggers a live partial or full page reflow. For this reason it is advisable that, when several DOM changes are needed, they are made on a detached element and, at the end, that element is attached to the DOM tree.

DOM Objects

There are several objects available to JavaScript exposed by the DOM. Those objects properties and functions conforms the API. The most important ones are:

window This object represents the browser window itself, and it allows the retrieval and modification of several properties related to the browser in general and the document window in particular. Since this

is the global object, all of its properties are initially accessible with no need to specify `window`, as if they were *local variables*, so for example `location` is the same as `window.location`. It has lots of attributes, some of the most important being:

window.location Through this property the location (URL) for the window can be accessed and modified.

window.innerHeight / window.innerWidth These readonly properties contains the size of the content area of the browser window. That is, only the area occupied by the document, not counting external browser elements like the title bar, the status bar or the menu bar.

window.screen This object describes the screen: its size, its color depth and the position of the window in the screen.

window.navigator This object contains information about the browser, such as the user agent, browser version, platform, language, plugins installed, etc. Modern browsers also include access to geolocation data in this object.

window.history This object provides an interface for manipulating the browser session history, i.e., the pages visited by the user in this tab. Initially it was not possible to change or read the history, only traveling through it, but an HTML5 extension encloses several methods to add and modify history entries (but not read).

window.localStorage This object contains the Web Storage API, one of the key components of HTML5. Basically, it simplifies the storage of significant amounts of data locally in the browser, so that it is available in successive visits using a key/value pairs interface.

window.applicationCache Another key component of HTML5, through this object the cache of static files is managed, allowing the rising of offline web applications.

window.setTimeout() / window.clearTimeout() This useful function executes a function after the specified delay. It returns a timer identifier that allows canceling it afterwards.

window.setInterval() / window.clearInterval() Similar to the previous function, this sets a timer that will repeatedly execute a function with a fixed time delay between calls.

window.open() It opens an additional window with the size and properties specified.

window.close() It closes the current window but it only works if the window was opened programmatically by JavaScript.

window.scroll() This function scrolls the window to a particular coordinate in the document.

window.escape() / window.unescape() A very useful utility function to encode/decode special characters in a string so it is suitable for cookie storage or HTTP requests.

document The interface of this object comprises all the necessary tools to retrieve and modify the document, with particularly valuable functions for DOM retrieval. Therefore, it is almost impossible to make a web application without accessing this object, either directly or employing a JavaScript library. The most relevant properties and methods are:

document.cookie This object contains the interface to get and set the cookies of a page, that is, a small amount of data stored in the client that will be usually to maintain a session with the server. Since these cookies will be transferred with every request to the server, its overuse could slow the page loading.

document.styleSheets This readonly object contains references to all the stylesheets in the page. There are also properties for accessing other kind of elements, like `document.links`, `document.images`, `document.forms`, etc, but in modern JavaScript paradigms these are not very used anymore.

document.title This object gets and sets the title of the document, which most browsers used for their window titles.

document.referrer This string contains the address of the page that linked to this page.

document.getElementById() This function returns the element with the specified id. It is the basic pillar of element manipulation, since it is the most used method to directly fetch an element. Because of that, multiple libraries have wrapped this function under the `$()` short name.

document.getElementsByClassName() Similar to the previous function, this one returns the elements with the given class name. This is natively implemented only in relatively modern browsers, but it is so obviously useful that it was already widely implemented in most third-party libraries and toolkits.

document.getElementsByTagName() The third-wheel of the last two functions, this one returns the elements with the given tag name. It seems useful for dealing with all the elements of a certain type, but since DOM manipulations are normally needed inside a certain document *block* rather than across element types in all the document, its counterpart that affects only an element children is more interesting.

document.createElement() This function returns a newly created element with the given tag name. This does not add the element to the document tree, it is needed to insert it manually later on.

document.createAttributeName() This function returns a newly created attributed node with the given name. This function does not add the attribute to any element, it is needed to set it manually later on. This is not very used, since the `element.setAttributeName()` is more convenient, but it could be more efficient if the same attributes need to be changed in a lot of different elements.

document.createDocumentFragment() This function creates a document fragment, an temporary holding object designed to store nodes before adding them to the document. It is not very known, but it is one of the best performance-wise solutions when several elements need to be added to the document.

document.evaluate() / document.createExpression() These functions provide an interface to work with XPath expressions. These advanced expressions enable complex and precise selections of

element nodes in the document tree. Regrettably, only some browsers support this powerful mechanism.

document.querySelector() / document.querySelectorAll() These functions allows the retrieval of elements using the well known CSS selectors. The difference between the two functions if that the former returns only the first matched element and the latter returns all the matched elements. It is only supported by very modern browsers, but it appears to have a bright future because of its ease of use and familiarity.

element Once an element is retrieved or created using the DOM, the returned object conforms to the `element` interface. This interface details all the actual properties to modify not only the DOM structure, but the style and content of that certain element.

element.parentNode Returns the parent node of this element.

element.childNodes Lists all the child nodes of this element.

element.nextSibling Retrieves the node immediately following this one in the document tree.

element.previousSibling Retrieves the node immediately preceding this one in the document tree.

element.attributes Arranges all the attributes of this element.

element.id Sets and gets the id attribute of the element.

element.className Sets and gets the class attribute of the element.

element.clientHeight / element.clientWidth Returns the element size, counting the padding but not the border.

element.offsetHeight / element.offsetWidth Returns the element size, counting both the padding and the border.

element.offsetLeft / element.offsetTop Returns the distances from the left/top border to the `offsetParent`'s node.

element.offsetParent This is the parent node from which all offset calculations are currently computed. Depending on CSS properties, it could be the immediate parent node or other ancestor.

element.innerHTML Sets and gets the content of the element as a string with HTML markup, quite useful and convenient. It is one of the fastest ways of modifying the DOM, since this task is basically the same one browser engines carry out when rendering an HTML page; therefore it is very optimized.

element.contentEditable Another nice addition in HTML5, it toggles the "editable" property of an element. When this is active, the user can edit the element like in a WYSIWYG editor (without any format buttons, just the content).

element.style This object maps (almost) all the CSS properties relevant to this element, so they can be directly consulted or altered.

element.appendChild() / element.removeChild() Insert/delete the given node as the last child of this element.

element.insertBefore() Insert the first given node as a child of this element, just before the second given node (that is also a child node).

element.getAttribute() / element.setAttribute() / removeAttribute() Gets/sets/deletes an attribute of this element.

element.addEventListener() This function attaches an event listener, that is, a function that will be called when that kind of event occurs in this element.

event Since JavaScript is designed to control the document behavior, the specification provides a powerful interface to manage events. These events are not usually created by the programmer, the common flow is that the browser generates them when a certain action happens and then the programmer should handle them. There are events linked to user actions like mouse clicks or key strokes, but there are also events dealing with automatic APIs like HTML5 storage.

event.clientX / event.clientY The coordinates associated with the event, useful for knowing the exact position of the mouse in mouse-triggered events.

event.charCode The code of the key pressed, if the event is a key press event.

event.preventDefault() Most events have a default action that the browser will trigger; this method cancels that action. For example, if the user clicked a link the browser will follow the anchor, but if we catch the event and call this method, the browser will not do anything.

event.stopPropagation() By default, events are propagated from the specific element to the top element in the DOM tree, notifying every parent handlers in that way. This method stops that propagation so that those handlers are not notified. It is a common mistake to assume that these two last methods are equivalent, but their effects are mutually exclusive.

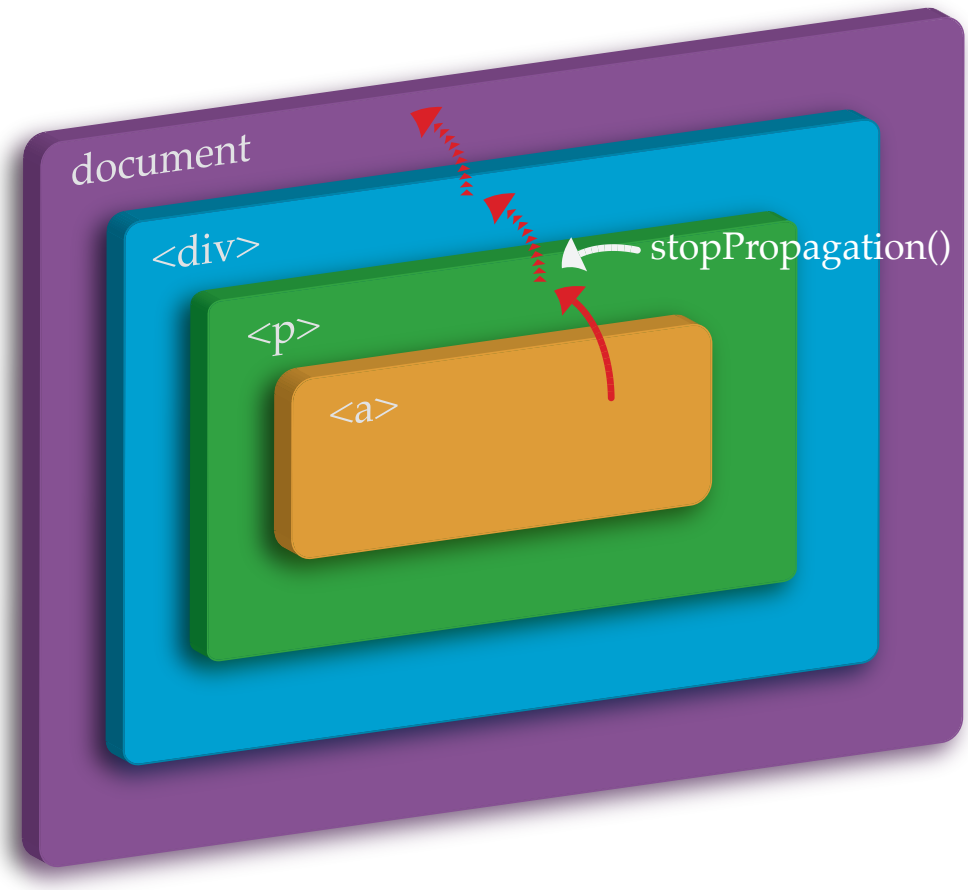


Figure 2.26: DOM event propagation and the effect of stopPropagation()

2.6.4 AJAX

Since its beginnings, the Web has been document based. This means that, when the visitor wants more information and clicks on a link, the browser would request and load another whole document. Even in web applications, the browser would have to reload the full page to send any information from the user and update the page.

This turns out to be inefficient and overkill in most situations, since web applications do not need to reload the full page in every user interaction but only update a little amount of information. To address this problem several solutions where developed, from frames to applets, but they were too cumbersome or foreign respect the basic web technologies.

Figure 2.27: This is not the AJAX logo you are looking for

In 1999, Microsoft proposed and implemented in its IE a new ActiveX control to asynchronously load new data from the server without need to reload the page. Using JavaScript, that data could be requested on demand once the page was loaded, and through callbacks the data could be processed and the user interface changed accordingly.

Later, the rest of the browsers vendors implemented a similar technology under the XMLHttpRequest object. As this was gradually introduced, web developers started using it but it did not become a popular approach until 2004, when several big web applications such as Gmail were developed.

Then, the term AJAX was coined [8] to designate the technologies involved in the process, though most of them can be replaced by others while maintaining the same spirit. Seeing how useful it had become, the W3C decided to standardize the XMLHttpRequest object.

As depicted in Figure 2.28 on the next page, this approach needs another layer of complexity in the client to handle the data and update the interface. This AJAX engine means that much more JavaScript code in the client is

needed. Eventually, this enables the rise of heavy web applications running in the client, with a substantial codebase to maintain and support across several browsers.

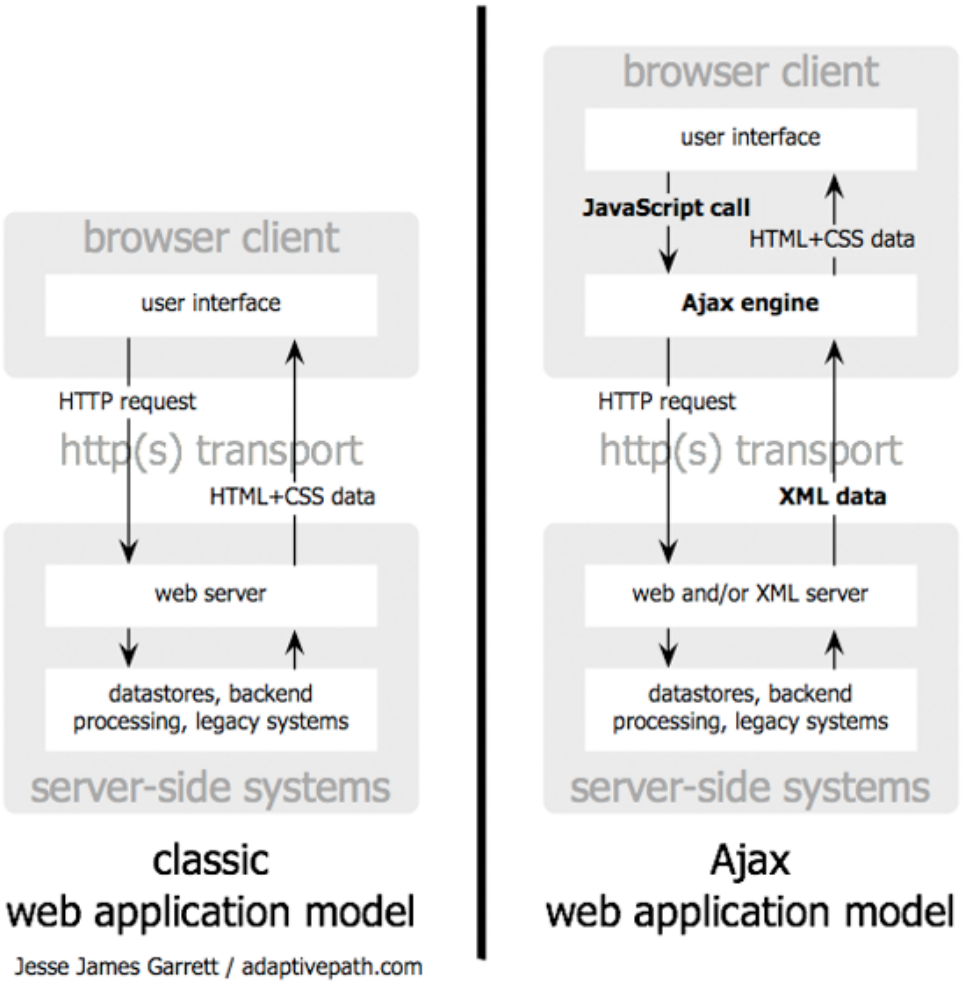


Figure 2.28: AJAX web application model compared to the classic one.
© Jesse James Garrett / adaptivepath.com

From the point of view of the user, the experience is much less disruptive than the classic model, as seen on Figure 2.29 on the following page. When a user clicks a link or a button in a traditional web application, the user interface blocks, goes blank, and he has to wait until another page is fully reloaded if he wants to continue with another task.

In an AJAX application, when the user performs such actions, the interface does not block; instead the AJAX engine is notified and the interface is *instantly* available. Meanwhile, backstage data gets interchanged

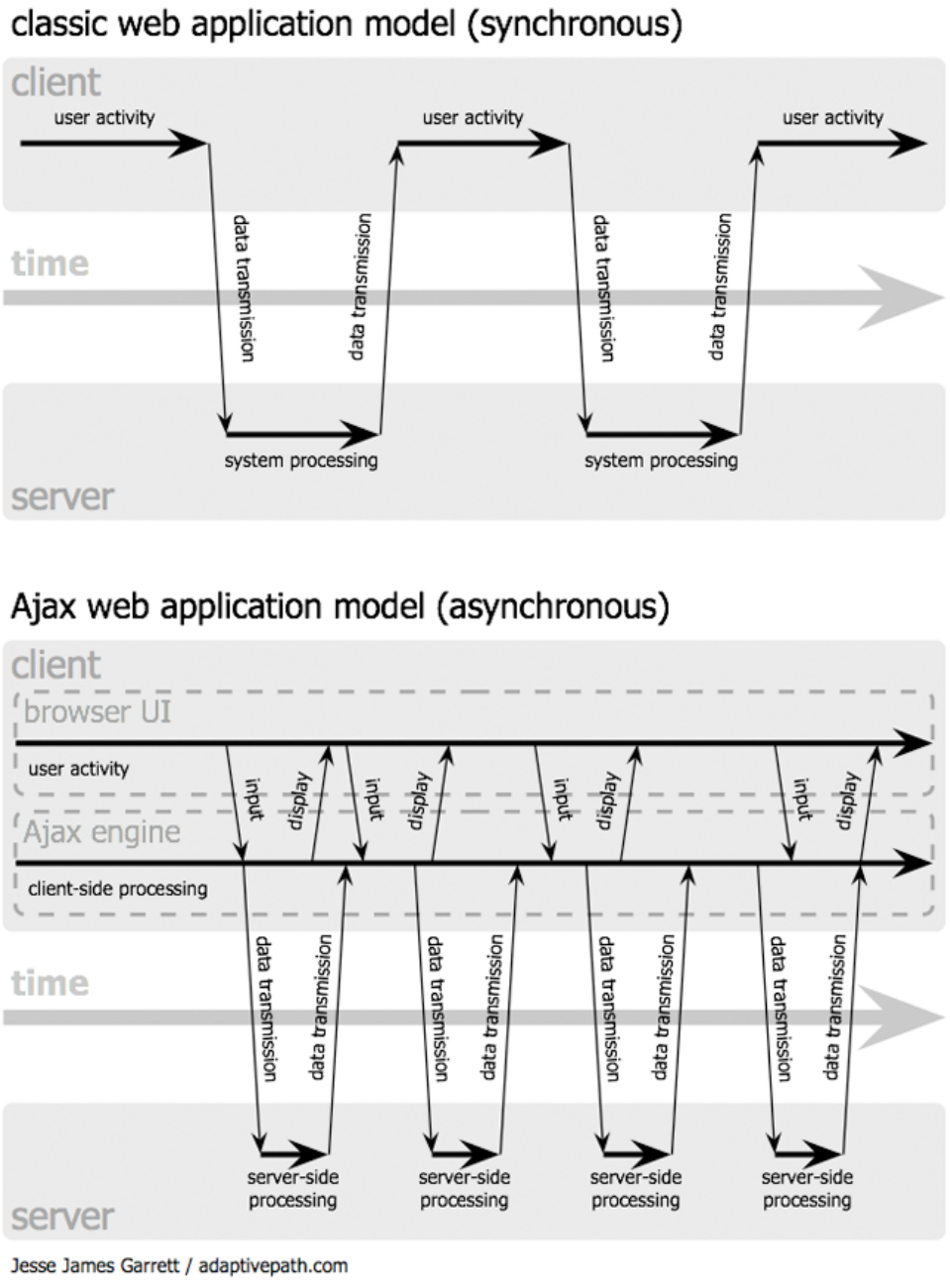


Figure 2.29: Flow of the actions in a AJAX web application.
© Jesse James Garrett / adaptivepath.com

and the client is updated, so that it can refresh the interface using the DOM and a mix of HTML and CSS.

Though it could appear that the application has more overhead now, the user will notice a much more responsive application. Also, the requests should be much faster than before, since the browser does not need to process again all the resources or redraw the whole page, and the server only needs to generate fragments of data.

However, there are some drawbacks to this approach. The first one is that the browser offers no feedback whatsoever to the user, but that is easily solved: the interface needs to reflect some visual feedback, like a spin ball. The second one is that the application will break the history of the user, since no new page has been served, it cannot go back to the previous state or even link to it. This has also been solved by the HTML5 History API, that allows to manipulate the browser history.

The third and hard one is that users with JavaScript disabled will not be able to use this web application at all. The best practice is to offer an alternate version with plain HTML pages, but sometimes that is not possible or it makes no sense. In the case of this project, since it was mostly experimental and not oriented for mainstream users, it was decided that such alternative would not be developed.

How XMLHttpRequest works

First of all, a XMLHttpRequest object must be created, and a custom function that will be act as a callback must be set in its `onreadystatechange` property. The next step consists of opening the connection to the server, with its `open()` method, specifying at least three parameters: first the HTTP method to use (`"GET"` or `"POST"`), then the url that we wish to contact and finally if the requests should be asynchronous or not. This last parameter is usually set to `true`, otherwise the user interface will block.

From now on, this object can make as many requests as it needs. To make a request its `send()` method is used, with additional data if the method is `POST` or with no data (or `null`) if the method is `GET`. Additionally, and only if needed, the `setRequestHeader()` method could be used to modify the headers of the HTTP request. If the request is asynchronous, the flow of the program will continue normally; if not, the program will block until the

server sends all the data.

Some time later, the server will answer with the response data. Then, the callback specified in the `onreadystatechange` property will be called, with all the needed data already updated in the XMLHttpRequest object. Actually, this callback will be called not only once but several times, each time reporting the progress of the request. To be sure that the request was completed it is need to check that the `readyState` property is exactly 4, and to know that the requested resource is ok that the `status` property is 200 (this is the HTTP state response).

If after that all was according to plan, the plain data will be accessible from the `reponseText` property. Also, if the response is in XML, the `reponseXML` property is also available with the parsed XML tree (the reason of the X in AJAX). The callback will then usually make some DOM manipulation to reflect the change in the interface.

Given the usefulness of this technique, most third-party libraries have implemented wrappers and more straightforward APIs to cover more use cases. Also, there are different variations, for example the current trend is not to use the XML format but to transmit everything in JSON, a much more efficient markup language that is native to JavaScript.

JSONP

By design, XMLHttpRequest carries a severe restriction: it is only allowed to request URLs from the same domain. There are security reasons for this decision, but in this mashup golden era it hinders a lot of its purposes. Thankfully, another technique has been popularized: JSON with padding (JSONP). Less elegant than AJAX but equally effective in most situations and without that ugly restriction.

The concept is simple and it is based on the fact that it is possible to add external scripts to a webpage. Just add a `script` tag with the desired URL and it will be executed; use the DOM and that script can be dynamically added after the page is loaded, just like AJAX. Of course, that script must be written in JavaScript, so that explains why JSON is used to pass the required data.

Listing 2.7 on the next page shows how some data could be expressed in JSON so that it can be used as an AJAX response. A question appears,

how is that data going to be executed in order to access it? The answer is by telling the external server to wrap it up in some function that we have define in our JavaScript code. The way to tell that to the other server is to add the name of the function to the requested URL as a parameter (usually called `jsonCallback` or just `callback`).

