Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería en Informática

PROYECTO FIN DE CARRERA

Developing a Heavy Client-Side Web Application: ScaleNet

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

March 2011

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

(no me echéis de casa que ya me voy yo)

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 exist, 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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1.2 Objectives

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1.3 Project Phases

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1.4 Document Structure

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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 Ajax Push Engine (APE) server. The other major addition to the system was the JavaScript Framework called MooTools. Both decisions are explained and justified in §2.7 and §2.8.

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*. Deutsche Telekom Laboratories (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].

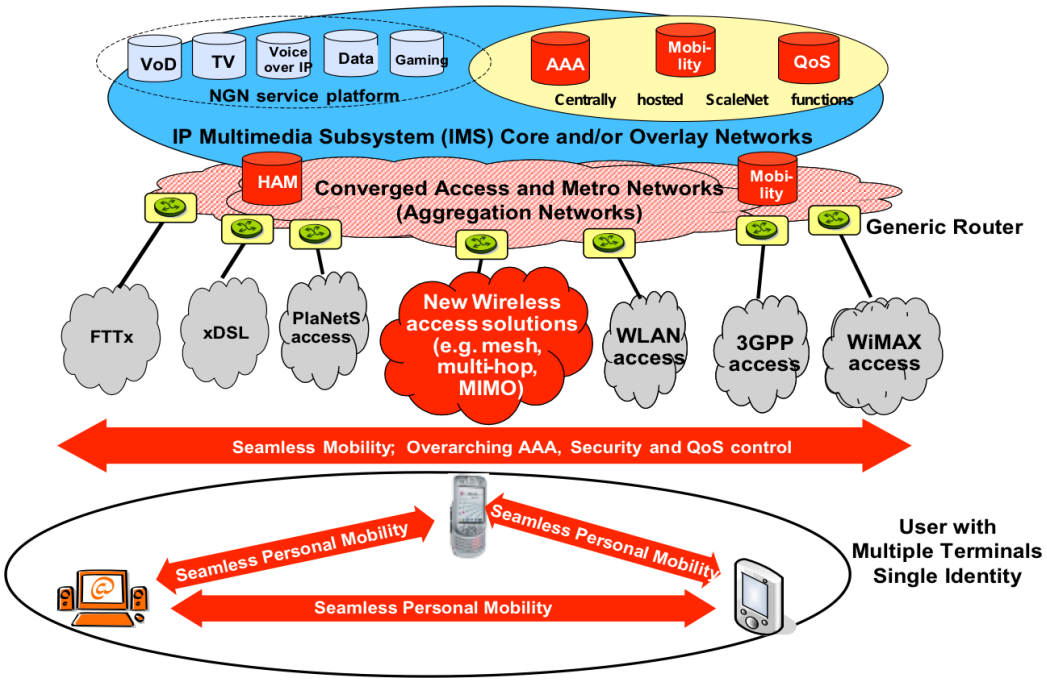


Figure 2.2: Structure of the system

At an upper level, multimedia services relay on the IP Multimedia Subsystem (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 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

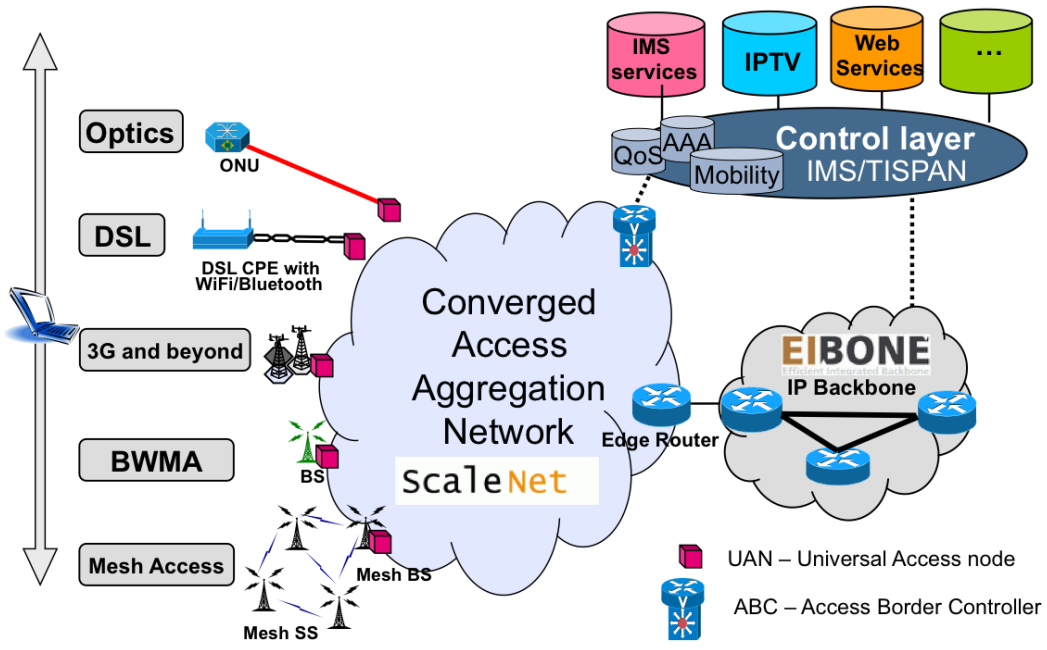
The work described in this document is primarily focused on the second application, i.e., video streaming. The idea is that the user can buy a video and play it anywhere using any supported device.