Listing 2.7: Some JSON data

```
{
  name:   "Kermit",
  animal: "Frog",
  age:    56,
  height: 45
}
```

For example, if we tell the other server that the function is called `doSomethingWithData()`, it can then wrap it up in a way that the data is the parameter for that function (see Listing 2.8). Since this code will be executed in our web application, it will call that function with that data at the exact moment the file is received and parsed. Therefore, that function must be defined in the global object, so it can process the received data without any problem.

Listing 2.8: Same JSON data wrapped in a custom function

```
doSomethingWithData({
  name:   "Kermit",
  animal: "Frog",
  age:    56,
  height: 45
});
```

Obviously, the server must explicitly support this technique, otherwise it is impossible to receive and change the server response without the use of proxies. Also, JSONP should only be applied with trusted third parties, since any malicious code could be injected in the page. The last security concern is that `POST` is not *supported*, so any parameter has to be passed

using `GET`, i.e., adding the parameters to the URL. However, due to its utility, almost all the popular libraries have similar tools that homogenizes the use of AJAX and JSONP.

2.7 JavaScript Framework: MooTools

So far, JavaScript seems a pretty powerful tool, focused on a limited scope but enough to make advanced web applications. Sadly, in the real world additional tools are needed to obtain a certain level of productivity. So let's briefly discuss why bringing yet another component to the application.

2.7.1 Why Use a JavaScript Framework?

Most web application rely on JavaScript frameworks, some are community driven efforts and others are custom made for an specific organization. All of them have as first goal to reuse pieces of code in common tasks, something even more important in JavaScript as it can be a very quirky language. In our case, there are several reasons that lead to using a well-stablished framework:

- Because we want to support different browsers. If we do not use a framework a lot of time would be spent debugging the *huge* differences between IE and the rest of the browsers. Popular frameworks have been tested by thousands of developers, so it is less probable that we fall into a bug.
- Because we want to speed up the development. Usually these frameworks cover several holes in the JavaScript specification that allows us fixing common issues with less code. Covering those holes by ourselves would be a waste of time, since in this project performance is not crucial.
- Because we want the interface to have advanced effects. We could just search for several scripts that makes one individual effect, but

that will result in redundancies, differences in quality code and waste time in searching.

In the end, all of that means that we can focus on just writing our application, avoiding reinventing the wheel over and over.

2.7.2 Making the Decision

By the previous standards, we have plenty of options to choose from: jQuery[*], Prototype[†], Dojo[‡], YUI[§], GWT[¶], Ext JS[‖], etc. Overall, these are very popular and they offer high quality and plenty of functionality, while maintaining a similar performance. However, for this particular project, and after some consideration, MooTools[**] was considered the best option. This decision was backed up by these reasons:

Compact It has a low footprint on the site load because it is reasonably lightweight for the functionality it offers. Particularly, it is more optimized in this aspect than Prototype, YUI or Dojo, but then it was also slightly more compact than jQuery.

Modular-Based Because of that, the installation can be customized to get only the modules we need, and the creation of our own extensions is easier.

Compatible It has been tested with most browsers: IE 6+, Firefox 2+, Opera 9+, Safari 3+ and Google Chrome 4+.

Functional It offers all the functionality required for the first phase of the project: drag&drop, resizing, animations, etc.

It also offers other functionality like AJAX support, `Hash` creation or `Cookie` handling, that ease the development in different browsers.

[*] http://jquery.com/
[†] http://www.prototypejs.org/
[‡] http://www.dojotoolkit.org/
[§] http://developer.yahoo.com/yui/
[¶] http://code.google.com/webtoolkit/
[‖] http://www.extjs.com/
[**] http://mootools.net

Object-Oriented By adding *Classes* to JavaScript, an abstraction that it is perfect for this application, since the server code is written is Java.

This way, we can use similar concepts both in the server and in the client. Moreover, the inherited code for ScaleNet already used JavaScript objects.

Extensive It also has a repository for official plugins called MooTools More (with similar code quality and documentation to the MooTools Core) and other third-party plugins can be found in the web.

Well-documented It has extensive documentation for every class of the framework.

Well-structured Its structure is perfect for a professional web application. Frameworks like jQuery are more focused in reducing the lines of code that in encouraging robust coding.* MooTools also helps reducing the lines of code, but it has more tools for writing code in a very modular, reusable and robust way, for example by using classes and other abstractions.

It also improves the readability of the code, something hard to do in JavaScript. Another important point of this framework is that it is based on prototype extensions (mainly DOM extensions), so the syntax is very Object-Oriented and the code seems very clean.

Used by the APE server So if we use that component, it will be very straightforward to write extensions in JavaScript also in the server. This will mean that we could use the same coding style and the same tools in the server as in the client.

Previous experience with jQuery resulted in quite faster development, but with time the solutions were hard to maintain without putting a lot of effort. With MooTools, several architectural design decisions like the use of *Classes* and *options* suited perfectly a non-trivial application like this one.

*A MooTools developer further discussed this in: http://jqueryvsmootools.com/

2.7.3 MooTools Core

The first thing to know about MooTools is that works by *monkey-patching* the native objects. This means that it modifies the prototype of that objects and extends or changes its functionality.

Some experts consider this to be a very bad practice because code from different parties can easily collide. However, this JavaScript feature is very powerful and in good hands it turns out extremely convenient. Due to this, it is very straightforward to write code with MooTools, and the resulting code do not have to look different from raw JavaScript.

Figure 2.30: MooTools logo

Under some circumstances these extensions just patch native methods that are not available in all browsers so that the developer can use them avoiding compatibility headaches. These additions are meaningfully named and can be organized into eight categories:

Core Traditionally MooTools declared several utility functions in the global scope, but these have been deprecated in favor of equivalents methods in native objects. Last version (1.3) only keeps a handful of them, mostly for type checking and extending prototypes.

Types Five important native types have been supercharged with a myriad of utility functions, filling a lot of holes in the JavaScript specification: to deal with collections and iterators (`Array`), to manipulate strings (`String`) and numbers (`Number`), to modify and custom call functions (`Function`), to add information to events (`Event`) and even to modify the properties in any object (`Object`).

Browser A new object (`Browser`) is created with all the information about the browser and its environment conveniently organized. This not only includes the browser version, the installed plugins and the user platfom, but it also detects some key features.

Class This is probably the heart of MooTools. `Class` is an object that encapsulates all the prototype-based inheritance system into the much more intelligible classic OOP. Basically, a Class is just an object

with shortcuts to simulate traditional class inheritance and interface implementation.

This does not hide any good side effect of the prototype system, for example a class can be modified and extended in any time: it is as dynamic as any JavaScript object. Simply, it is easier to use for a programmer that prototypes.

Listing 2.9: MooTools class definitions

```
var Animal = new Class({
  Implements: [Options, Events],
  options: {
    name: "Unnamed",
    pace: 0,
    children: []
    // onWalk: $empty
  },
  initialize: function(options) {
    this.setOptions(options);
    if (this.options.pace > 10)
      this.fast = true;
  },
  walk: function(distance) {
    for (var i = 0; i < distance; i += this.pace) {
      this.fireEvent('onWalk', distance);
    }
  }
});

var Horse = new Class({
  Extends: Animal,
  initialize: function(options) {
    options.pace = 20;
    this.parent(options);
  }
});
```

On top of that, three powerful abstractions are built into the framework and appear in most other classes (that implement them):

Options Handy way of dealing with settings within an *instance*[*], that is, attributes that may be optional (non-optional attributes could be defined as plain properties). By using a `Hash` to store all these properties, constructors only need one parameter to hold any combination of them.

Because its options and the default values must be defined at the *class declaration*, the framework can transparently merge both hashes. Therefore, since the developer does not need to handle this task anymore, it results in very clean constructors while resulting in a pretty extensible solution.

Events The concept of a event is greatly stretched in MooTools, as any class can define and fire custom events so that its instances can hook methods in the code of its parents. It integrates well with MooTools options, so it is very easy to declare and use them.

Chain This abstraction is designed to chain pieces of code to be executed asynchronously but in order. The usual example is to deal with animations in several steps, but it can be applied to a lot of different problems. Since JavaScript is single-threaded but asynchronous by nature, it is very welcomed to have an alternate way to arrange chunks of code with a lower priority to the background while preserving certain order between them.

Element As any other good modern framework, MooTools offers extensive support for DOM manipulation. It has two global shortcut functions baked in: the dollar function `$()` acts mostly as an alias for `document.getElementById()`, while `$$()` selects an array of DOM elements based on the specified CSS selector.

These two functions returns objects of type `Element`: DOM elements supercharged with utility extensions. For example, the constructor allows to quickly build an element with its own attributes, styles, size

[*]Since everything is an object in JavaScript, there are not really instances. In this context, an instance is an object cloned from a MooTools class, usually with `new` statement defining particular values.

and events at once. Not only it is more expressive but it handles for us developers the differences between all supported browsers.

Fx In the beginning, MooTools was only a lightweight library to add visual effects, but with time it became a full-fledged framework. Consequently, it is no surprise to say that it has a very robust animation system in place.

The base class is `Fx`, but normally developers should use `Fx.Tween` (to animate one property in an element) or `Fx.Morph` (to animate more than one property at the same time). There are also shortcut methods in `Element` for simple and quick animations. Apart from those classes, `Fx.Transitions` offers a broad collection of tweening transitions.

The idea behind this effect library is that several key steps are specified and then additional steps are generated to fill the time. Depending on the type of the effect, intermediate values are calculated and applied using CSS at a custom-spaced time interval to simulate the animation.

Request An AJAX wrapper that accumulates many options. There are subclasses to deal with HTML and JSON responses, and generally it takes a lot of work when setting AJAX requests. For example, there are shortcuts in the `Element` class that with a simple one-line method can completely replace an element with a remote fragment.

Utilities Four classes that have no place in previous categories: a cookie handler, a JSON encoder/decoder, a `domready` event (that springs when the, ehem, DOM is ready) and a Flash bridge (`Swiff`).

2.7.4 MooTools More

In a separate package, official plugins outside of the previous categories have been compiled. This includes advanced pieces of code that apply in too specific situations to be included in the main distribution, like form handling, interface widgets, advanced effects and other extras. There are too many to be explained in this document, but the most important ones used in the project could be quickly enumerated:

Class.Occlude This class implements the singleton pattern, and ties the resulting object to a predefined DOM element.

Hash A simple hash implementation, with methods that make it more useful to hold collections than a simple JavaScript objects.

Element.Position / Element.Measure Handy wrappers that calculate the exact dimensions of an object.

Fx.Elements To animate several elements at the same time.

Fx.Accordion A visual effect suited to make room for several elements in a limited space: as one element expands, the others gradually collapse.

Fx.Move Another visual effect to move an element from one location to another. Positions need not to be defined in coordinates, they can rather be automatically calculated from target elements.

Drag / Drag.Move To easily add drag&drop capabilities, making use of events to treat all possible outcomes. It also offers shortcuts to make an element automatically draggable and/or resizable.

Request.JSONP Similar class to `Request`, but using the JSONP technique to retrieve remote data from other domains.

Assets To dynamically load scripts, stylesheets, images and other resources.

Hash.Cookie This class handles the creation of a cookie that will contain a plain hash.

2.8 Push Server: the APE Server

Because of technical limitations of traditional browsers, it is not trivial to develop real time applications with JavaScript. More precisely, since it cannot directly push data from the server to the browser, the only native solution is to poll in regular time intervals — an inefficient approach.

The apprehended solution in the old system was to embed a Java applet solely to communicate with the server. The Java API for applets includes support for sockets, so it is possible to pass data in real time. However, the big drawback is that it needs the Java plugin to be installed in browsers, and there is no plugin for mobile browsers — neither in iOS nor in Android.

It was decided that mobile browsers must be supported by the second iteration of this project, so a new solution native to those browsers should be considered, getting rid of this Java applet.

2.8.1 Comet

Shortly after AJAX was popularized, another technique —called in contrast Comet— was developed to solve the particular use case of pushing data from the server to the client. Since then, several alternatives for creating real-time applications have been developed:

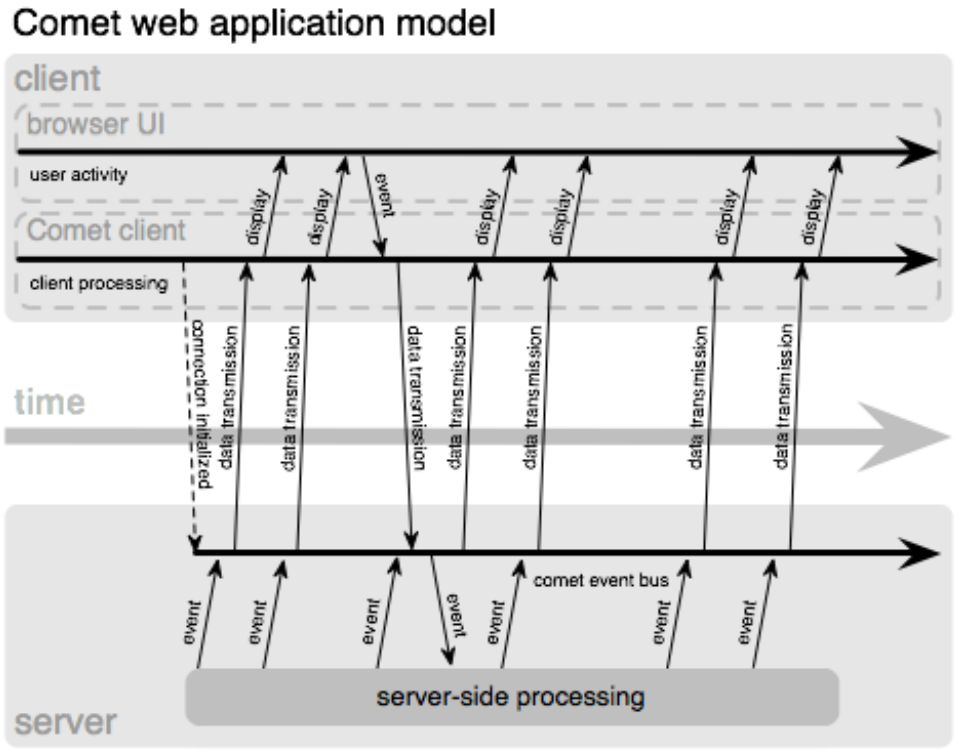


Figure 2.31: Typical flow in a Comet application (compare with Figure 2.29 on page 92)
© Alex Russell / infrequently.org)

WebSocket API An HTML5 extension[*] easy to use that just offers a socket to any server. This would be the perfect choice (and it should be chosen in the future), but at the moment it severely lacks support among the supported browsers, so it cannot be considered.

Socket.IO Since real sockets in browsers are out of the equation, an application with both browser and server components is the only way to go. One of the options with a brighter future relies on NodeJS[†], an effort to bring JavaScript to the server.

Socket.IO[‡] is a simple software that simulates real sockets in the browsers and uses NodeJS for its server component. It holds quite interesting ideas, but it was discarded because then it was in a pretty immature state.

CometD This is a similar approach by the Dojo Foundation[§] (so it works well with the dojo framework), creating a new protocol called Bayeux[¶]. In a quick glance it was rejected because it seems too complex for this task, and it could end in adding an additional framework in the mix.

APE server And finally we get to the winner of our contest. The APE project[‖] is a solid full solution with two components (server/browser), and it is focused on supporting real time data streaming. Just visiting its website explains why it seems like a better solution, because of the extensive documentation and well-explained examples — from simple to advanced ones.

One of the reasons why it was chosen it that it offers many layers of tinkering. If we only want a socket to a existing server, it has a proxy socket built so it is not needed to write any additional server code. But if we need to develop an advanced application, custom modules for the server can be written in JavaScript.

The other big reason is that it is written with MooTools, so bringing APE to the table bears little overhead for the client code. In the server,

[*] http://dev.w3.org/html5/websockets/
[†] http://nodejs.org/
[‡] http://socket.io/
[§] http://cometd.org/
[¶] http://svn.cometd.com/trunk/bayeux/bayeux.html
[‖] http://www.ape-project.org/

a hypothetical custom module could benefit from having the same framework as in the browser.

After considering all options, the APE project looked the most promising one. Eventually, just the proxy socket was needed, so it was merely a drop-in replacement for the Java applet. In any case, as its server deployment (see § 3.9 on page 190) consists on almost exclusively installing the Debian APE package, it results in an elegant and painless solution.

2.8.2 How the APE Server Works

As we said, the system needs two components: one to be installed in the server and a script to be included in our web application. The first component is a typical web server that listens upon a port, with the special peculiarity that only understands the APE protocol.

In that server, modules can be written in JavaScript (or even C), with a convenient API to access common web resources like sockets, pipes or MySQL connections. There are some modules and plugins implemented by default, like one that acts as a proxy for TCP sockets, or other that redirects data from a server application to the client. With those two simple modules a lot of applications can be written without needing a custom module.

Figure 2.32: Real official APE documentation

The server maintains a list of named channels; each channel holds one or more users that can write and read from that channel. Again, every user can have more than one connection, for example a user that has two tabs open in the same browser, or that have two sessions in two different devices at the same time. For this, the server DNS must be

configured to answer to multiple dynamic subdomains (`1.ape.domain.com`, `2.ape.domain.com`, `42.ape.domain.com`, etc).

In the browser, a set of scripts must be added to our web application so that they can talk with the backend: the APE JavaScript Framework (JSF). The configuration is very basic: it only needs the URL for the rest of the scripts and the base domain of the APE server.

Listing 2.10: TCPSocket usage in APE JSF

```
var client = new APE.Client();
var socket;

client.load();
client.addEvent('load', function() {
  this.core.start();
});

client.addEvent('ready', function() {
  socket = new this.core.TCPSocket();
  socket.open(host, port);
  socket.onopen = function() {
    socket.send('hello world');
  };
  socket.onread = function(msg) {
    console.log('New message: ' + msg);
  };
  socket.onclose = function() {
    socket = null;
    console.log('Connection with the server closed.');
  };
}

window.addEvent('unload', function() {
  socket.close();
}
```

Then the APE client can be created, the connection opens and the two

parties start exchanging messages. Messages are formatted as JSON arrays that contains two groups of objects: *commands* and *raws*. The former ones come from browsers to the server while the latter ones go the other way around.

Commands should be taken as actions that the clients want to accomplish, like opening the connection or sending some data. Commands are composed by a name, a challenging number (increased each time, to numerate the messages) and optional parameters. Depending on the name, the message is processed by the very APE server or served to a custom module that will answer to that action.

Raws can be expressed as data sent by the server, indeed it is composed by a name, a timestamp and the data as a JSON object. Again, the name reveals the purpose of the transmission and the data format. It is not uncommon for a message (that is, a JSON array) to contain more than one raw.

To send a command, the client creates a `GET` or `POST` request to a increasingly numerated subdomain, encoding the JSON array as the only parameter. When there is data to send, the server answers and the raw(s) can be found in the body response.

2.8.3 Transport Methods

Though it is not completely essential to know how exactly the APE server works in the backend, it should be understood how it earns its push capabilities. There are four transport methods implemented that the developer can choose from, but each of them has its strengths and weaknesses.

Long-polling This is the default transport method (so it will be selected unless noted otherwise), and it works in pretty much every JavaScript-enabled browser. It is based on AJAX but with a twist: the client request a resource but the server keeps the HTTP connection open until it has something to say.

When there is data to send (or after a certain timeout), the server sends the response. Then, the client immediately re-request the same resource, so the server has always a connection open to the browser.

Of course, the main drawback of this approach is that it is more like a hack, so it could be done more efficiently.

JSONP Similar to the previous one but over JSONP instead of AJAX, so it offers cross-domain possibilities.

XHRStreaming Also very similar to the first one, with the exception that the server do not close the connection when it has information to send. Therefore it only needs one connection, so it is more efficient. Sadly, this only works with recent browsers.

WebSocket As said before, this only works in a few new browsers, so it does not make sense to support it just yet if most times it is going to fallback to the default transport method.

2.9 Mobile Web Development

Web development has been, from the beginning, device independent. Every device can access the same page and format it differently to a certain screen resolution. Of course some of the less powerful devices suffered restrictions, since they could not handle JavaScript, full CSS or even images.

With the rise of touch devices like the iPhone or the Android phones, it appeared a new wave of mobile browsers capable of render websites almost as exactly as desktop browsers. The beautiful thing about these new mobile browsers is that they are all based on the Webkit engine.

They can display more or less features, but they are somewhat homogenous in their implementation. Also, since they are very recent, modern standards like HTML5 and CSS3 are widely supported.

The most important thing to notice is that the user input is notably different from a traditional computer. Instead of using a mouse to go to an exact pixel in the screen, the user directly touches the display with his fingers. This calls for new paradigms, since not all interfaces are cut for this kind of interaction:

- Buttons must be bigger, since fingers are not as precise as mouses.

Figure 2.33: An iPhone, iPad and Android, the three most popular mobile platforms that run Webkit

- The whole interface need to be at a certain size, since a good mobile web application should not require the user to zoom to see the content. This usually means less information in the screen.

- More than one finger can be used, in contrast to only one mouse in the desktop. That enables several new gestures, like pinching, that are newly available to web applications.

- By default, drag&drop is reserved to scroll the page. If the web application needs to use it, like the current web interface does, it has to capture all finger events and re-create the scrolling experience.

- Double tapping is a movement that makes more sense on these devices than in desktop web applications, and it could be seen as a right click gesture on the desktop.

- There is also very important to notice that the screen could be oriented in portrait or landscape, so the interface must be adaptable.

The current ScaleNet interface, being optimized for big screens, it is not even usable on these devices. Even when zooming manually, most actions rely on drag&drop and therefore are not even accessible, so it is

impossible to move or delete a session. It is clear that if these devices have to be supported, a mobile version needs to be developed fixing those issues.

To deal with these peculiarities, there are several additions to the web stack tools to make nicer web applications. This applies specially in iOS devices, but Android also understands some of them. The most useful and distinct features are:

Touch events Since mouse events do not make any sense there, special custom events for *touches* and gestures are available.

Viewport To deal with the fact that most websites are developed for bigger screens, by default web applications are rendered against a virtual viewport wider than the real screen. Then, it is up to the user to zoom in to read the page.

At the same time, web applications specially designed for these devices can detect them and define the size of the viewport, so the user do not have to zoom to have a decent experience. Another solution is the use of CSS3 conditional styles and media queries.

Special forms While maintaining most of the form elements, there are some missing —like file uploading— and some new interface elements —like select boxes. There are extra features that can be triggered like autocompletion, autocorrection, autocapitalize, custom keyboards for numbers/emails, etc. Moreover, since the virtual keyboard could appear, web applications should be prepared for having very little room.

App icon Both iOS and Android allow the creation of shortcuts to display in the home screens. For that, it is possible to define the icons that those shortcuts can use.

Hiding the browser User Interface (UI) Once a web application is launched from an iOS shortcut, it can dictate if the browser UI should be hidden or not. In the case that the UI is hidden, it can also define which aspect should the menubar have (light, dark or transparent).

Startup image Native iOS apps can specify a startup image to be displayed when loading the application. More than being a visual triviality, it is

designed to display an empty interface so that the user *feels* that the application loads faster. Web applications can also set such an image with a meta tag.

Lastly, it has to be noted that there are mobile JavaScript frameworks that have been developed with these restrictions in mind. Some of them try to bring a lightweight framework like Zepto.js, others try to develop a full solution for web applications like Sencha or JQTouch, and others just try to ease the development of specific features.

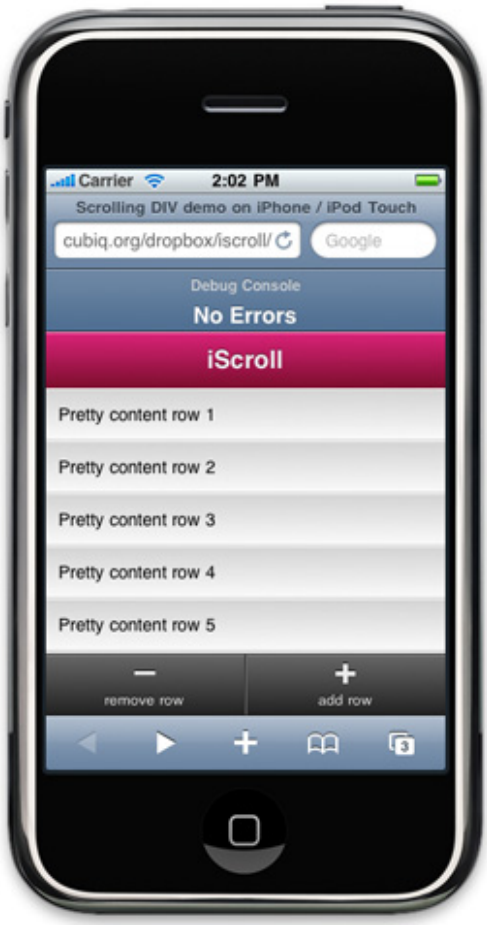


Figure 2.34: iScroll in action

For this project, one of those frameworks was used: iScroll*. The goal of this script is to provide a scrolling effect inside an element, since the only scrolling available in these mobile browsers is the page scrolling. With mostly three lines of setup, this script simulates that native scrolling effect with some advanced CSS3 properties (animations and so on).

The common use case for this effect is the implementation of a bottom toolbar, like in native apps. These mobile browsers do not offer support for fixed positioned elements, so this is the only way that this kind of toolbar can be simulated.

*http://cubiq.org/iscroll

Chapter 3

Development and Testing

Walter : Did you learn nothing from my chemistry class?

Jesse : No. You flunked me, remember? You prick! Now let me tell you something else. This ain't chemistry —this is art. Cooking is art. And the shit I cook is the bomb, so don't be telling me.

Walter : The shit you cook is shit. I saw your set-up. Ridiculous. You and I will not make garbage. We will produce a chemically pure and stable product that performs as advertised. No adulterants. No baby formula. No chili powder.

Jesse : No, no, chili P is my signature!

Walter : Not anymore.

Pilot

Breaking Bad

3.1 Introduction

Last chapter was focused in work done by others; this chapter brings light to the actual work done in this project, a new iteration on the PNAI. As the previous compilations were needed to understand the environment of this work, the following sections explain which additions were requested and developed.

User requirements are the first thing to clarify, extracted from the received from tutors and other T-Labs members previously involved in the project. This feedback was continually given as the project evolved, mostly for design concerns or when adding additional technologies, not for the implementation itself.

Later on, the system design is elucidated with diagrams that highlight the changes respect the base version, detailing the implementation process. Furthermore, noteworthy hiccups found in the implementation process are specified using the resulting algorithms and solutions.

3.2 New Requirements

There are new requirements that need to be satisfied within the redesigned PNAI interface. These requirements were extracted in several team meetings as the work advanced. As explained in § 1.3 on page 4, they could be divided into roughly three phases: an initial phase – a full redesign –, a second phase – adding more functionality – and a third phase – implementing the mobile version.

As this was a new iteration over an already developed product, there were multiple requirements that were already set and implemented. Those requirements are not discussed since they are out of the limits of this work. Therefore, this is a list of only the new requirements resulted from the team feedback:

Table 3.1: User requirement 1 – Redesign interface

USER REQ. 1	**Redesign interface**
Priority	High
Phase	Initial phase
Description	Overall design must be modernized, keeping the same elements, but giving a fresher, more professional look. Animations could be included to give a more responsive experience.

Table 3.2: User requirement 2 – Adapt to different resolutions

USER REQ. 2	**Adapt to different resolutions**
Priority	High
Phase	Initial phase
Description	The new design must not be static like the previous one, it must adapt when the user resizes the browser so that it fills in the available space and does not trigger browser scrollbars.

Table 3.3: User requirement 3 – Show device name

USER REQ. 3	**Show device name**
Priority	Medium
continued on next page	

Table 3.3: User requirement 3 – Show device name (continued)

USER REQ. 3	**Show device name**
Phase	Initial phase
Description	The device name (set by the user) must be displayed besides the device representation.

Table 3.4: User requirement 4 – Load real user devices

USER REQ. 4	**Load real user devices**
Priority	Low
Phase	Initial phase
Description	The old interface always loaded the same three devices, independently of the user and its real registered devices. This new interface shall support an undefined number of devices, and load them dynamically at the beginning. It also shall create or delete devices at any moment when the backend commands it.

Table 3.5: User requirement 5 – Put screenshots in session icons

USER REQ. 5	**Put screenshots in session icons**
Priority	Low
Phase	Initial phase
continued on next page	

Table 3.5: User requirement 5 – Put screenshots in session icons (continued)

USER REQ. 5	**Put screenshots in session icons**
Description	The old interface always showed the same generic icon for all sessions. This new interface shall show the real screenshot already stored for each video file.

Table 3.6: User requirement 6 – Reorganize devices

USER REQ. 6	**Reorganize devices**
Priority	Medium
Phase	Initial phase
Description	User must be able to move around its devices in the PNAI interface, reordering them as they wish. In any case, all devices must be fully visible at any time.

Table 3.7: User requirement 7 – Resize devices

USER REQ. 7	**Resize devices**
Priority	Low
Phase	Initial phase
continued on next page	

Table 3.7: User requirement 7 – Resize devices (continued)

USER REQ. 7	**Resize devices**
Description	User must be able to scale its devices in the PNAI interface, making them bigger or smaller, within certain bounds to guaranty that sessions have room to be drawn inside them.

Table 3.8: User requirement 8 – Browser compatibility

USER REQ. 8	**Browser compatibility**
Priority	High
Phase	Initial phase
Description	It shall be compatible with latest versions of all popular devices: IE 7+, Firefox, Opera, Safari and Google Chrome. The only exception shall be IE 6, for practical reasons.

Table 3.9: User requirement 9 – Integrate the IPTVplus interface

USER REQ. 9	**Integrate the IPTVplus interface**
Priority	High
Phase	Second phase
continued on next page	

Table 3.9: User requirement 9 – Integrate the IPTVplus interface (continued)

USER REQ. 9	**Integrate the IPTVplus interface**
Description	The IPTVplus interface (see Figure 2.14 on page 48) must be implemented as a sidebar inside the PNAI interface. That way, the user must be able to buy new content within the PNAI interface, all just in one page.

Table 3.10: User requirement 10 – Buy content onto a device

USER REQ. 10	**Buy content onto a device**
Priority	Medium
Phase	Second phase
Description	When the user buys new content from this PNAI interface, it shall be able to select another device as the destination to directly play that content other than the default device.

Table 3.11: User requirement 11 – Buy content onto a buddy

USER REQ. 11	**Buy content onto a buddy**
Priority	Medium
Phase	Second phase
continued on next page	

Table 3.11: User requirement 11 – Buy content onto a buddy (continued)

USER REQ. 11	**Buy content onto a buddy**
Description	When the user buys new content from this PNAI interface, it shall be able to select another buddy as the destination to directly play that content.

Table 3.12: User requirement 12 – Resize/collapse sidebar

USER REQ. 12	**Resize/collapse sidebar**
Priority	Medium
Phase	Second phase
Description	The user must be able to horizontally resize the sidebar to a custom size, that must be less than 50% of the page width. Therefore, the IPTVplus interface (see Figure 2.14 on page 48) must be implemented as a sidebar inside the PNAI interface.

Table 3.13: User requirement 13 – Design and adapt a mobile interface

USER REQ. 13	**Design and adapt a mobile interface**
Priority	High
Phase	Third phase

continued on next page

Table 3.13: User requirement 13 – Design and adapt a mobile interface (continued)

USER REQ. 13	**Design and adapt a mobile interface**
Description	A new mobile interface must be created for popular modern smartphones with a touchscreen, specifically for the iOS and Android platforms. The interface shall retain all functionality of the PNAI desktop interface. Elements and abstractions must be adapted to mobile peculiarities so that the user experience is comfortable.

Outside of these requirements, there were some *implied* requirements, due to the experimental nature of this project: this effort was planned for showcasing purposes rather than for real customers. For example, delivering a product with certain quality but as fast as possible derives on some decisions like trying to not rewrite other parts of the system unless completely necessary.

3.3 Use Cases

From the point of view of the user, there will be not one but two different interfaces. Each one having particular restrictions and abstractions. Therefore, it is useful to distinguish between the additional use cases for the desktop interface, and the separate use cases for the mobile interface.

3.3.1 New Use Cases

From the previous requirements, there are new use cases that needs to be addressed, mostly from the second phase. For starters, three important

scenarios are related to the new content tab, to cover exactly the three use cases missing in Figure 2.5 on page 14.

Table 3.14: Use case 7 – Buy new content

Use Case 7	**Buy new content**
Actor	System user
Precondition	The user must have at least one device online. Also, the sidebar must be opened having the active tab set to *Content*.
Postcondition	A new session must start on the default user device, or if that device is offline, on one of the online devices. The user must be notified with a popup and the session icon must appear in the PNAI interface for that device.
Main Path (M)	**1.** User clicks on the *Buy* (*Play* if the content is free) button of the wanted content. **2.** A popup window appears asking for confirmation. **3.** The user confirms the action clicking on the button. **4.** The popup window closes. **5.** A popup appears to notify the user that the action is in progress. **6.** The content starts playing on the default device, or another online device if it is disconnected. **7.** The popup disappears and a session icon is created inside of the destination device.

continued on next page

Table 3.14: Use case 7 – Buy new content (continued)

USE CASE 7	**Buy new content**
Alternate Path (A1)	**3b.** The user cancels the action. **4b.** The popup window closes and action is cancelled.
Alternate Path (A2)	**7c.** There is an error with the server and the session does not initiate. **8c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **9c.** Action is cancelled.

Table 3.15: Use case 8 – Buy new content onto a device

USE CASE 8	**Buy new content onto a device**
Actor	System user
Precondition	The user must have at least one device online. Also, the sidebar must be opened having the active tab set to *Content*.
Postcondition	A new session must start on that selected device. The user must be notified with a popup and the session icon must appear in the PNAI interface for that device.

continued on next page

Table 3.15: Use case 8 – Buy new content onto a device (continued)

USE CASE 8	**Buy new content onto a device**
Main Path (M)	**1.** User starts dragging the content icon. **2.** A copy of the content icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** User drops the cloned content icon into another device that is online. **4.** A popup window appears asking for confirmation. **5.** The user confirms the action clicking on the button. **6.** The popup window closes. **7.** A popup appears to notify the user that the action is in progress. **8.** The content starts playing on that device. **9.** The popup disappears and a session icon is created inside of the destination device.
Alternate Path (A1)	**2b.** User drops the content icon into a blank space. **3b.** Action is cancelled.
Alternate Path (A2)	**4c.** The user cancels the action. **5c.** The popup window closes and action is cancelled.

continued on next page

Table 3.15: Use case 8 – Buy new content onto a device (continued)

USE CASE 8	**Buy new content onto a device**
Alternate Path (A3)	**8c.** There is an error with the server and the session does not initiate. **9c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **10c.** Action is cancelled.

Table 3.16: Use case 9 – Buy new content onto a buddy

USE CASE 9	**Buy new content onto a buddy**
Actor	System user
Precondition	The user must have at least one buddy online. Also, the sidebar must be opened having the active tab set to *Content*.
Postcondition	A new session must start on the default (or online) device of that selected buddy. The user must be notified with a popup and the session icon must appear in the PNAI interface for that buddy.

continued on next page

Table 3.16: Use case 9 – Buy new content onto a buddy (continued)

USE CASE 9	**Buy new content onto a buddy**
Main Path (M)	**1.** User starts dragging the content icon. **2.** A copy of the content icon appears under the user's cursor, and follows the cursor until the user drops it. **3.** Without releasing the icon, the user moves the mouse to the buddies tab, and keeps it there for a second. **4.** The buddies tab gets selected, revealing the buddies. **5.** User drops the cloned content icon into a buddy that is online. **6.** A popup window appears asking for confirmation. **7.** The user confirms the action clicking on the button. **8.** The popup window closes. **9.** A popup appears to notify the user that the action is in progress. **10.** The content starts playing on the default (or online) device of that selected buddy. **11.** The popup disappears and a session icon is created right to the destination buddy's name.
Alternate Path (A1)	**2b.** User drops the content icon into a blank space. **3b.** Action is cancelled.

continued on next page

Table 3.16: Use case 9 – Buy new content onto a buddy (continued)

USE CASE 9	**Buy new content onto a buddy**
Alternate Path (A2)	**5b.** User drops the content icon into a blank space. **6b.** Action is cancelled.
Alternate Path (A3)	**7c.** The user cancels the action. **8c.** The popup window closes and action is cancelled.
Alternate Path (A4)	**10c.** There is an error with the server and the session does not initiate. **11c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **12c.** Action is cancelled.

To handle the requirements from UR9 to UR12, a new sidebar needs to be implemented with extra functionality. This collapsible sidebar will have two tabs to control the content of the sidebar, triggering some new use cases.

Table 3.17: Use case 10 – Resize sidebar

USE CASE 10	**Resize sidebar**
Actor	System user
Precondition	The application is fully loaded.

continued on next page

Table 3.17: Use case 10 – Resize sidebar (continued)

USE CASE 10	**Resize sidebar**
Postcondition	The sidebar must be opened at the exact point the user released the mouse, within certain bounds.
Main Path (M)	**1.** User starts dragging the left border of the sidebar. **2.** The width of the sidebar grows or shrinks following the mouse, readapting the device list to that space in every step. **3.** When the user releases the mouse, the sidebar stays in that size.
Alternate Path (A1)	**3b.** The mouse goes to the half left part of the page or further to the left. **4b.** The sidebar stays filling only the 50% right part of the page. **5b.** When the user releases the mouse, the sidebar stays in that size (half of the page).

Table 3.18: Use case 11 – Collapse sidebar

USE CASE 11	**Collapse sidebar**
Actor	System user
Precondition	The sidebar must be opened.
Postcondition	The sidebar must be collapsed.

continued on next page

Table 3.18: Use case 11 – Collapse sidebar (continued)

USE CASE 11	**Collapse sidebar**
Main Path (M)	**1.** User clicks on the sidebar handler. **2.** The sidebar is collapsed and the devices are readapted in that new space.
Alternate Path (A1)	**1b.** User clicks on the currently selected tab. **2b.** The sidebar is collapsed and the devices are readapted in that new space.
Alternate Path (A2)	**1c.** User starts dragging the left border of the sidebar until its gets to the right part of the page (exactly to the 15% space in the right). **2c.** The sidebar is collapsed and the devices are readapted in that new space. **3c.** User releases the mouse.

Table 3.19: Use case 12 – Open sidebar

USE CASE 12	**Open sidebar**
Actor	System user
Precondition	The sidebar must be collapsed.
Postcondition	The sidebar must be opened.

continued on next page

Table 3.19: Use case 12 – Open sidebar (continued)

USE CASE 12	**Open sidebar**
Main Path (M)	**1.** User clicks on the sidebar handler. **2.** The sidebar is opened and the devices are readapted in that new space. **3.** The previous selected tab gets selected.
Alternate Path (A1)	**1b.** User clicks on any tab. **2b.** The sidebar is opened and the devices are readapted in that new space. **3b.** That tab gets selected.
Alternate Path (A2)	**1c.** User starts dragging the left border of the sidebar until its gets to the left part of the page (exactly the 85% space in the left). **2c.** The sidebar is collapsed and the devices are readapted in that new space. **3c.** The previous selected tab gets selected. **4c.** User releases the mouse.

Table 3.20: Use case 13 – Select tab

USE CASE 13	**Select tab**
Actor	System user
Precondition	The sidebar must be either collapsed or opened with the active tab set to the other tab.
Postcondition	The tab must be selected, i.e., the sidebar should show the content for that tab.

continued on next page

Table 3.20: Use case 13 – Select tab (continued)

USE CASE 13	**Select tab**
Main Path (M)	**1.** User clicks on the tab. **2.** The tab is selected and its content is changed to the content for that tab.
Alternate Path (A1)	**2b.** If the sidebar is collapsed, then the sidebar is opened. **3b.** The tab is selected and its content is changed to the content for that tab.

The initial phase requirements also need some use cases, although less critical than the others. For example, the ability of moving and resizing devices (UR6) can be divided into two use cases.

Table 3.21: Use case 14 – Move device

USE CASE 14	**Move device**
Actor	System user
Precondition	There is at least one registered device for that user.
Postcondition	The device is moved to a new position.
	continued on next page

Table 3.21: Use case 14 – Move device (continued)

USE CASE 14	**Move device**
Main Path (M)	**1.** User moves the mouse over the device he wants to move. **2.** Handlers for moving and resizing that device fade in. **3.** User starts dragging the handler or the title of that device (this title is covering the top of the user). **4.** The device moves following the mouse. **5.** When the user releases the mouse, the device stays in that position.
Alternate Path (A1)	**4b.** The mouse is moved out of bounds. **5b.** The device stays inside of those limits, in the nearest position to the mouse. **6b.** When the user releases the mouse, the device stays in that position.

Table 3.22: Use case 15 – Resize device

USE CASE 15	**Resize device**
Actor	System user
Precondition	There is at least one registered device for that user.
Postcondition	The device is resized.

continued on next page

Table 3.22: Use case 15 – Resize device (continued)

USE CASE 15	**Resize device**
Main Path (M)	**1.** User moves the mouse over the device he wants to resize. **2.** Handlers for moving and resizing that device fade in. **3.** User starts dragging the resize handler (at the bottom right part of the device). **4.** The device resizes following the mouse, keeping its relative ratio. These limits must ensure that the device icon is fully displayed. **5.** When the user releases the mouse, the device stays in that size.
Alternate Path (A1)	**4b.** The mouse is moved out of bounds. **5b.** The device stays inside of the fixed limits, in the nearest position to the mouse. These limits must ensure that the device icon is fully displayed and that there are enough space for the session icons to be drawn inside. **6b.** When the user releases the mouse, the device stays in that size.

3.3.2 Mobile Use Cases

All of the previous use cases are defined considering the desktop interface. However, as the third phase consists on developing another very different interface, more use cases have to be defined for that mobile version. Some of the new use cases, like resizing devices or the sidebar, do not need to be implemented, since those components do not make sense in the mobile interface.

The PNAI interface is only a frame inside of the ScaleNet interface, but the mobile version is a full application outside of any frame. So there are a few more things that need to be handle, like user identification actions. The rest of the actions, equivalent to the desktop version, have to be changed to follow more appropriate interaction patterns.

Table 3.23: Use case 16 – Login (mobile)

USE CASE 16	**Login (mobile)**
Actor	System user
Precondition	The application is fully loaded.
Postcondition	The user must be identified, and the interface with the rest of the actions must be shown.
Main Path (M)	**1.** User enters its name and password, and taps on the *Login* button. **2.** Credentials are checked. **3.** Credentials are correct, so the user interface loads with his devices and buddies.
Alternate Path (A1)	**3b.** Credentials are incorrect, so the same form is shown with a message to inform that the credentials are incorrect.

Table 3.24: Use case 17 – Logout (mobile)

USE CASE 17	**Logout (mobile)**
Actor	System user

continued on next page

Table 3.24: Use case 17 – Logout (mobile) (continued)

USE CASE 17	**Logout (mobile)**
Precondition	The application is fully loaded and it is not blocked, and the user is logged in.
Postcondition	The session must end, showing the login form.
Main Path (M)	**1.** User taps on the *Logout* button. **2.** The session is deleted. **3.** The login form is shown.

Table 3.25: Use case 18 – Select tab (mobile)

USE CASE 18	**Select tab (mobile)**
Actor	System user
Precondition	The application is fully loaded and it is not blocked, and the user is logged in.
Postcondition	The tab must be selected, i.e., the main view should show the content for that tab.
Main Path (M)	**1.** User taps on the tab. **2.** The tab is selected and the main view is changed to show the content related to that tab.

Table 3.26: Use case 19 – Stop a session of a device (mobile)

USE CASE 19	**Stop a session of a device (mobile)**
Actor	System user
Precondition	A session is already running on a device, and it is showing in the PNAI interface near that device's icon.
Postcondition	Session must terminate, i.e., the content must stop playing. The users must be notified with a popup and the session icon must be deleted from the PNAI interface.
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Stop* option. **4.** The popup menu disappears. **5.** A popup appears to notify the user that the action is in progress. **6.** The content stops playing. **7.** The popup disappears and the session icon is deleted from the view.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.

continued on next page

Table 3.26: Use case 19 – Stop a session of a device (mobile) (continued)

USE CASE 19	**Stop a session of a device (mobile)**
Alternate Path (A2)	**6c.** There is an error with the server and the content keeps playing. **7c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **8c.** Action is cancelled.

Table 3.27: Use case 20 – Stop a session of a buddy (mobile)

USE CASE 20	**Stop a session of a buddy (mobile)**
Actor	System user
Precondition	A session owned by the user is running on a device, and it is showing in the PNAI interface near that buddy's name.
Postcondition	Session must terminate, i.e., the content must stop playing. The user must be notified with a popup and the session icon must be deleted from the PNAI interface. The buddy is *not* notified, the content stops without warning.

continued on next page

Table 3.27: Use case 20 – Stop a session of a buddy (mobile) (continued)

USE CASE 20	**Stop a session of a buddy (mobile)**
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Stop* option. **4.** The popup menu disappears. **5.** A popup appears to notify the user that the action is in progress. **6.** The content stops playing. **7.** The popup disappears and the session icon is deleted from the view.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.
Alternate Path (A2)	**6c.** There is an error with the server and the content keeps playing. **7c.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **8c.** Action is cancelled.

Table 3.28: Use case 21 – Copy a session to a device (mobile)

USE CASE 21	**Copy a session to a device (mobile)**
Actor	System user

continued on next page

Table 3.28: Use case 21 – Copy a session to a device (mobile) (continued)

USE CASE 21	**Copy a session to a device (mobile)**
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface near that device/buddy. Also, there is another device online.
Postcondition	Session must be copied to that device, i.e., the content must be duplicated and played on that device. The user must be notified with a popup and the session icon must appear in the PNAI interface for the second device.
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Duplicate* option. **4.** The popup menu disappears. **5.** The interface changes to notify the user that now he has to select the destination device. **6.** If the devices tab is not selected, user taps on the devices tab icon. **7.** User taps on the device he wants. **8.** The interface changes back to the normal state. **9.** A popup appears to notify the user that the action is in progress. **10.** The content starts playing in the other device. **11.** The popup disappears and the session icon is created near the destination device.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.

continued on next page

Table 3.28: Use case 21 – Copy a session to a device (mobile) (continued)

USE CASE 21	**Copy a session to a device (mobile)**
Alternate Path (A2)	**6c.** User taps the *Cancel* button, or in the add content tab. **7c.** The interface changes back to the normal state. **8c.** Action is cancelled.
Alternate Path (A3)	**10d.** There is an error with the server and the content is not duplicated. **11d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **12d.** Action is cancelled.

Table 3.29: Use case 22 – Copy a session to a buddy (mobile)

USE CASE 22	**Copy a session to a buddy (mobile)**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface near that device/buddy. Also, there is another buddy online.
Postcondition	Session must be copied to that buddy, i.e., the content must be duplicated and played on the buddy's default device. The user must be notified with a popup and the session icon must appear in the PNAI interface near the name of that buddy. The buddy is *not* notified, the content plays without warning.

continued on next page

Table 3.29: Use case 22 – Copy a session to a buddy (mobile) (continued)

USE CASE 22	**Copy a session to a buddy (mobile)**
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Duplicate* option. **4.** The popup menu disappears. **5.** The interface changes to notify the user that now he has to select the destination buddy. **6.** If the buddies tab is not selected, user taps on the buddies tab icon. **7.** User taps on the buddy he wants. **8.** The interface changes back to the normal state. **9.** A popup appears to notify the user that the action is in progress. **10.** The content starts playing in the buddy's device. **11.** The popup disappears and the session icon is created near the destination device.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.
Alternate Path (A2)	**6c.** User taps the *Cancel* button, or in the *Add Content* tab. **7c.** The interface changes back to the normal state. **8c.** Action is cancelled.

continued on next page

Table 3.29: Use case 22 – Copy a session to a buddy (mobile) (continued)

USE CASE 22	**Copy a session to a buddy (mobile)**
Alternate Path (A3)	**10d.** There is an error with the server and the content is not duplicated. **11d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **12d.** Action is cancelled.

Table 3.30: Use case 23 – Transfer a session to a device (mobile)

USE CASE 23	**Transfer a session to a device (mobile)**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface near that device/buddy. Also, there is another device online.
Postcondition	Session must be transferred to that device, i.e., playback must be stopped at the source and started at the destination device. The user must be notified with a popup and the session icon must appear in the PNAI interface for the second device.

continued on next page

Table 3.30: Use case 23 – Transfer a session to a device (mobile) (continued)

USE CASE 23	**Transfer a session to a device (mobile)**
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Hand over* option. **4.** The popup menu disappears. **5.** The interface changes to notify the user that now he has to select the destination device. **6.** If the devices tab is not selected, user taps on the devices tab icon. **7.** User taps on the device he wants. **8.** The interface changes back to the normal state. **9.** A popup appears to notify the user that the action is in progress. **10.** The content stops playing in that device and starts playing in the other device. **11.** The popup disappears, the session icon is deleted from the device and created near the destination device.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.
Alternate Path (A2)	**6c.** User taps the *Cancel* button, or in the add content tab. **7c.** The interface changes back to the normal state. **8c.** Action is cancelled.

continued on next page

Table 3.30: Use case 23 – Transfer a session to a device (mobile) (continued)

USE CASE 23	**Transfer a session to a device (mobile)**
Alternate Path (A3)	**10d.** There is an error with the server and the content is not transferred. **11d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **12d.** Action is cancelled.

Table 3.31: Use case 24 – Transfer a session to a buddy (mobile)

USE CASE 24	**Transfer a session to a buddy (mobile)**
Actor	System user
Precondition	A session is already running on a device/buddy, and it is showing in the PNAI interface near that device/buddy. Also, there is another buddy online.
Postcondition	Session must be transferred to that buddy, i.e., playback must be stopped at the source and started at the buddy's default device. The user must be notified with a popup and the session icon must appear in the PNAI interface near the name of that buddy. The buddy is *not* notified, the content plays without warning.

continued on next page

Table 3.31: Use case 24 – Transfer a session to a buddy (mobile) (continued)

USE CASE 24	**Transfer a session to a buddy (mobile)**
Main Path (M)	**1.** User taps on the session icon. **2.** A menu appears over the session icon. **3.** User taps the *Hand over* option. **4.** The popup menu disappears. **5.** The interface changes to notify the user that now he has to select the destination buddy. **6.** If the buddies tab is not selected, user taps on the buddies tab icon. **7.** User taps on the buddy he wants. **8.** The interface changes back to the normal state. **9.** A popup appears to notify the user that the action is in progress. **10.** The content stops playing in that device and starts playing in the buddy's device. **11.** The popup disappears, the session icon is deleted from the device and created near the destination device.
Alternate Path (A1)	**3b.** User taps outside the menu. **4b.** The popup menu disappears. **5b.** Action is cancelled.
Alternate Path (A2)	**6c.** User taps the *Cancel* button, or in the *Add Content* tab. **7c.** The interface changes back to the normal state. **8c.** Action is cancelled.

continued on next page

Table 3.31: Use case 24 – Transfer a session to a buddy (mobile) (continued)

Use Case 24	**Transfer a session to a buddy (mobile)**
Alternate Path (A3)	**10d.** There is an error with the server and the content is not transferred. **11d.** The content of the popup changes to notify the user that there was an error with the server and the action could not be completed. After 5 seconds it disappears. **12d.** Action is cancelled.

3.4 Components

Most of the components remained untouched after this work was finished, so for most interfaces Figure 2.9 on page 28 is still a good approximation. The only exception is the removal of the Java applet in favor of two APE components: the client code (APE JSF) and the server code (APE Server).

Figure 3.1 on the next page shows how these additions reflect in the overall system, and how this new *proxy* does not affect components in the backend. Regarding this view, there are several things to consider:

- All web interface files (HTML, CSS, JavaScript) that were previously served by the OSGi server are now served by the Apache server. Exactly, they are in a folder alongside the PHP files for IPTVplus and other services.

  This change responds to development and testing convenience, since any change in those *static* files meant that the OSGi bundle had to be compiled, replaced and restarted. From now on, any change in those files will be immediately reflected, and the performance should be even better than before.

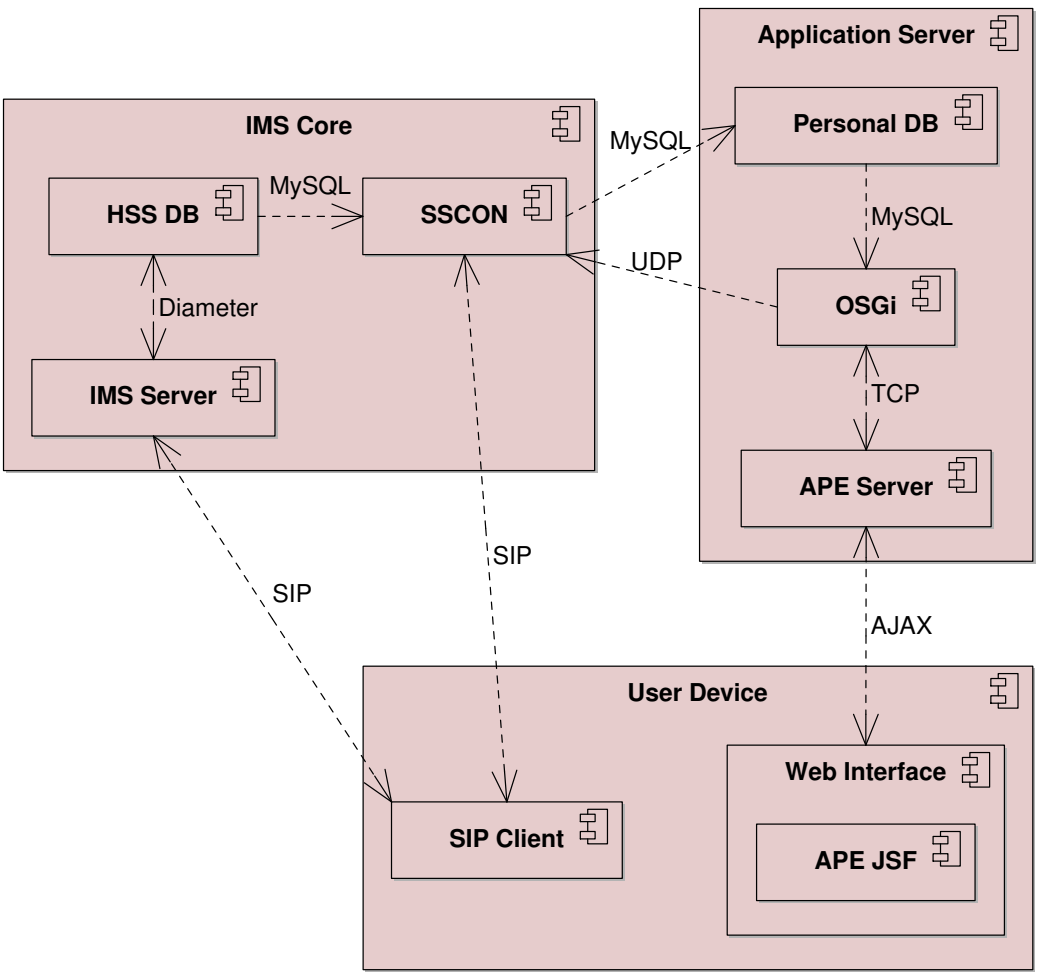


Figure 3.1: Component diagram for the new PNAI

- The **APE server** acts just as a proxy between the OSGi bundle and the pure JavaScript interface. Its solely function is to pass the AJAX requests to the TCP socket, and to translate any TCP transmission into AJAX responses. For the OSGi, it is as if it were talking to the Java applet, since the received messages follow the same pattern.

- The **APE JSF** is a JavaScript component that handles the push communication between the server and the browser, allowing the JavaScript codebase to not worry about that. The communication is asynchronous and in real time, so the web interface can do other things while waiting for the response.

  As it is used, basically it provides an object that emulates a TCP socket, with functions to send data and callbacks to receive data, that are transmitted in different Comet protocols (see § 2.8.1 on page 104). After a period of inactivity, the real socket is closed, so no resources are wasted.

- Since the Java applet is gone, that component diagram is valid for the desktop and mobile interfaces. The only difference in those web applications is, precisely, the code related with the interface.

Figure 3.2 on the next page shows the modified flow after these changes were done. All communications in the servers are exactly the same as in Figure 2.10 on page 30, so that right part is omitted for brevity. The scenario is also the same: the user wants to transfer a session from his device to one of his buddies.

Again, blue lines belong to the main logical flow, yellow ones belong to the video streaming process, and the new pink ones belong to the constant flow between the APE components. That connection between the APE server and the APE JSF is maintained through the whole session using the APE protocol.

After the initial handshake is performed, this protocol is composed by two types of messages, commands and raws: the first ones are requests from the browser and the second ones are responses from the server. In this case those messages simply wrap the data sent through the TCP socket.

There are two things to notice in this flow. First, the connection is always alive and ready; for that the browser must immediately make another

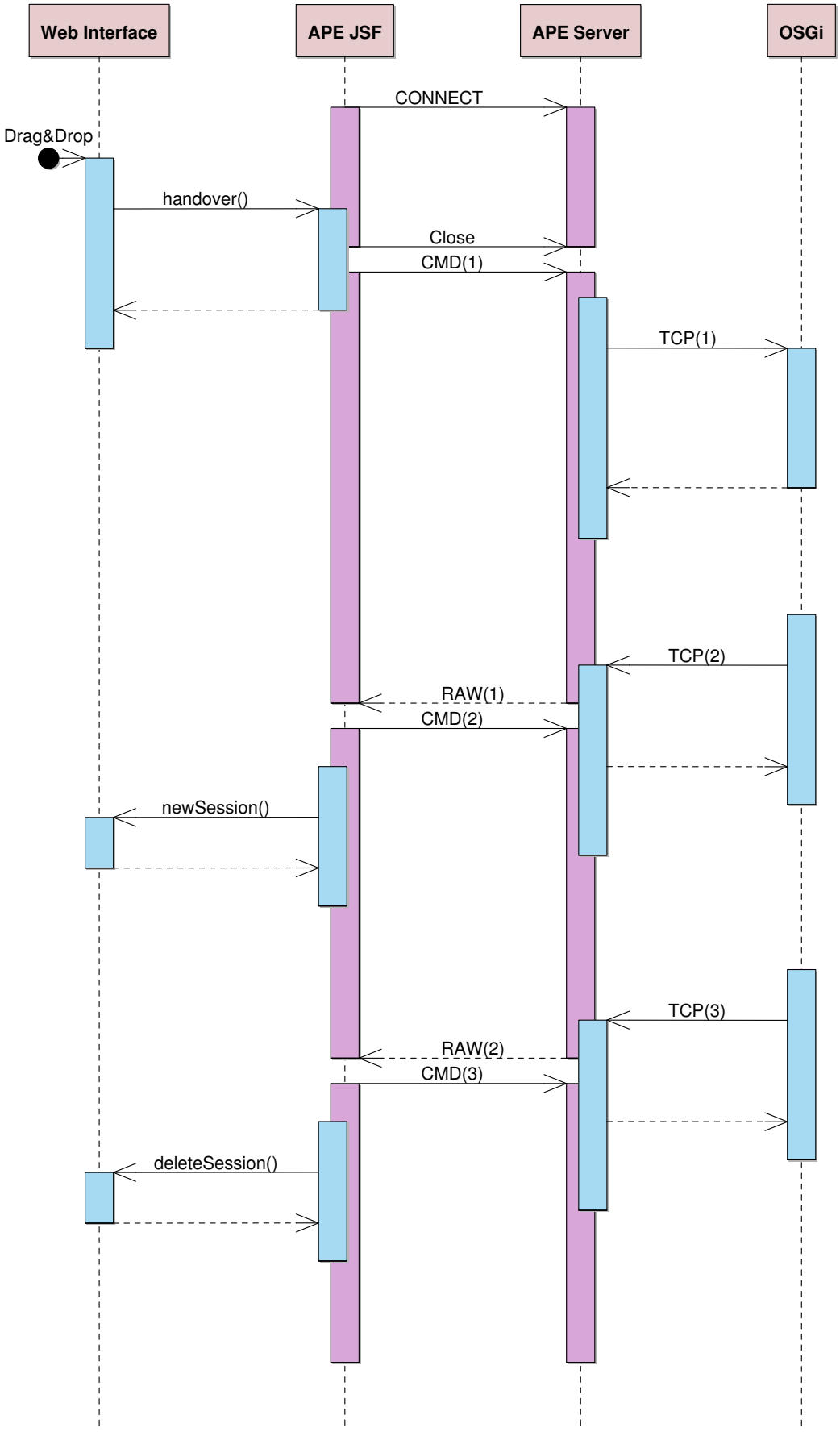


Figure 3.2: Sequence diagram for the new PNAI

request after the servers responds, normally that request is just to say "I am ready" (`CMD(2)` and `CMD(3)`). In other occasions, the browser has to send a request with data, so it must cancel the current request and make another one (`CMD(1)`).

After all, the addition of the APE server just affects the web interface written in JavaScript. Most of the code can be preserved and, with a new module written in JavaScript porting the Java applet functionality (see Figure 2.11 on page 40), the existing code only needs minor modifications.

3.5 Interface Design

The interface design was not a straightforward task, but rather an iterative process where several alternatives were weighted. Constant feedback from other members of the project helped shaping it until getting to the final design. At the end of this work two interfaces were fully designed to bring the same functionality under both desktop and mobile constraints.

3.5.1 Redesign

The previous design had several flaws that needed to be fixed. Also, since more functionality were implemented, additional abstractions needed to be designed. Considering these two facts, the main points behind this redesign were:

Organization The previous design was quite chaotic, the elements were just scattered and the visual impact was odd. By constraining the design to a grid, elements fall into place, so the user impression improves a lot.

Simplicity For example, the previous background was a little distracting, and the colors gave more importance to the background than to the foreground elements. By simplifying the shapes and desaturating the colors, now elements are more easily recognizable.

Also, buttons are easier to distinguish now, because they are bright orange while the rest of the interface is desaturated. All new images also follow this convention, so that they do not feel out of place. Another example is the new trash icon: it is desaturated by default, but when the user drags something onto that icon, it activates showing the full color version of that image.

Legibility Some texts in the previous design were very difficult to read: white over very light orange, for example. In this design this was a concern, and it results in a high contrast color palette: white over black, black over white. Other design tricks also help with the contrast, like adding shadows in some texts labels.

Smoothness Animations have been heavily used in the interface, to give the interface a responsive feeling to the user actions. For example, when the user drags the icon it slightly fades and when the mouse is over a valid container it turns opaque.

Also, when the icon is released onto an invalid container or when a session is deleted/duplicated/transferred, the icon smoothly *flies* from one point to another. These nice little touches are not just superfluous gimmicks, but they enhance the user experience.

Adaptability The previous interface was static, all elements had fixed sizes and positions. Now, the elements are designed to be resized automatically to fit the available space.

Customizability The user can now move and resize his devices, so there are new elements to make this task easier. When the mouse is over a device, two handles appear: one to move it and another one to resize it. The user just has to drag them to see how the devices change live.

Another visual element added for resizing the sidebar is another handle, acting as a cue to the user. The two panes of the interface resize accordingly with the position of that knob.

Intuitiveness For the user it makes more sense tagging the sessions with a still image from the video rather than with a generic icon. That is the most effective way to visually discriminate two different sessions.

Condensedness On the one hand simplicity was one of our goals, but on the other hand more information will be added to the sidebar. This has been solved by adding *tabs* to the sidebar so that the user can alternate between his buddy list and the new content list.

Also, in the new content list an *accordion* has been implemented to handle the different categories. So when content from one category is shown, the others are collapsed.

Figure 3.3 shows how the new interface looks like, in contrast to the old interface showed in Figure 2.6 on page 16 (that figure only contains the main frame). In that figure there are several obvious changes, new icons like the trash and buddies, new elements and the overall look. However most of the same abstractions are used, like drawing sessions inside of the devices.

Figure 3.3: New PNAI interface

But now there are two tabs; the previous figure showing the buddy list like the old interface. Figure 3.4 on page 154 exposes the second tab, the content list, that appears if the user clicks on the *Content* tab. This content list encloses all the information of the IPTVplus page (see Figure 2.14 on page 48) but without having to leave this page. Initially the content list appears completely collapsed (Figure 3.4(a) on page 154) and then the user

can click on any category and videos from that category will be shown (Figure 3.4(b) on the next page).

As explained before, the sidebar can be dynamically resized by dragging the sidebar knob, making it as bigger as half the page. The sidebar can also be collapsed by clicking on that knob or on the current tab, and the interface results like in Figure 3.5 on page 155.

Figure 3.6 on page 156 describes several steps in the transferring of a session. First the user drags the desired current session and drops it on another device. Then a menu appears and the user clicks on the first option. The interface notifies the backend so the video stops playing in the first container and starts playing in the second one. While that happens, the user sees a popup with information about the status of the request. When everything is finished, the popup disappears and the icon *flies* to the target device.

The process of buying new content can be triggered by clicking on any button that says *Buy* (the session starts playing on the default device) or by dragging the icon from the content list to the desired device/buddy. Since the buddy list is hidden, to select a buddy the user has to move the session to the *Buddy* tab and then the buddy list appears.

Figure 3.7 on page 157 contains some details of this process. First we can see how the user is dragging the session to the device, and how the dragged session icon has a different appearance depending on its position. The second detail is the popup window that appears to confirm the action.

Sessions are deleted the same way as before, by dragging them from the device/buddy to the container. Figure 3.8 on page 158 exhibits another detail when deleting a session. Notice that when the session icon reaches the trash, the trash icon recovers its true color.

When the mouse stops on top of a container, two handles appear. The one at the bottom left corner is for resizing that particular device. Figure 3.9 on page 159 shows how the device readapts to the required scale, resizing also the active sessions to the available space.

The other handle appears in the top left corner and it is for moving the device to another position. Figure 3.10 on page 160 shows how a device can be repositioned to anywhere in the left pane of the page. There is also an interesting detail, as those figures also reveal how the sessions are drawn

(a) Initial view, categories are collapsed

(b) User clicks on a category

Figure 3.4: New content sidebar

Figure 3.5: PNAI interface with the collapsed sidebar

in the buddy list.

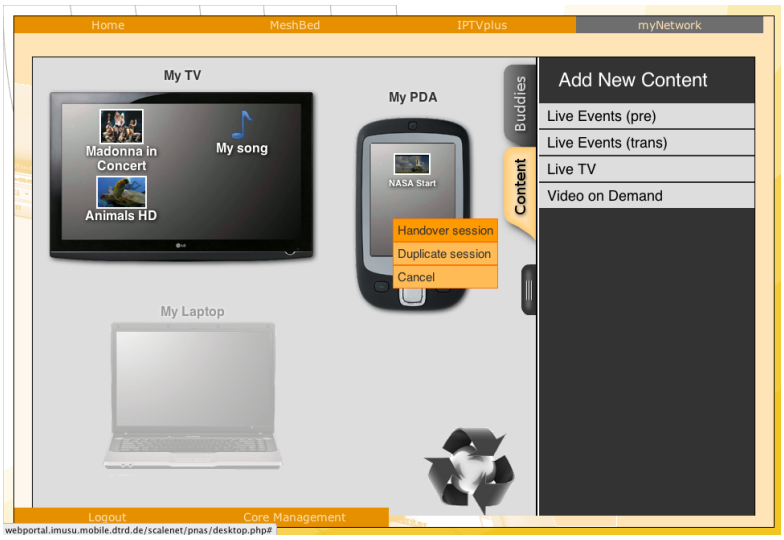
All these interface customizations are remembered the next time the user visits that page, using cookies to store that information. Also, if the user resizes the window, the position and size of the devices are recalculated on the fly.

3.5.2 Mobile Design

For the mobile version, the interface has been completely rethought. The design is based on the iOS guidelines for native applications: there is a title to inform the user where he is, there is also a tab bar and the main content is displayed in a list view.

Figure 3.11 on page 161 illustrates the login view and the three tab views. In some views, scrolling is needed to reveal all the content, so the user can drag the main view up or down with his finger while the title and the tab bar maintain their position in the screen. To achieve this effect, the iScroll library has been used.

The screen that welcomes the user is the login view. This is new respect the desktop version because the login was handled by other component,



(a) User drops the session onto another device, menu appears

(b) A popup shows some information, session starts moving

(c) Content continues playing on the other device

Figure 3.6: Transferring a session between two devices

(a) User starts dragging the icon, that is cloned with a dimmed appearance

(b) When the icon reaches a valid device, the icon turns opaque

(c) Popup window that ask for confirmation

Figure 3.7: Buying new content from the PNAI

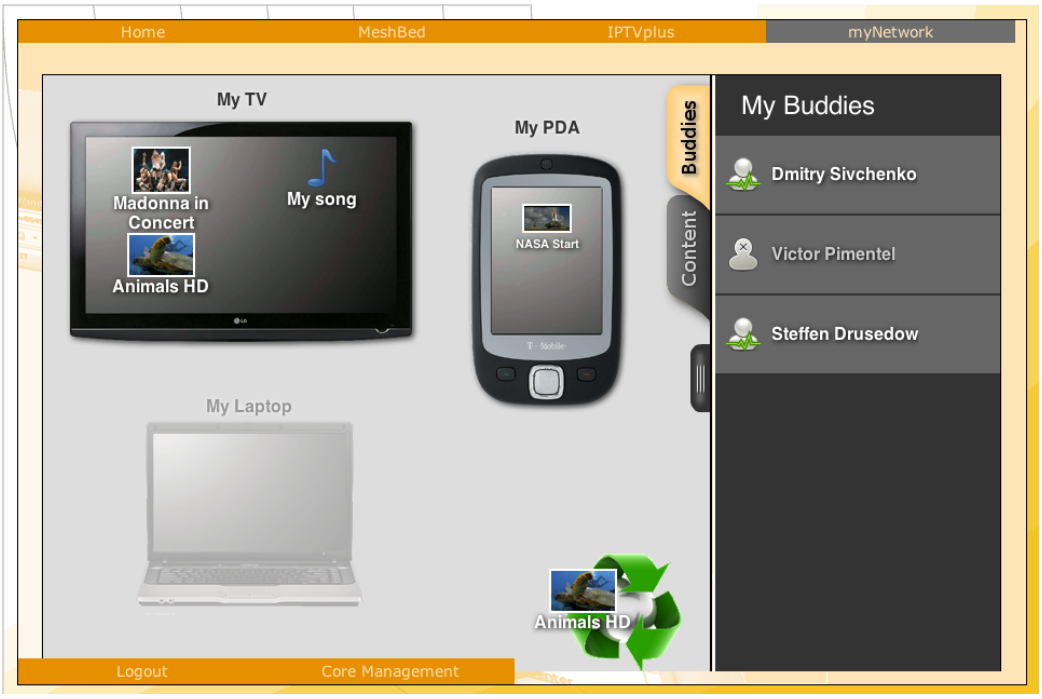


Figure 3.8: Deleting a session in the new PNAI

while in the mobile version there should be no frames and it should be redesign for mobile devices. It also explains why a button for logging the user out is added to the app's title.

The functionality is divided into three tab views: *devices*, *buddies* and *Add Content*. The first one is equivalent to the left view in the desktop version, while the other two are similar to the sidebar tabs. The user can reveal one of the tabs at a time, limiting the information in the screen comparing with the desktop version, something normal because the available space is smaller.

Once the user has logged in, the user sees his list of devices with the current sessions. The main difference from the desktop is that sessions are not shown inside of the device icon but at its side. To navigate between the tabs, a tab bar is always available at the bottom of the screen, working and looking exactly as any native application in the iOS platform.

To maintain the same feeling, the buddy list is designed in a similar way, but more compact. The content list follows another iOS guideline for displaying list, and it looks like a regular `UIListView`*, making the web

*`UIListView` is a component defined in the iOS SDK for creating lists.

(a) When the mouse stops over a device, the handle appears

(b) If the user drags that handle, the device resizes according to the position of the mouse

Figure 3.9: Scaling a device in the PNAI

(a) Initial position of the device

(b) User drags the device's title to the new position

Figure 3.10: Moving a device in the PNAI

(a) Initial login view

(b) Devices tab view

(c) Buddies tab view

(d) Add content tab view

Figure 3.11: New PNAI interface for mobile devices

application feel even more like a native application.

Regarding the implementation, except the icons, most of the interface is done strictly with CSS properties: all buttons, shadows, gradients and backgrounds are done with multiple CSS3 gradients. Even the font for the offline text uses the `@font-face` property.

It is important to remark that this interface is not a native application, it is the same web application running full screen in an iPhone. In this case, the user added a shortcut to the ScaleNet web page on the home screen, so it is shown as any other app (see Figure 3.12(a) on the next page).

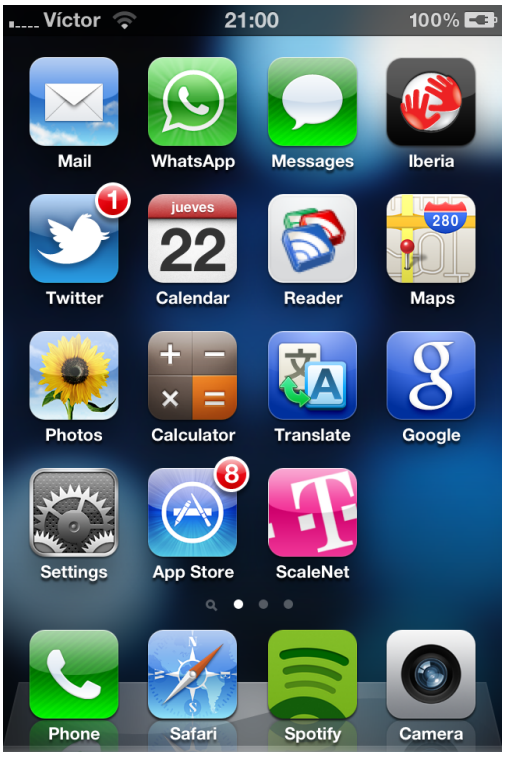
However the web app is flexible and when the web page is visited from the Safari browser it adapts to the available space (see Figure 3.12(c) on the facing page). Because of this, the app can also be used in landscape mode (see Figure 3.12(d) on the next page), triggered automatically when the user rotates the device.

When any app is launched, iOS has a mechanism that shows a static image until the app is loaded. According to the guidelines, this image should not be a splash image just for showing the logo of the app or company, but it should contain an empty interface with maybe a loading message. This *tricks* the user perception so that the app seems to load much faster, and since web applications are usually slower to load, it is a very welcome addition (see Figure 3.12(b) on the facing page.

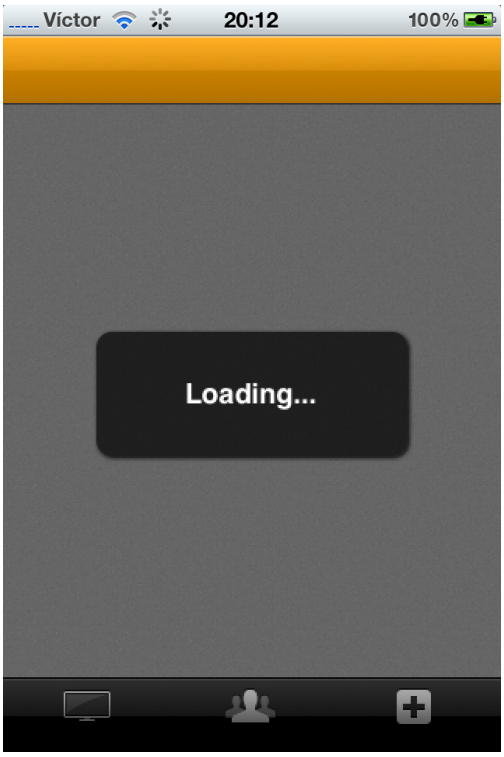
Because drag&drop is not viable, as it is used for scrolling, a different approach needed to be taken for the session actions. While the previous approach first selected the destination and then the action, now it is the opposite, first the action and then the destination.

The idea is simple: the user taps on the session he wants, then a menu appears with three options (transfer, duplicate, stop), he selects one of them and finally, if he chooses transfer or duplicate, he taps on the destination device or buddy. This approach was explained in detail in use cases 19 to 24. Figure 3.13 on page 164 shows how the process looks like in this new interface.

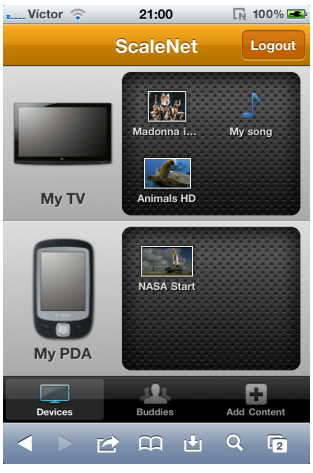
Overall, the main point in the design was to follow the iOS guidelines for native apps, so the menu is pixel-exact with the built-in menu that appears to copy, cut and paste text. The behavior is also copied: when the user taps outside of the menu, the action is cancelled.



(a) The pink icon launches ScaleNet



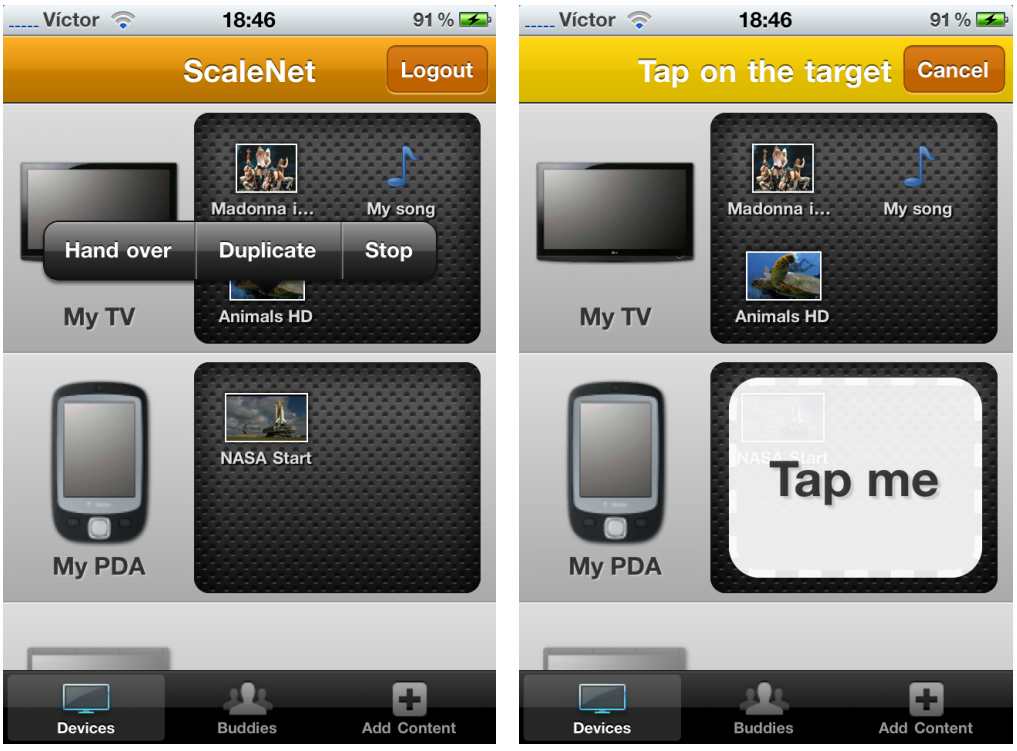
(b) The startup image in action



(c) The interface adapts to the Safari browser

(d) When the device is rotated, it automatically resizes to a landscape view

Figure 3.12: Mobile PNAI interface in different situations



(a) A menu appears when the user taps over the session

(b) Then the target selection mode activates, title slowly "pulses" into yellow

(c) User selects a device, the interface updates with information

(d) The session is transferred

Figure 3.13: Transferring a session in the mobile PNAI interface

In the target selection mode, the title background displays a yellow pulse effect to get the user attention, similarly to the title that appears when there is a phone call in progress. Other elements, like the information panel or the *Tap me* area, were designed trying to keep it simple but elegant.

The purchase process of new content is slightly different, since there is a specific tab for it (see Figure 3.14 on the following page). In the content list there are image stills, some information and buttons to buy those contents.

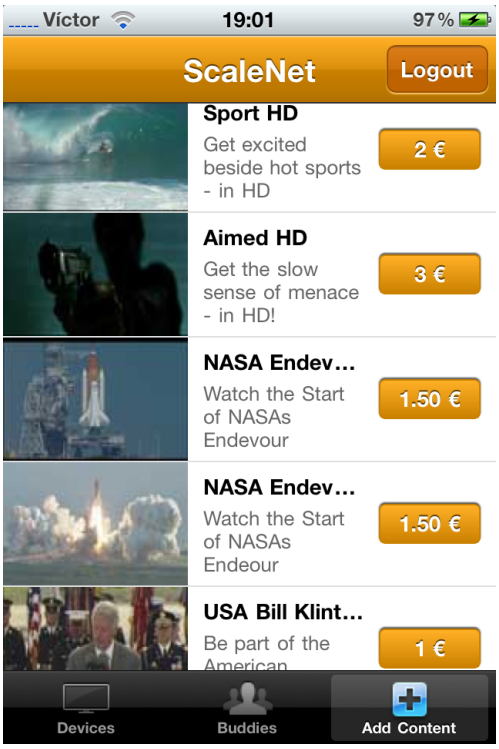
To actually buy anything the user has to tap twice the button, since it is very easy to accidentally tap on it. This behavior is the same as in the AppStore or the iTunes app, so the user should be already familiarized with it.

Once the desired content has been chosen, the application automatically changes to the device tab so the user can select the destination. The rest of the process is the same as in a transfer or duplication. In any action, the user can also select a buddy as the target, he only has to change to the buddies tab and then he can select any online buddy.

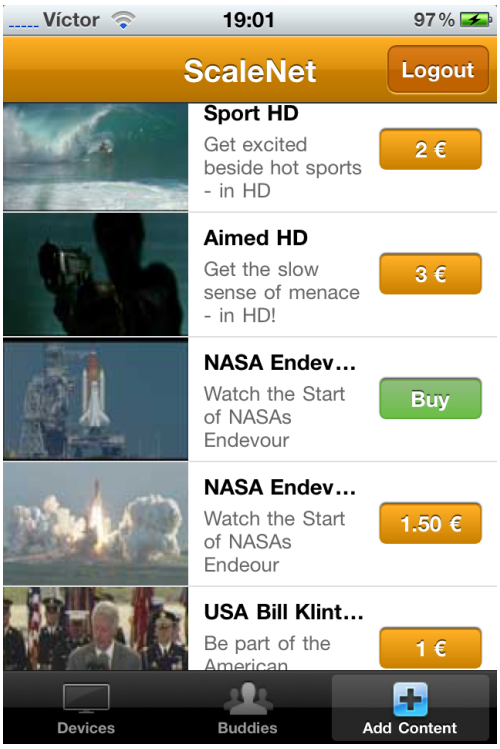
3.6 Architectural Design

The biggest chunk of work – measured in lines of code – comes from the JavaScript codebase. All code from the Java applet was ported to JavaScript, adding more classes to the front-end code. Most of the existing code was rewritten to seize specific features of MooTools, and other classes needed to be added for the new elements.

At the same time, some code from the OSGi bundle shifts to PHP files, including all statics files needed to run the web application. But the OSGi bundle keeps being in charge of the TCP socket, so it is as necessary as before. Thankfully, the bundle does not need to be update and any build should work in both versions of the web application – with and without the applet.

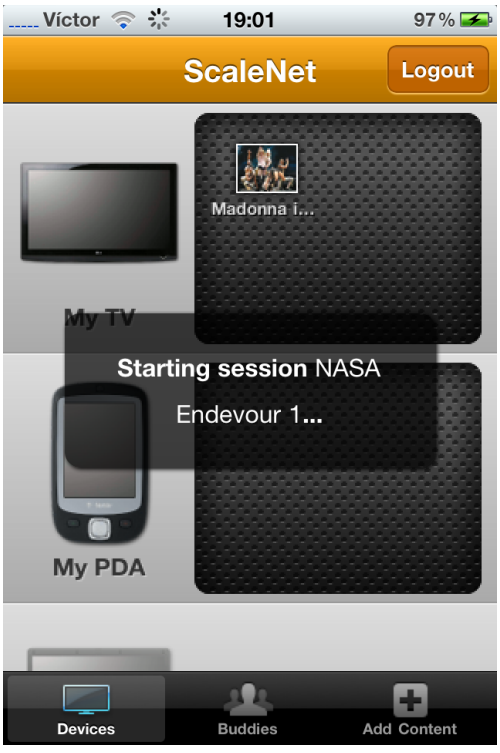


(a) The user wants to buy a session



(b) The first tap reveals the button, the second tap buys the content

(c) Target selection mode, only online devices and buddies are selectable



(d) The progress is notified

Figure 3.14: Buying content in the mobile PNAI interface

3.6.1 JavaScript Codebase

The basic design of the classes is pretty much the same as before, with all important elements of the interface having their respective classes to handle both the drawing and functionality behind that particular element.

All classes are implemented as MooTools classes, to embrace its utilities and handy abstractions. One of the things that are possible is to easily gain functionality from more than one class, by *implementing* that MooTools class (in reality it is more like extending).

Another pitfall was that the browser compatibility had to be very broad, and the codebase should remain the same for every browser. The MooTools library homogenized most of peculiarities, nevertheless some very specific hacks were needed for IE. This became most handy in animations, but any DOM homogenization is welcomed.

To avoid duplicity, it was decided that the same JavaScript logic would handle the two interfaces. Those designs are not just visually different, but they adopt very contrasting abstractions for interface elements and user actions.

The chosen solution was to dynamically load only the needed JavaScript files depending on the platform. It also results on two methods to create each one of the interfaces and diverted paths in the execution flow. But that applies in few situations, and most of the code the same – not similar but physically the same.

The Java code for the applet was relatively straight to port to JavaScript. As figure 3.15 on the next page shows, the model classes are very much alike, since they mostly contain data and methods to transform that data into a string that can be transmitted back to the server.

The main difference is how the models are instantiated. Before, the factory pattern was used to create objects from a string, but now that approach would have been more difficult to implement in JavaScript and specially with MooTools classes. Now, the only way to create an object is directly *instantiating* that object, even if no option is provided. Later, the `fromString` functions can be used to parse any string and fill its options.

The other difference is related with the way the parameters of a class are given, something used in almost all classes. This `Options` interface – provided by the MooTools framework – is a very powerful approach that

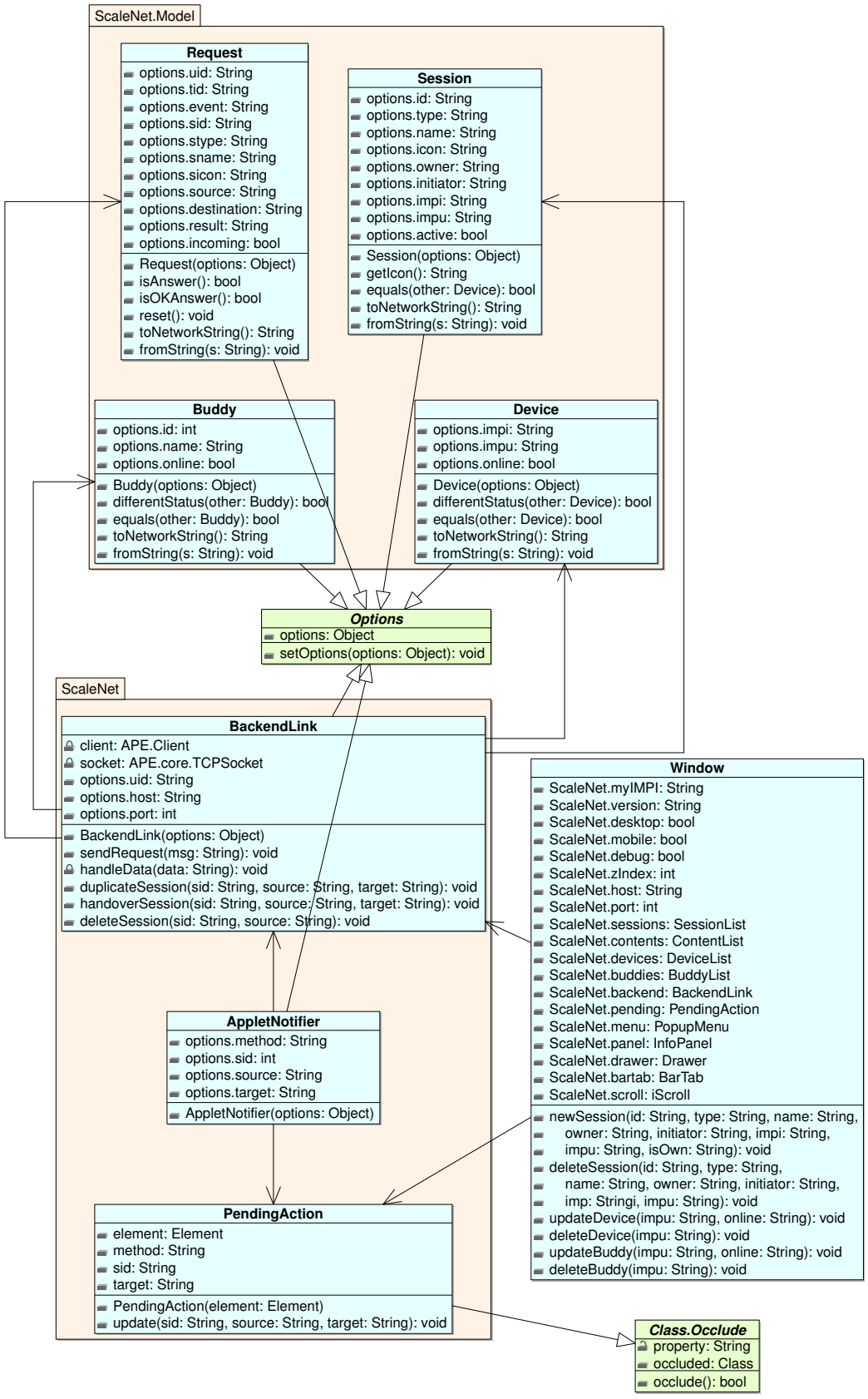


Figure 3.15: Class diagram for the new PNAI: Model and BackendLink

allows the setting of any number of optional parameters in a class with only one line, without having to explicitly treat every value in the constructor.

About the `BackendLink`, obviously the `ScalenetApplet` class is gone, since that was the main class of the applet and now there is no applet. Most of its functionality has been added to the `BackendLink` class, so now it is in charge of all the communications with the backend. This includes sending information to the TCP socket and handling the data received in that socket, rerouting the actions as necessary.

One of the goals was to be able to change the APE stuff with the applet by modifying as less code as possible. A new, very simple class called `AppletNotifier` now wraps all the code needed to use the `BackendLink`. By changing only a line of code there, the applet – or any other new compatible system – could be used again instead. Basically it acts as a bridge between the logic of the application and the backend communications.

To maintain that possibility, some functions in the global object had to be kept because the applet directly calls them. By design, no function should be declared in the global object, so they have been reduced to the minimum. For obvious reasons, these functions have to keep the same parameters as before.

Apart from the backend link, the `PendingAction` class keeps storing the data about the pending action, but now it is a little different. In the application there is only one object of this class instantiated, `ScaleNet.pending`, and that object is updated when necessary. MooTools offers `Class.Occlude`, a class that makes this kind of singleton pattern very easy, by linking a MooTools object to any DOM element.

This brings us to one of the design decisions: to store all singleton objects in one place available from everywhere instead of using constructors to retrieve the cached objects. This is to avoid some overhead and be faster in this very basic task. As polluting the global object is seldom a good idea, there is a new object created in the global scope: `ScaleNet`.

In the `ScaleNet` object there are also defined all MooTools classes, acting similarly to a Java package in that the functionality is more compartmentalized. Finally, that object contains some parameters to ease the development and customizability:

ScaleNet.myIMPI The user id, set by PHP directly in the page.

ScaleNet.version The version of the code (1.0), useful for future reference.

ScaleNet.desktop/ScaleNet.mobile Booleans to indicate whether we are in the desktop version or the mobile one. Very useful to discriminate chunks of code depending on the context.

ScaleNet.debug Boolean to activate debugging logs.

ScaleNet.zIndex Needed for handling the position of the devices so that the newest ones appear on top of the oldest ones.

ScaleNet.host/ScaleNet.port Host and port of the socket in the backend.

Figure 3.16 on the facing page shows all code involving containers. To deal with dimensions the new `ContainerDimensions` class has been extracted from `Container`, to homogenize and ease the work with positions and sizes across the application. Also, it encapsulates all the logic needed to transform dimensions between pixels and percentages.

The `Container` class has received some improvements to implement the new functionality. This includes the addition of functions to make the DOM element both draggable and resizable. These functions use some MooTools classes for the hard work, but the dimensions are treated there to avoid IE glitches.

The device dimensions are defined in percentages, to be able to place the elements filling the page even if the user reopen the web application with a different window size. IE does not like that, and the animations only works smoothly if the dimensions are specified in absolute units, i.e, pixels. To avoid that bug, those functions automatically transform percentages to pixels at the beginning of the resizing/repositioning, and they change it back to percentages when the action is done.

The `Container` class also demonstrates how the same code can handle two interfaces. Since the created elements are very different, two functions are added to create the specific version – `createHTMLContainer` for desktop and `createMobileHTMLContainer` for mobile. Each of them uses MooTools to dynamically create and add the elements to the DOM. When a container is created, the desktop/mobile flags can be checked so the correct method is called.

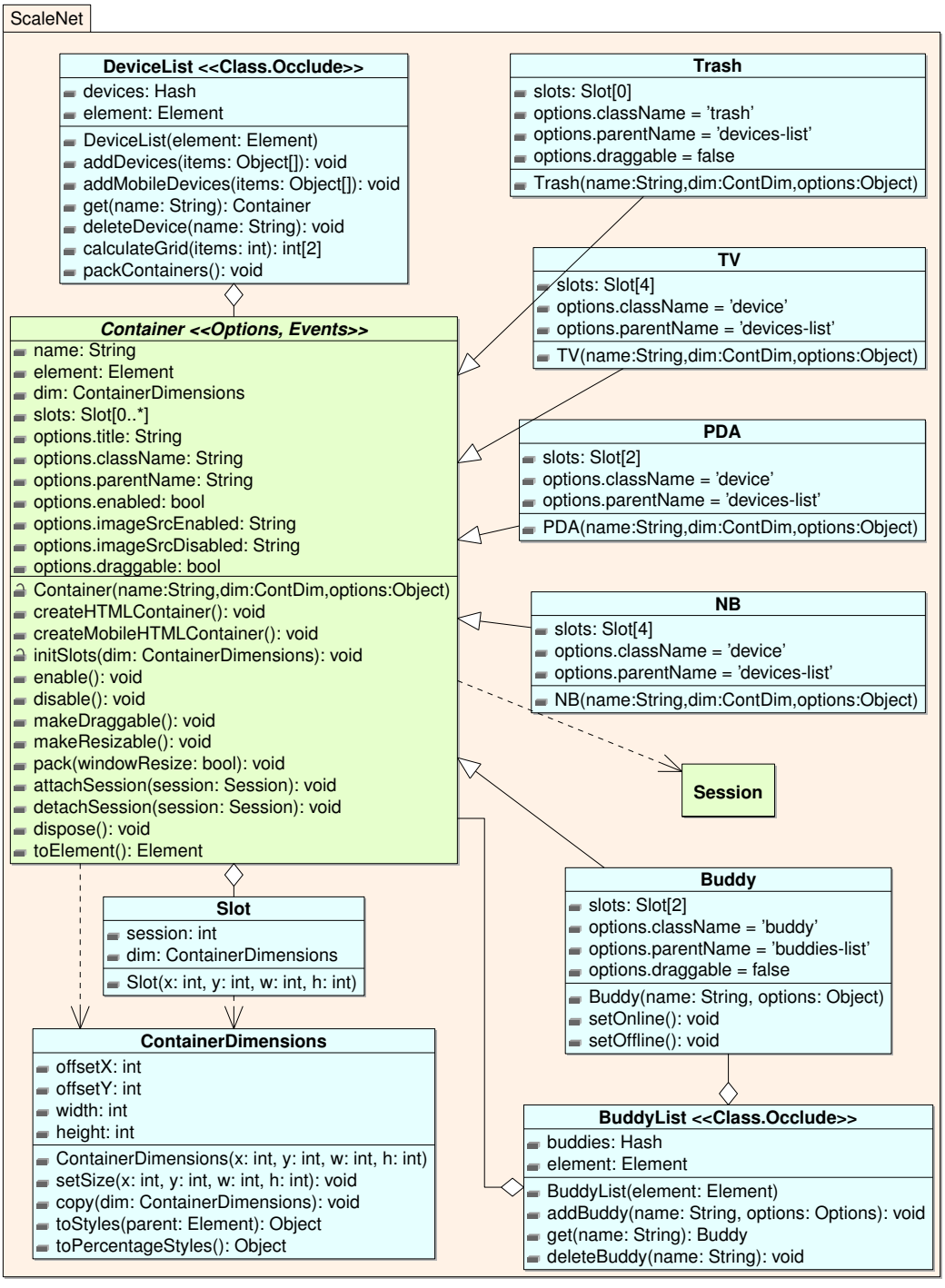


Figure 3.16: Class diagram for the new PNAI: Devices and Buddies

The last new method is called `pack` and its goal is to resize the container to the available space. This is needed for the desktop version because devices have to maintain the same aspect ratio, and because their sessions are drawn on top of the container – in another element acting like a mask. When the window size changes or when the element is resized, that mask needs to be repositioned and resized to the updated size.

Apart from that, the class implements the Events class, another MooTools addition that comes very handy to inject additional code in prefixed places in the class. Basically, functions can be registered for some custom *events*, in the code of the class the `fireEvent` function is called when appropriate, so when that code is executed all registered functions are called. For example, the `enable` method fires the `enabled` event, so any other class can inject their code every time the `enable` method is executed.

The `DeviceList` class has been added to offer some functionality similar to the existing `BuddyList` class. The most important method is `addDevices`, a function that populates the device list with the specified devices, ordering them so that they fill all available space (see § 3.7 on page 181). Another function is provided to *pack* every device at once, useful when resizing the window or the sidebar. The rest is just the usual structure for handling a list.

`BuddyList` and `Buddy` are not very interesting, they just have been updated to MooTools classes, but its functionality is mostly the same. If something, the code has been simplified in favor of moving as much code as possible to the `Container` class, to avoid duplicating code.

Figure 3.17 on the facing page comprises all classes around sessions and content items in general. In the `Session` class, the `moveTo` method continues to be very important, since the visual representation of a session is not created until the session is moved to a container. It is also in charge of moving sessions from a container to another, creating a nice animation using MooTools.

A `Session` is also draggable in the desktop version, so a similar method to the one in the `Container` class is implemented. Unlike containers, sessions should be prepared to be dropped, so the logic is a bit more complex. The `dropInto` method is called when the session is dropped into another valid container, and depending on the container it either requests

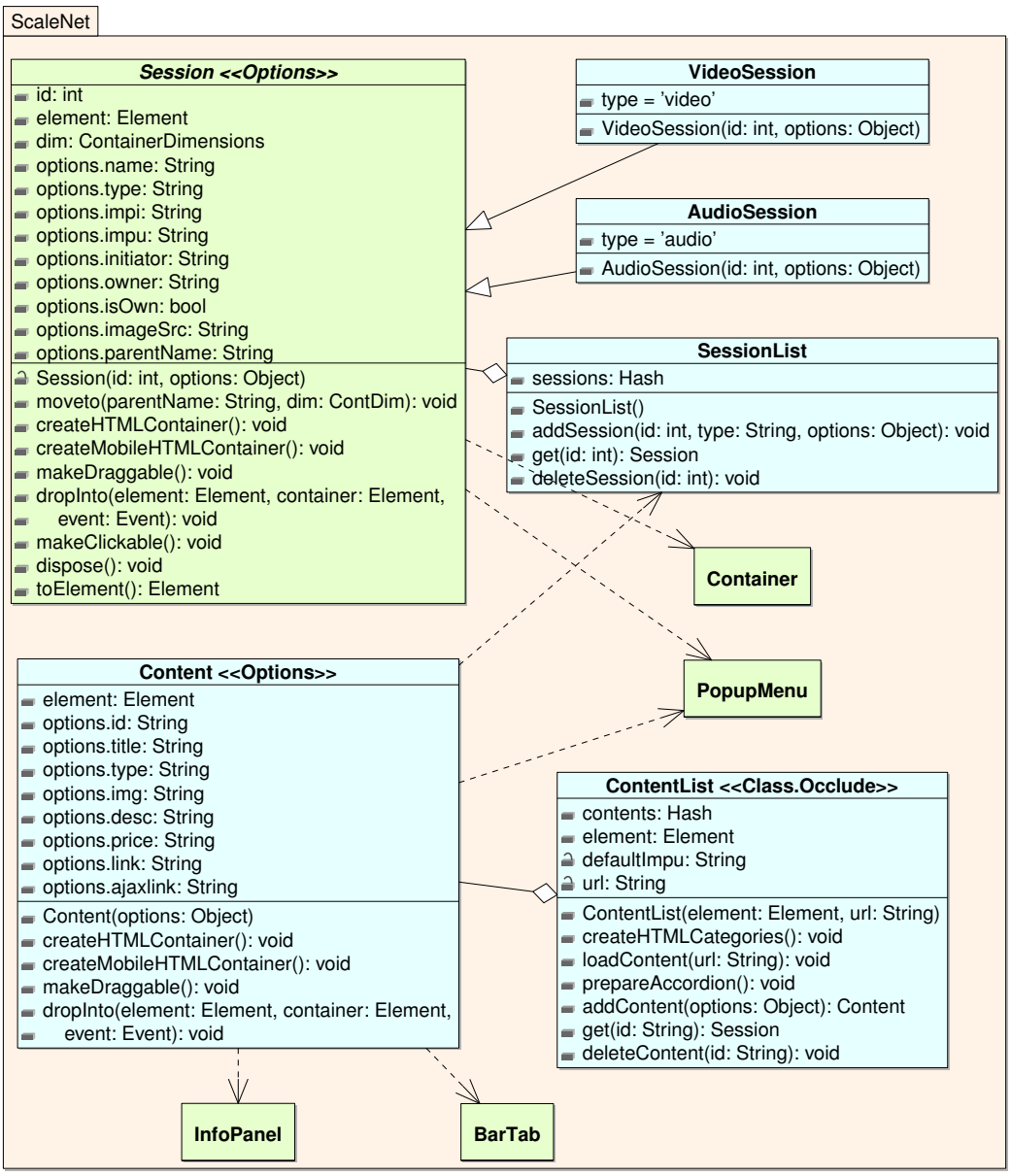


Figure 3.17: Class diagram for the new PNAI: Sessions and Content

the deletion of the session or it shows the popup menu.

The mobile version is much more straightforward, as the `makeClickable` method just has to show the popup menu when the element is clicked. `SessionList` is another addition similar to other lists, and it just collects all the known sessions.

The `Content` and `ContentList` classes are completely new, as the functionality was simply not there. `Content` stores all information about a particular piece of content, and also includes the needed code to handle purchases of that content.

In the desktop there are two approaches: to directly buy the piece by clicking the *Buy* button, or by dragging the icon to a specific container. For the second approach the system is very similar to dragging sessions, with a method to add the dragging functionality and another to handle a successful action. In both situations, a confirmation popup will appear – the `options.link` sets the address to open. One important difference is that, when the user drags the icon to the buddies tab, that tab gets revealed.

In the mobile version the `createMobileHTMLContainer` method handles the whole *tap twice to buy the content* experience, and the posterior selection of a target. Because of that, this method is somewhat complex, but since it does not need any additional confirmation, it can directly tell the backend that the user wants to buy that content – for that the address contacted in the AJAX call is the one stored in the `options.ajaxlink` property.

There is one last interesting detail about this class: when a new piece of content is created, it traverses the current session list and updates the session icon with the thumb icon for that content. This means that, when the content is loaded, the current sessions get their characteristic icons, even if that information was not provided by the OSGi bundle through the socket. However, if there is a problem and the content is not loaded, the current sessions will not have custom icons.

The `ContentList` class is in charge of not only collecting and categorizing the pieces of content, but also of preparing the interface for that content list. The `loadContent` method is the most important one, since it dynamically retrieves that list from the server after the page is loaded. The information is obtained with a JSONP request to a PHP page.

Figure 3.18 on page 176 gathers the rest of the classes in the JavaScript

codebase, mostly related with interface elements and their behavior. The `PopupMenu` class is in charge of showing the popup menu to the user with the available options – transfer, duplicate and stop sessions –, and react to the user selection. In that occasions, the `AppletNotifier` is created on demand. In the mobile version, this class also is in charge of the select destination mode.

The `InfoPanel` class is responsible of showing the informational panel to provide feedback to the user. There are two different types of messages: normal messages and errors; each of them have different icons to alert the user depending on the result. The interesting bit about this class is that it allows the chaining of messages, so that every message is shown to the user at least for a couple of seconds even if they are triggered at the same time.

In the desktop version a third party MooTools class is used, called `Growl`* because it emulates the looks of a notification system for Mac. Though some little fixes had to be added to make it compatible with the latest version of MooTools, it automatically handles the chaining of messages.

However, in the mobile version another approach was needed, because the previous class was designed for desktop browsers but not for mobile browsers. It was more efficient to do it by hand, therefore a very basic notification system was implemented allowing also the chaining of messages with a simple but elegant look.

The `Drawer` class is specific to the desktop interface, and it manages all aspects regarding the sidebar, such as tabbed navigation. It also deals with its resizing and visibility, something that needed a lot of attention to fix quirky bugs in IE.

Its mobile counterpart is the `BarTab` class, that administers the navigation system based on the bottom tabs and nearly everything related in the mobile interface, like orientation changes. To successfully implement the tab bar, it uses iScroll, another JavaScript component designed for Webkit mobile browsers (see § 2.9 on page 109).

*http://icebeat.bitacoras.com/mootools/growl/

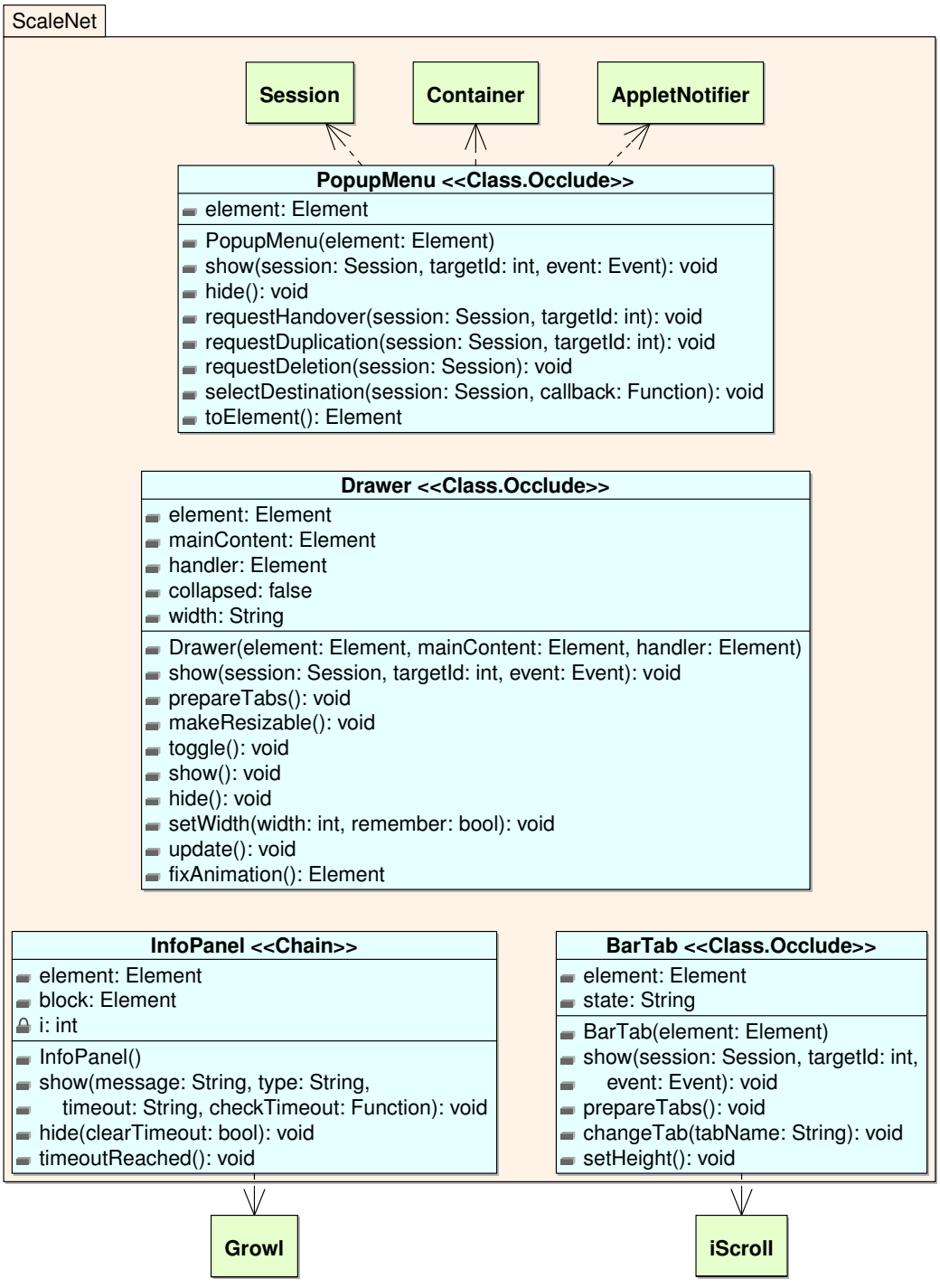


Figure 3.18: Class diagram for the new PNAI: Interface elements

3.6.2 PHP Codebase

In the server all modified code is implemented in PHP scripts, sharing the same tree as the IPTVplus interface. Figure 3.19 on the next page shows all the new files in bold, in contrast to § 2.2.4 on page 47 The main page remains mostly the same, just updating some styles to modernize and fix some bugs without changing the overall style.

In the desktop, the user starts visiting the `index.php` (see Figure 3.20 on page 179), where he can choose between the old IPTVplus application, the PNAI (at the right) and another different application, the MeshBed. Once the user clicks on the *myNetwork* link, the `personal.php` page presents a login form (see Figure 3.21 on page 179).

If the user logs in successfully, that page will change to include the `session.php` page. In that page a frame is defined: previously it linked to the server built by the OSGi bundle, but now it just links to the `desktop.php` page in the `pnai` directory. The whole picture looks like Figure 3.3 on page 152.

However, if the user try to visit the front page from an iOS or Android device, he will be redirected to the `mobile.php` page in the `pnai` directory. If the user is not logged in, the `mobile-auth.inc.php` displays the login form (see Figure 3.11(a) on page 161), but if he is already logged in, his devices are directly shown (see Figure 3.11(b) on page 161).

That way, all code related to the new PNAI is all isolated in the `pnai` directory. Both `desktop.php` and `mobile.php` are the respective main pages for the two interfaces, and they define only some basic elements needed in the page structure. In the `css` directory all the styles are stored in two different files – one for each file. Also, that directory contains the web fonts used for some effects. Finishing with the assets, the `images` directory stores all the needed art files for the interface.

Finally, the `js` directory is the home for all JavaScript code. Each file includes one or more MooTools classes organized by topic. Most of them are self explanatory, except `utils.js` that defines the `InfoPanel` and `PopupMenu` classes.

The desktop version includes all files except `bartab.js` – that contains only that class –, `iscroll` – the third party library –, and `main-mobile.js` – the main function that prepares all. The mobile version includes all files

```
scalenet/
├── index.php
└── sub/
    ├── includes/
    │   ├── c2d-ajax.php
    │   ├── c2d.php
    │   ├── inhalt.php
    │   └── popup.php
    ├── IPTVplus.php
    ├── IPTVplusJSON.php
    ├── personal/
    │   └── sessions.php
    ├── personal.php
    └── pnai/
        ├── cache.manifest
        ├── css/
        ├── desktop.php
        ├── images/
        ├── index.php
        ├── js/
        │   ├── ape-jsf/
        │   ├── appletglue.js
        │   ├── backendlink.js
        │   ├── bartab.js
        │   ├── buddies.js
        │   ├── containers.js
        │   ├── content.js
        │   ├── iscroll/
        │   ├── main-mobile.js
        │   ├── main.js
        │   ├── model.js
        │   ├── mootools-1.2.4-core-yc.js
        │   ├── mootools-1.2.4.4-more.js
        │   ├── notifications.js
        │   ├── sessions.js
        │   ├── sidebar.js
        │   └── util.js
        ├── mobile-auth.inc.php
        └── mobile.php
```

Figure 3.19: New PHP directory

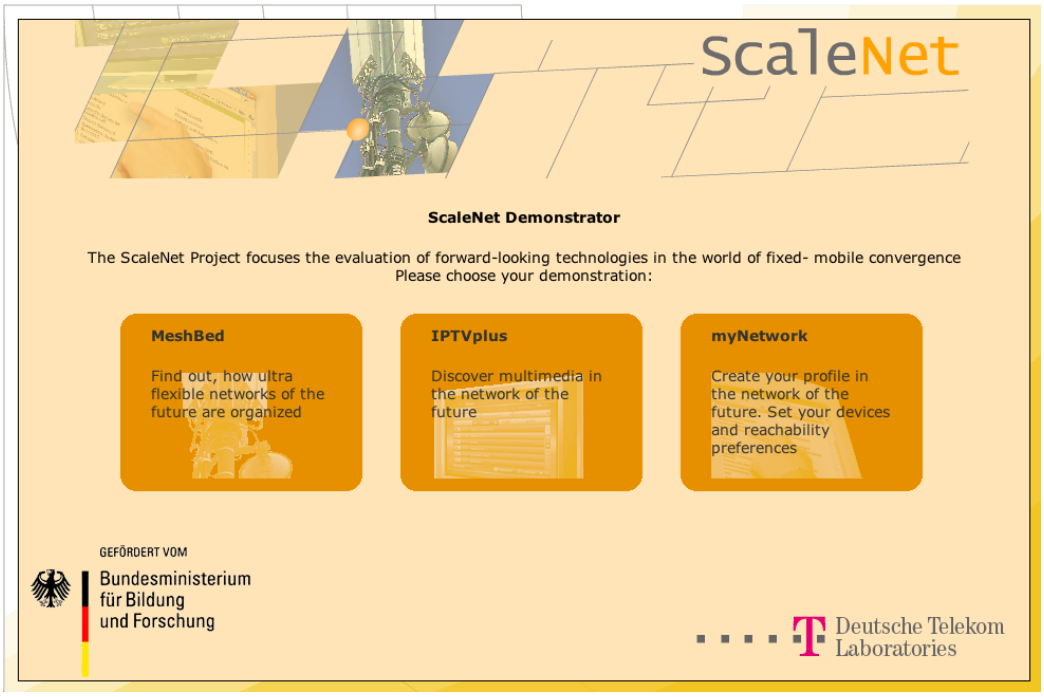


Figure 3.20: ScaleNet front page

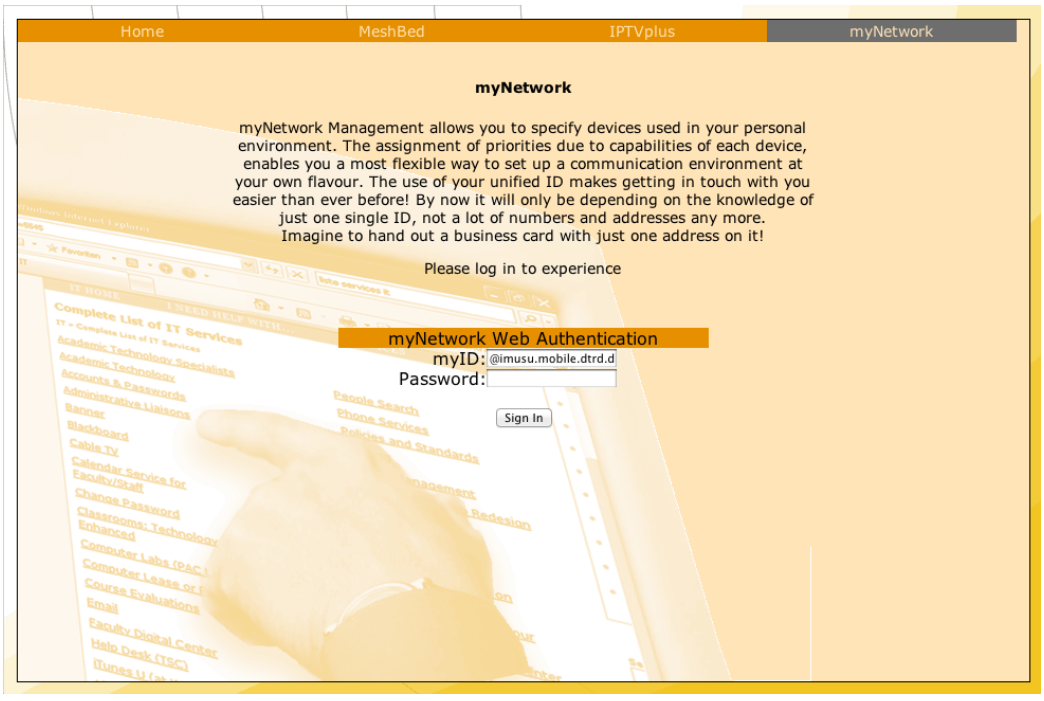


Figure 3.21: ScaleNet login page

except `sidebar.js` – the `Drawer` class –, `notifications.js` – the Growl library –, and `main.js` – the desktop version of that main function.

The last file to explain, and probably the less important, is cache.manifest. This file is used to help the mobile browser caching the application static assets – JavaScript, CSS and images. Its content is basically a plain list of all static resources.

Going up to the `sub` directory, there are two new PHP scripts. Both are designed as the AJAX versions of other two scripts, so their output is formatted in JSON/JSONP instead of HTML. `IPTVplusJSON.php` returns the list of contents available to buy and `c2d-ajax.php` receives purchase requests so the content can start streaming.

Listing 3.1: JSON content list example

```
myCallback({
  "error": false,
  "content": [
    {
      "id": "MYID",
      "img": "/scalenet/sub/iptvplus/images/...jpg",
      "title": "Content title",
      "type": "tv",
      "priority": "everybody",
      "price": "3",
      "desc": "Content description",
      "quality": "100\/300\/...2000\/4000",
      "link":
        ↪"popup.php?price=3&link=c2d.php?%3Ftype%3Dtv%26...",
      "ajaxlink":
        ↪"c2d-ajax.php?type=tv&quality=...&content=MYID&impu="
    },
    {...}
  ]
});
```

Both connects to the IMS server through a socket, then they send a command, the IMS responds and then they format the output accordingly.

`IPTVplusJSON.php` sends the `list` command, it receives the content list in a string where every new line is a new content, and it formats it like in Listing 3.1 on the facing page. It is no surprise that it contains exactly the same information as in the `Content` MooTools class.

The response, a JavaScript object, is wrapped into a function call which name is defined by passing a parameter to the script. One of the properties of that object is `error`, that contains a message for the user if the content list could not be retrieved. The main property is `content`, an array made by objects with the required data for each piece of content.

As explained before, the `link` is used by the desktop version and the `ajaxlink` by the mobile version. Each of one is already prepared to be called with all parameters except the destination `impu`. Consequently, the `c2d-ajax.php` script receives that request with those parameters, then it opens the socket and sends a `refer` command. The response to the browser is the `error` message in case of a problem or simply an object with the content `{result: 'OK'}` in case of success.

3.7 Additional Implementation Details

In the PNAI reimplementation there are some details worthy of further discussion. In this section the managing of devices in the desktop is explained. Before there were always three devices in static positions, but now the system should be prepared to having an undefined number of devices.

3.7.1 Positioning Devices

The first time that the user visits the page in a new browser, the system has to place the devices in the screen. Since the number and type of devices – therefore its size – are different for each user, an algorithm must be calculated to placed the devices in the available space. There are some restrictions that the algorithm have to comply with:

- There is an undetermined number of elements to place.

- For the sake of simplicity, the elements and the canvas are rectangular.
- Every element, including the canvas, has a different size.
- The elements have to be placed in the most comfortable way possible for the user, ideally using all the available space.

The solution to the above problem is known as a variant of 2-D rectangle packing and it is, regrettably, NP-hard. At this point, it is clear that we need a simplification. Furthermore, the original problem also presents a big issue, as it does not address an important constraint: the possibility of resizing the elements to fit the canvas.

A simplified algorithm is proposed and implemented, relaxing some terms while obtaining acceptable results. The important concepts are:

- The main goal is to draw a virtual grid of M×N cells (like a table). Every cell can be a position for a device.
- Indeed, we are going to calculate the smallest grid of M×N cells in which the devices can be placed.
- Finally, we are going to resize the elements to fit within that cell, giving that every cell has the same size.

To obtain the squared grid for N elements, we simply have to calculate the ceiling of the squared root of N as seen in eq. (3.1).

$$Grid_x = \lceil \sqrt{N} \rceil \tag{3.1}$$

This formula synthesizes the idea that, given a certain number of elements N:

- If N is a square number (that is, it exists an integer x that fulfills $x^2 = N$), then you could fit a whole grid of $x \times x$ with N elements.
- Otherwise, this integer N must be between the square of two consecutive integers, that we are going to call x and $x + 1$. That is, $x^2 < N$ and $(x + 1)^2 > N$. In visual words, that means that a grid of size x cannot hold that number of elements, and a grid of size $x + 1$ can hold that number of elements but there will be empty *cells*.

- In that case, we choose the grid of size $x+1$, this way we want to apply the ceiling function to the squared root to obtain the next following integer.

Figure 3.22 on the next page shows consecutive examples using from one element to thirteen elements, so that we can appreciate what all of this means. Without looking very hard at the examples, we can guess an improvement without making the calculation severely complicate.

It is easy to see that, at certain points, a whole row of the grid is completely empty, so we are wasting vertical space. In theory, we could detect when this happens and try to reduce the vertical height of the grid by one, effectively converting this squared grid into a rectangle grid with different number of columns and rows.

Parting from an example: if we have $N = 19$, then we obtain $x = 5$ by applying the previous formula. This will lead us to a 5x5 grid, but the last row will be completely empty. The question that we have to ask ourselves is: how big has to be the rectangle grid in order to be able to place this number of elements?

Briefly, the answer is $x \cdot (x-1)$. That is a very simple mathematical way of describing that we are decreasing the number of rows by one. In this case, we need a grid of 5×4 elements: that will hold up to 20 elements.

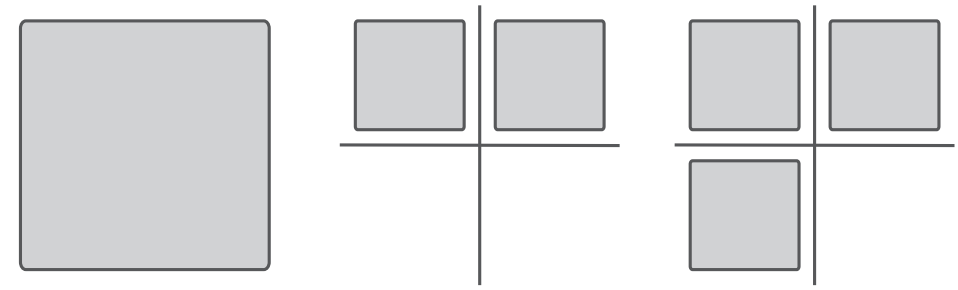
To discover whether we have to decrease the number of columns or not, we must compare that number $x \cdot (x-1)$ with the actual count of elements. If we know that this number is bigger or equals to the number of elements, then we know that a grid of $x \cdot (x-1)$ elements can hold those elements.

On the contrary, if we know that this number is strictly lower than the number of elements, then we know that a grid of $x \cdot x$ is unavoidable. This can be formulated as in eq. (3.2).

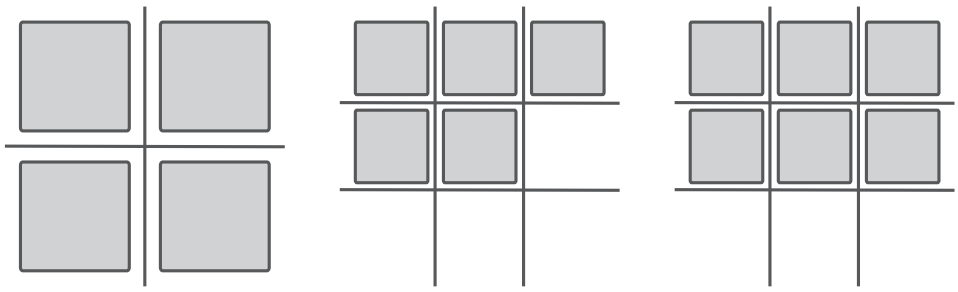
$$Grid_y = \begin{cases} Grid_x - 1 & \text{if } N \leq Grid_x \cdot (Grid_x - 1), \\ Grid_x & \text{if } N > Grid_x \cdot (Grid_x - 1). \end{cases} \tag{3.2}$$

Another question appears: Do we need to shrink the grid only by one? Is there any case in which we have to shrink the grid by two or more?

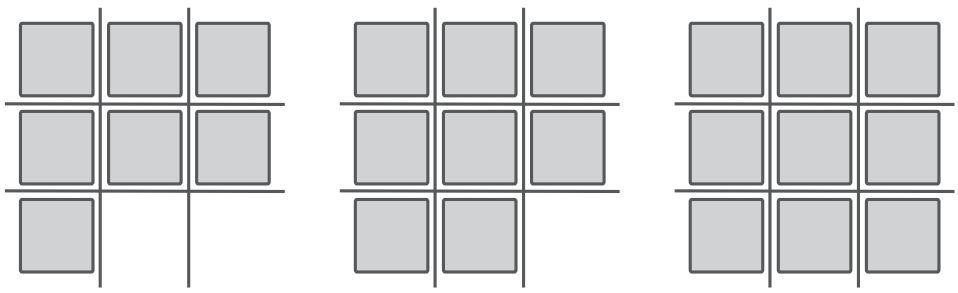
The answer is no.



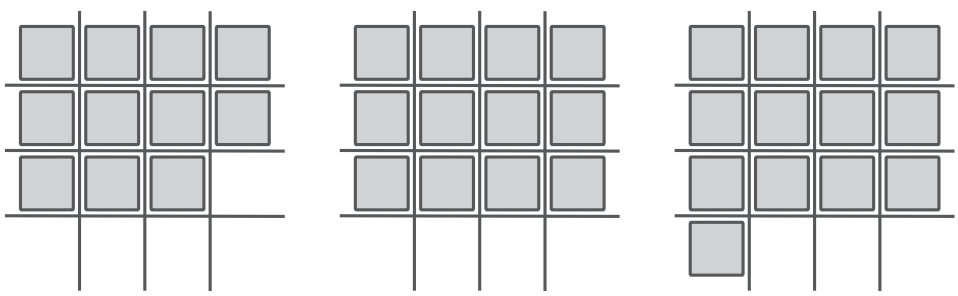
(a) 1 element (b) 2 elements (c) 3 elements



(d) 4 elements (e) 5 elements (f) 6 elements



(g) 7 elements (h) 8 elements (i) 9 elements



(j) 11 elements (k) 12 elements (l) 13 elements

Figure 3.22: Initial algorithm for positioning devices

Well, in Figure 3.22 on the facing page is obvious that this does not happen with sixteen elements or less, and that is a sufficiently big number of devices a person can own in the system. But we can mathematically prove this by calculating if a grid of $(x-1)\cdot(x-1)$ elements can hold more elements that the grid of $x\cdot(x-2)$.

If that is the case, then we would not need to shrink the grid in any case by two because the grid will be already horizontally shorter. It is quick to prove this is true following the steps explained from eq. (3.3) to eq. (3.5).

$$(x-1)\cdot(x-1) < x\cdot(x-1) \tag{3.3}$$
$$x^2-2x+1 < x^2-2x \tag{3.4}$$
$$1 < 0 \tag{3.5}$$

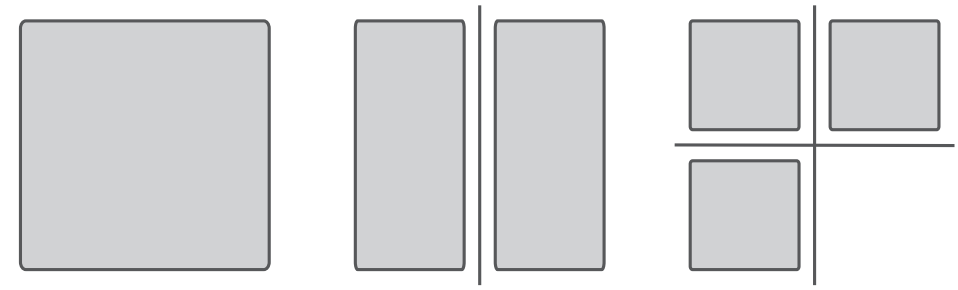
Figure 3.23 on the next page shows the results after applying this final algorithm, filling more space in some situations. Using this algorithm we can calculate the height, width, vertical and horizontal offset for every element, so that it keeps its aspect ratio while placing it in the middle of the allocated space.

If we want to work with percentages, we only have to divide the size of the container between the number of columns and rows. For example, if we want to fill the container at 100%, then every element will be of size $100/x \times 100/y$. The actual values for the offset needed for every particular element is then easy to calculate if we fill the canvas one by one.

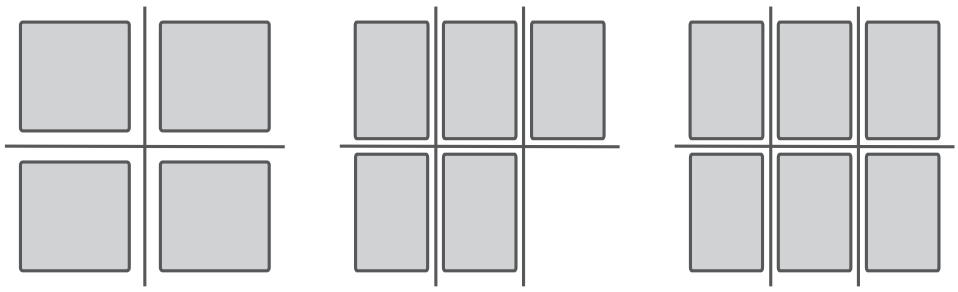
3.7.2 Remembering Positions

Next time a user visits this page, the system will remember the last position of those elements instead of calculating the grid again. Because the user can change the size of the window at any time, we cannot relay on fixed positioning with pixels, because our canvas could be bigger or, worse, smaller than the one we have calculated. Besides being not very elegant, we can find several situations where the page is unusable.

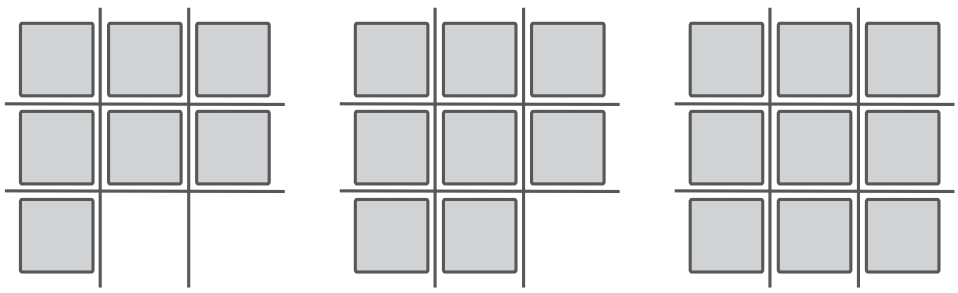
The best way to avoid all that trouble is treating every position or size in terms of percentages. This is how is done in the code, and it allows the user



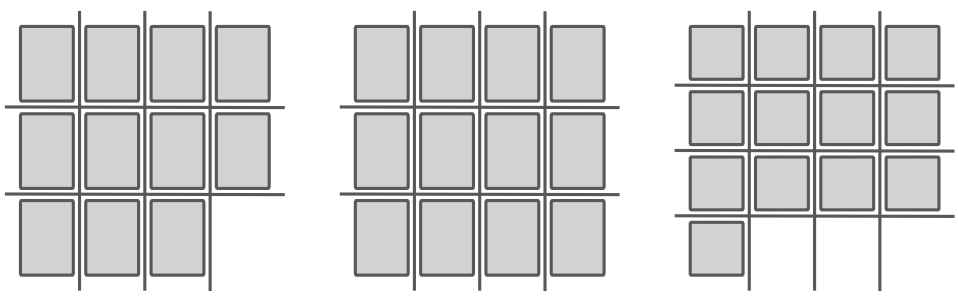
(a) 1 element (b) 2 elements (c) 3 elements



(d) 4 elements (e) 5 elements (f) 6 elements



(g) 7 elements (h) 8 elements (i) 9 elements



(j) 11 elements (k) 12 elements (l) 13 elements

Figure 3.23: Final algorithm for positioning devices

to resize the window at any time: the devices will be resized dinamically according to that window size.

To store and retrieve painlessly these values, we are going to take advantage of a useful MooTools class: `Hash.Cookie` [9]. With this utility, we only have to specify the name for the `Cookie` and we can store a `Hash` into a `Cookie` without worrying about the `Cookie` itself. Besides loading the data of the `Cookie` directly on the `Hash` at its creation, if we change a value of the `Hash` it will be automatically updated in the `Cookie`.

The reason for using a `Cookie` is mostly because it reduces complexity on the server, since it does not have to store the position of every device. Other good reason is that it is the most simple way of allowing different arrangements in different places; for example the user may want to arrange radically different its devices in a big screen like in a TV or on a smaller screen like in a netbook. Finally, it is universal as it is supported by almost every browser.

The final decision is to have one `Cookie` for each device. This is very straightforward for the implementation, since a `Cookie` can have the name of the container. Each `Hash` is composed by the four values needed for positioning the element: `offsetX`, `offsetY`, `width` and `height`, as percentages respect the container. An example is presented in Listing 3.2.

Listing 3.2: Cookie Hash example

```
{
  offsetX: 15,
  offsetY: 50,
  height: 10,
  width: 20
}
```

Then, each time the object size or dimension changes, the `Hash` (and therefore the `Cookie`) is updated. These changes happen mostly in two situations: when we resize a device (changing its size but no its position) or when we move around a device (changing its position but not its size).

3.8 Software Validation and Verification

Due to the nature of the service applications running in the demonstration, none of them include unit tests or any other form of automated software testing. Though these tests are a proved tool for developing reliable software, the scope of this project is limited to demonstrations purposes and the resulting application will not be used in the wild by many people.

Therefore it is preferred to speed up the development as much as possible rather than testing every little possibility so that no minor bugs are introduced. For these reasons this application did not include those tests, and the validation process consisted on trying all major use cases in all devices and browsers. For development and testing there were several devices:

- The main development laptop, running Ubuntu. To ease the testing phase, two virtual machines were used in this laptop*:
  - Windows Vista with IE 7.
  - Windows Vista with IE 8.
- A laptop running Windows XP.
- A desktop computer running Windows Vista, with a TV acting as its monitor screen.
- A laptop running Windows Vista.
- An iPhone 3G running iOS 3.

Most of the development time was spent on the Ubuntu laptop, using Google Chrome as the default browser because of the fantastic debugging tools it has built in. On the other machines, several browsers were tested in a regularly fashion to ensure the application to keep a certain compatibility with them. Table 3.32 on the next page lists all used browsers and under which platforms they were tested.

*Microsoft offers free virtual machines to test websites with all major versions of IE: http://www.microsoft.com/download/en/details.aspx?id=11575

Table 3.32: Test browsers

IE 7 (Windows)	Firefox 3.6 (Windows/Ubuntu)
IE 8 (Windows)	Opera 10.5 (Windows)
Safari 4 (Windows)	Google Chrome 5 (Windows/Ubuntu)

Initial testing also included IE 6, but as the project advanced it was clear that support for this dated browser had to be dropped in order to maintain a sane development pace. Even then, a considerable amount of the project time was spent fixing bugs with more modern versions of IE, by far the most buggy of the five browsers.

The mobile version was properly tested just in the Safari browser for the iPhone during all development, and also in Google Chrome under Ubuntu, given its a similar engine – WebKit – and that it is much faster and easier to debug. At the end, some further testing in the Android emulator was done, but only to assess that the main interface was functional, knowing that the overall design was going to look different.

Continuously, and specially in every major step in the process, the application was tested in every browser. Detailing here a formal collection of test cases will not add any useful information to this document, so that was dismissed.

Actually, real tests consisted on manually trying every path of the 24 use cases, the first 15 in the desktop browsers and the last 9 in the mobile browser. Tables on § 2.2.3 on page 16 and § 3.3.1 on page 121 are exhaustive enough to describe the initial conditions and the expected outcome. Also, in § 3.5 on page 150 lots of pictures already depict the real outcome; they are not mockups but real screenshots of the application running under several browsers.

3.9 Deployment

The installation of the whole ScaleNet application is out of this project scope, however it is necessary to explain the upgrade from a previous version. Since one of the goals was to be as less disruptive as possible, it should be fairly easy to deploy the new system into an existing installation. The steps can be roughly summarized in three:

Replace the PHP sources. The whole `scalenet` directory has to be copied over the old one in the same location.

Install the APE server. This just means installing a package.

Configure BIND. DNS needs some custom rules.

3.9.1 Install the APE Server

The APE download page [10] contains packages for different operating systems and architectures. In this case, since the system is Debian-based we should use the DEB package. Once the correct package is downloaded, it can be installed on the Application Server by typing Listing 3.3 from the same directory as the package is stored.

Listing 3.3: APE installation command

```
sudo dpkg -i ape-1.0.i386.deb
```

After that, the APE server daemon (`aped`) is automatically started with the default configuration [11]. It can be checked by visiting the url `webportal.imusu.mobile.dtrd.de:6969`.

3.9.2 Configure BIND

The IMS core is the machine that provides the DNS service through BIND, and that service needs to be configured to allow the APE server to use a lot of different dynamic subdomains like `1.ape.webportal`, `2.ape.webportal`, `567.ape.webportal`, etc.

This is how the APE server works by default, and it appears that there is no way to configure the APE server for using only one domain [12].

So, in the file `/etc/bind/imusu.dnszone` located in the IMS core we have to look for the *webportal* entry and change that section to look like Listing 3.4.

Listing 3.4: BIND configuration

```
webportal              1D IN A          192.168.5.234
ape.webportal          1D IN A          192.168.5.234
*.ape.webportal        1D IN CNAME      ape.webportal
```

To apply the changes, we have to restart BIND using the command in Listing 3.5.

Listing 3.5: BIND restart command

```
sudo /etc/init.d/bind restart
```

Chapter 4

Discussion and Outlook

Fry : But I know you in the future.
I cleaned your poop.
Nibbler : Quite possible. We live long and are celebrated poopers.

The Why of Fry
Futurama

4.1 Discussion

The work done in this project has been evaluated favorably by *the client*, that is, the T-Labs staff that asked for this work. The outcome did fulfill the requirements, so in that sense the project could be qualified as a success.

The developed codebase ended very organized, comparing to other JavaScript projects, so it may be easy for others to continue evolving that implementation. Though hacks and dirty fixes needed to be spilled all over the place because of IE quirks and bugs, that did not tarnish the overall architecture. In conclusion, MooTools not only took care of cross-browser issues, but it also helped organizing the codebase.

In the desktop, the resulting interface is much more functional and rich, the user experience has improved a lot and future demonstrations should be a bit more effective. The fact that no underlying components need to be changed for that is a very nice addition, as it does not complicate the deployment.

The new desktop interface is also very customizable, since the user can move around his devices or the sidebar and leave the interface exactly as he wants. But it is more important that now the web page can transparently adapt to different screen sizes, because the initial static version was only usable in a big screen.

Another point made in this work is that, with the HTML5 and the latest additions to current browsers, it is becoming relatively easy to replace external plugins such as Java applets with plain JavaScript. The developed solution is now more broadly compatible with current devices.

Giving that support for this language is getting increasingly better in all platforms, that the new generation of mobile devices do not run these plugins, and that native SDKs are very heterogeneous between them, the only real cross platform solution will be have to be based on JavaScript. Therefore it is important to verify that web applications can be almost as powerful as any native application.

The mobile version of the app also proves that a mobile web application can be as intuitive as any native application. The use of elements that are similar to the native elements, not in their appearance but in how they interact, it is very important to create a feeling of familiarity in the user. The

UI mechanisms are instantly recognized by the user so he already know how the app works without needing any help.

Once installed, the integration with the iOS/Android platforms is pretty nice, the user accesses the website exactly as any other native app. However, the discovery and *installation* of web apps are far less convenient for the user compared to native apps, since there is no web app market integrated into the system.

That and other limitations are only derived of current implementations, there are no technical impediments to create a web application market as the Chrome Web Store* proves. In the future, it will be very probable that more platforms will opt for embracing that option.

There is a severe drawback in that mobile version, one that the desktop version do not suffer from. Although the current generation of mobile browsers is very advanced compared with the previous one, they are still a step behind in efficiency and it is difficult to achieve complete fluidness.

More specifically, the mobile version suffers from a rendering issue with the current Safari browser when viewing the content list. When the user changes to the content list, the browser engine has to render more information than it is designed to handle at once, so it starts painting the elements in blocks (see Figure 4.1).

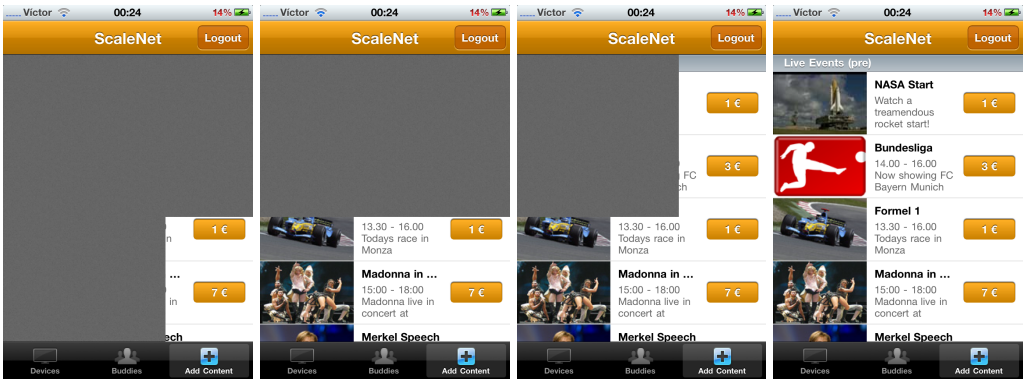


Figure 4.1: Rendering issue in the mobile PNAI interface

When the view is fully drawn, scrolling still feels sluggish as the new revealed content gets painted. However, if the content list is scrolled to the end, it can be scrolled without no rendering issue from the beginning,

*https://chrome.google.com/webstore

which proves that the browser could handle that web application nicely if we could pre-load the hidden views.

The view is loaded with lots of medium sized images, and certainly the CSS3 rules used to paint the buttons does not help, but the root of the problem is the lack of support for fixed positioned element. That forces us to use the iScroll library, adding more overhead as the scrolling has to be done programmatically by JavaScript instead of embracing the much snappier native scrolling.

This problem is known by popular mobile applications, and one solution consists of rethinking the interface so that the content is scrolled using the native method, and the tab bar is not fixed but it moves as the browser scrolls. Google apps like Mobile Gmail or Google Reader used this interface for a while, but it felt kind of weird so they evolved towards more natural approaches.

The real solution to this problem comes with iOS 5, as it includes support for fixed elements and native scrolling not only for the whole page but also for inner elements. Once this update lands, iScroll will not be needed anymore and therefore the page can be very fluid keeping the same experience we want.

4.2 Outlook

There is still room for improvements that can be implemented in the application, and the design can be stylized by a professional designer. In the future, it could be implemented for real users, maybe in a next generation network or in an existing network. Before that, some shortcomings have to be resolved, like reflecting the real registered devices in the interface for each user.

Like with IPTVplus, other ScaleNet services could be integrated in the same interface. Also, it could be interesting to redesign the other pages of the system or even integrate them into the same interface. For example, it would be useful some controls to manage the devices or the buddies

– adding, editing and removing them – all in the same place, instead of having to go to external pages.

Looking forward to the future, the mobile version can be reengineered in some ways to make it more responsive. Since the development was done with an iPhone, Android only was poorly tested in a simulator. Therefore, it will be desirable to further test and polish the web application in Android devices, Windows 7 phones and even making it bigger for the iPad and tablets.

At the same time, other routes can be explored, by converting this web page into a native app. It would be appealing to evaluate how can this web be ported to native platforms, either by using only native SDKs or by mixing the existing code with native code.

Appendix A

Budget

Dexter : It seems ironic that I, an expert on human dismemberment, have to pay 800 dollars to have myself virtually dissected.

The Lion Sleeps Tonight
Dexter

A.1 Project Phases

As explained in § 1.3 on page 4, the project could be divided equally in three phases. Giving that the project accounts for a total of six months, each phase takes approximately two months. Figure A.1 on the facing page depicts a Gannt diagram with the project phases and their tasks. Each phase can be roughly divided into three tasks:

Design This task consists on designing both the interface and the internal architecture. In this phase some state of the art research has to be done, but due to previous experience in the field it could be completed at the same time the system is designed.

Implementation This task refers to the actual coding according to the prior design.

Testing This task just includes testing and debugging the application in all relevant browsers.

In reality, due to the dynamic nature of the project, these tasks are not as easy to divide as they collide: some design is done with actual code, and when implementing something it is almost impossible to not test the code to some point. However, those are a good approximation of the main tasks in those weeks, in that there is more designing at the beginning and more testing at the end.

In the first phase, the design task is important because it includes the initial research, but it was very easy to start since I had already the previous version to tinker with. The implementation task is very important in this phase because all the JavaScript codebase will be ported to MooTools, so there will be a lot of changes, not in the external interface but in the real code.

The first testing task was not decisive because IE compatibility is delayed to the second phase, so the only thing to do is trying the developed application in the real demonstrator.

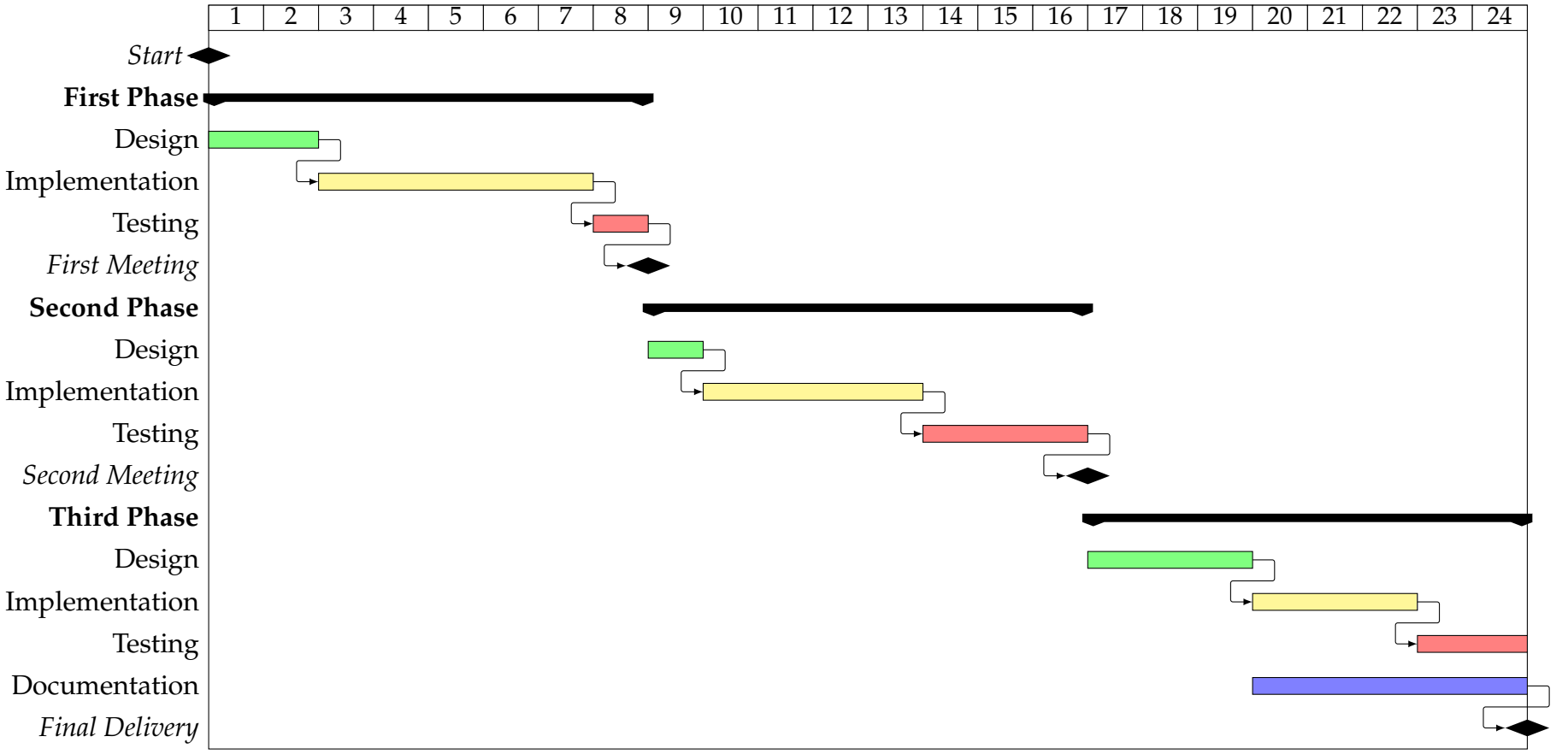


Figure A.1: Project schedule

In the second phase, the design is not very time consuming, since the initial research is already done and the interface is more or less defined. Graphically, only the sidebar has to be designed, and it is easy to design the architecture of the new functionality because the paradigms are already defined.

The implementation is again the beast, since the Java code is fully ported to JavaScript, new classes have to be implemented and some PHP scripts have to be written to retrieve the content the application needs. The testing is now much more time consuming, since testing with IE is contemplated.

In the third and final phase, the mobile interface was designed from scratch, that explains why the design task takes longer. In contrast, there is not a lot of code to be written, because the code from the desktop version will be reused, so the implementation task is shorter than before.

Finally, the testing task includes proving that everything is working in every browser, so that should take some time. But it will be shorter than in the second phase, because the relevant bugs with IE were already polished. In parallel to this third phase, certain documentation needs to be written for future reference, so some time for that is reserved.

At the end of each phase, there is a meeting with all members involved in the project to explain how the work is progressing. In these meetings not only the results are shown but also important feedback is retrieved from my tutor and other T-Labs immediate superiors. The last meeting is reserved to summarize all work and to deliver the software.

A.2 Material Expenses

The material expenses only account for the material needed for the development, that is, the two laptops and the iPhone. Equipment used in the demonstrator is not included in the expenses because as a shared resource those computers are considered already amortized by previous projects.

Due that the provided machines belongs to T-Labs and they will be repurposed after this project, it would be misleading adding the whole cost to this project budget. A life of two years (24 months) is considered for the

iPhone, and three years (36 months) for the laptops. Since the project only lasts six months, a simple amortization is used to calculate the real cost.

Table A.1 comprises the computed material expenses in euros. Two laptops, one display and one phone are considered for these calculations, the equipment used those six months specifically for this project. Software costs are not taken into account because all software used is free; the only software with cost is Windows XP and the cost of that SO is already included into the laptop cost.

Table A.1: Material expenses

Concept	Cost/unit	Amortized %	Total amount
Fujitsu-Siemens Lifebook S7020	700	16.67%	116.67
Fujitsu-Siemens Lifebook S6420	700	16.67%	116.67
Toshiba PA 3553 (Display)	200	16.67%	33.33
iPhone 3G 8GB	600	25.00%	150.00
		Total without VAT	416.67
		VAT (16%)	66.67
		Total	483.33

A.3 Human Resources Expenses

The most important cost in the project is the engineer's wage, the person responsible for developing this project. Figure A.1 on page 201 will help estimating the resulting cost, considering a working day of 8 hours, a working week of 5 days, 24 weeks of work, and an average wage of 50 Euros per hour of work.

However, not every task requires the 100% time or effort of that engineer, he could be working on other parallel projects (indeed, he was). Because of that, an estimated effort is assigned to each task. Table A.2 details the human resources cost taking all this into account; as before, amounts are in euros.

Table A.2: Human resources expenses

Concept	**Hours**	**Effort**	**Total amount**
Design (1st phase)	80	100%	4,000
Implementation (1st phase)	200	100%	10,000
Testing (1st phase)	40	60%	1,200
Design (2nd phase)	40	60%	1,200
Implementation (2nd phase)	160	40%	3,200
Testing (2nd phase)	120	60%	3,600
Design (3rd phase)	120	80%	4,800
Implementation (3rd phase)	120	80%	4,800
Testing (3rd phase)	80	80%	3,200
Documentation	200	20%	2,000
		Total	38,000

A.4 Total Expenses

Table A.3 shows the final budget of the project, by adding the material expenses and the human resources expenses. As before, amounts are expressed in euros.

Table A.3: Total budget

Concept	**Cost**
Material expenses	483.33
Human resources expenses	38,000
Total	38,483.33

The total cost of this project is **thirty eight thousand, four hundred and eighty three euros with thirty three cents**.

Bibliography

[1] Matthias Siebert et al. *ScaleNet – Converged Networks of the Future*, April 2006. it - Information Technology, ISSN: 1611-2776, volume 48, pages 253–263. Also available online from: http://telematics.tm.kit.edu/publications/Files/204/it0605_253a.pdf. 2.2

[2] Dmitry Sivchenko et al. *ScaleNet – IMS Demonstrator*. Deutsche Telekom Laboratories, August 2008. Internal Document. 2.2.1

[3] Dmitry Sivchenko et al. *ScaleNet – Personal Network Administration Interface*. Deutsche Telekom Laboratories, June 2008. Internal Document. 2.2.3

[4] Dave Raggett et al. *HTML 4.01 Specification*. W3C, December 1999. Last checked on October 2011. http://www.w3.org/TR/html401/. 2.5.1

[5] Bert Bos et al. *CSS 2.1 Specification*. W3C, December 2010. Last checked on October 2011. http://www.w3.org/TR/CSS21/. 2.5.2

[6] W3C. *CSS Current Work*. Last checked on October 2011. http://www.w3.org/Style/CSS/current-work. 2.5.2

[7] Víctor Pimentel. *Las novedades de HTML5* (Spanish). AnexoM, Jazztel, June 2009. Four part article series last checked on October 2011. http://www.anexom.es/tecnologia/diseno-web/las-novedades-de-html5-i/

http://www.anexom.es/tecnologia/diseno-web/las-novedades-de-html5-ii/
http://www.anexom.es/tecnologia/diseno-web/las-novedades-de-html5-ii/
http://www.anexom.es/tecnologia/diseno-web/las-novedades-de-html5-y-iv/
. 2.5.3

[8] Jesse James Garnett. *Ajax: A New Approach to Web Applications*. Adaptive Path, February 2005. Last checked on October 2011. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. 2.6.4

[9] MooTools. *MooTools Docs: Class Hash.Cookie*, Version 1.2.5. Last checked on October 2011. http://mootools.net/docs/more/Utilities/Hash.Cookie. 3.7.2

[10] Weelya. *APE Server Download*, Version 1.0. Last checked on October 2011. http://www.ape-project.org/download/APE_Server.html. 3.9.1

[11] Weelya. *APE Setup Manual*, Community Wiki. Last checked on October 2011. http://www.ape-project.org/wiki/index.php/Setup. 3.9.1

[12] Weelya. *Advanced APE Configuration*, Community Wiki. Last checked on October 2011. http://www.ape-project.org/wiki/index.php/Advanced_APE_configuration. 3.9.2

Acronyms

AJAX	Asynchronous JavaScript and XML
APE	Ajax Push Engine (see §2.8)
API	Application Programming Interface
AS	Application Server
BIND	Berkeley Internet Name Domain
CGI	Common Gateway Initiative
CSS	Cascading Style Sheets
DAS	Device Access Specification
DNS	Domain Name System
DOM	Document Object Model
ECMA	European Computer Manufactures Association
FMC	Fixed and Mobile Convergence
GPL	General Public License
GUI	Graphical User Interface
JAR	Java Archive
JIT	Just-in-time compilation
JSF	JavaScript Framework
JSON	JavaScript Object Notation

JSONP JSON with padding

JVM Java Virtual Machine

HTML HyperText Markup Language (see §2.5.1)

HTTP HyperText Transfer Protocol

IE Internet Explorer

IIS Internet Information Services

IMS IP Multimedia Subsystem

IP Internet Protocol

IPTV Internet Protocol Television

ISO International Organization for Standardization

MIME Multipurpose Internet Mail Extension

MMOG Massively Multiplayer Online Game

NGN Next Generation Network

NB Notebook

OOP Object-Oriented Programming

OSGi Open Services Gateway Initiative (see §2.4.3)

P2P Peer-To-Peer

PDA Personal Digital Assistant

PHP PHP: Hypertext Preprocessor (see §2.3)

PNAI Personal Network Administration Interface (see §2.2.3)

QoS Quality of Service

RGB Red Green Blue

RGBA Red Green Blue Alpha

SIP	Session Initiation Protocol
SGML	Standard Generalized Markup Language
SSCON	Session Controller
TCP	Transmission Control Protocol
T-Labs	Deutsche Telekom Laboratories
TV	Television
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
VOD	Video on Demand
VoIP	Voice over IP
W3C	WWW Consortium
WEBGL	Web-based Graphics Library
WHATWG	Web Hypertext Application Technology Working Group
WOFF	Web Open Font Format
WWW	World Wide Web
XHTML	Extensible HyperText Markup Language
XML	Extended Markup Language

Index