2.2.2 IMS Demonstrator

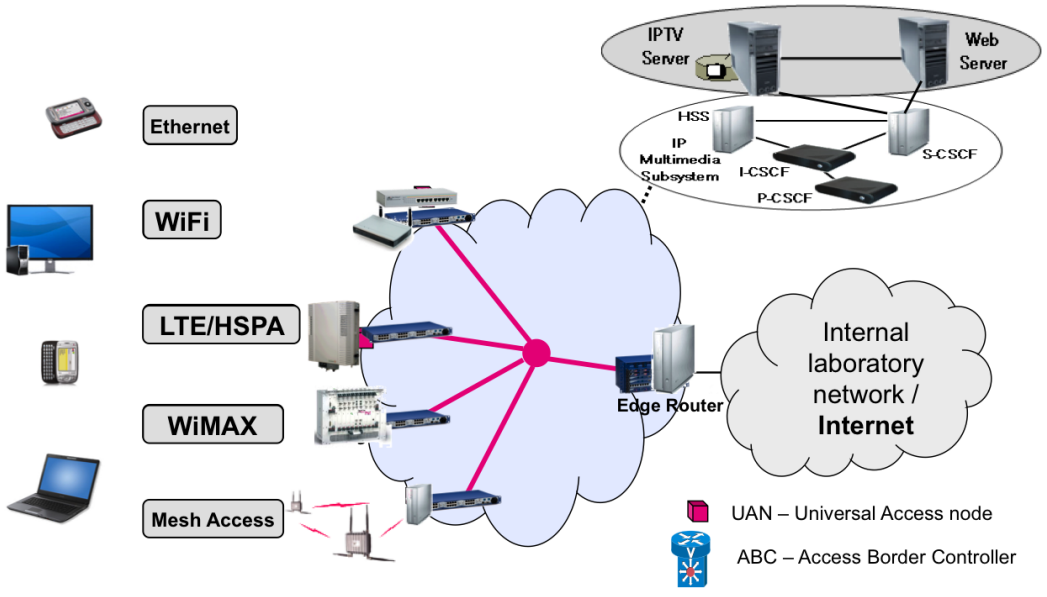
A logical view of the system is depicted in Figure 2.3(a), explaining the important nodes based on the capabilities needed. The information relevant to this project is contained in the upper right corner of the figure, the nodes behind the control layer.

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) the actual network and hardware are exposed, replacing the same space as in the logical view (Figure 2.3(a)).

Figure 2.4 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 belongs



(a) System architecture



(b) Architecture of the demonstrator

Figure 2.3: IMS architecture

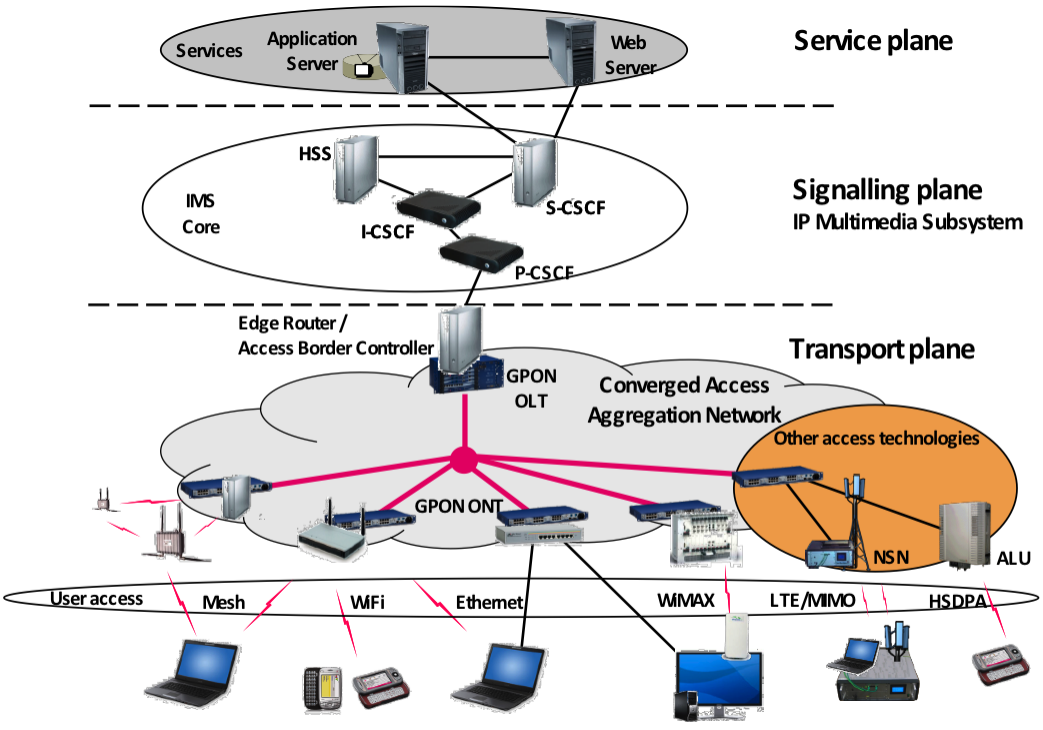


Figure 2.4: Setup of the demonstrator

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: Hypertext Preprocessor (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

*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/

it serves Java applications based on the Open Services Gateway Initiative (OSGi) framework*.

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 Personal Network Administration Interface (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 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:

*http://www.osgi.org/

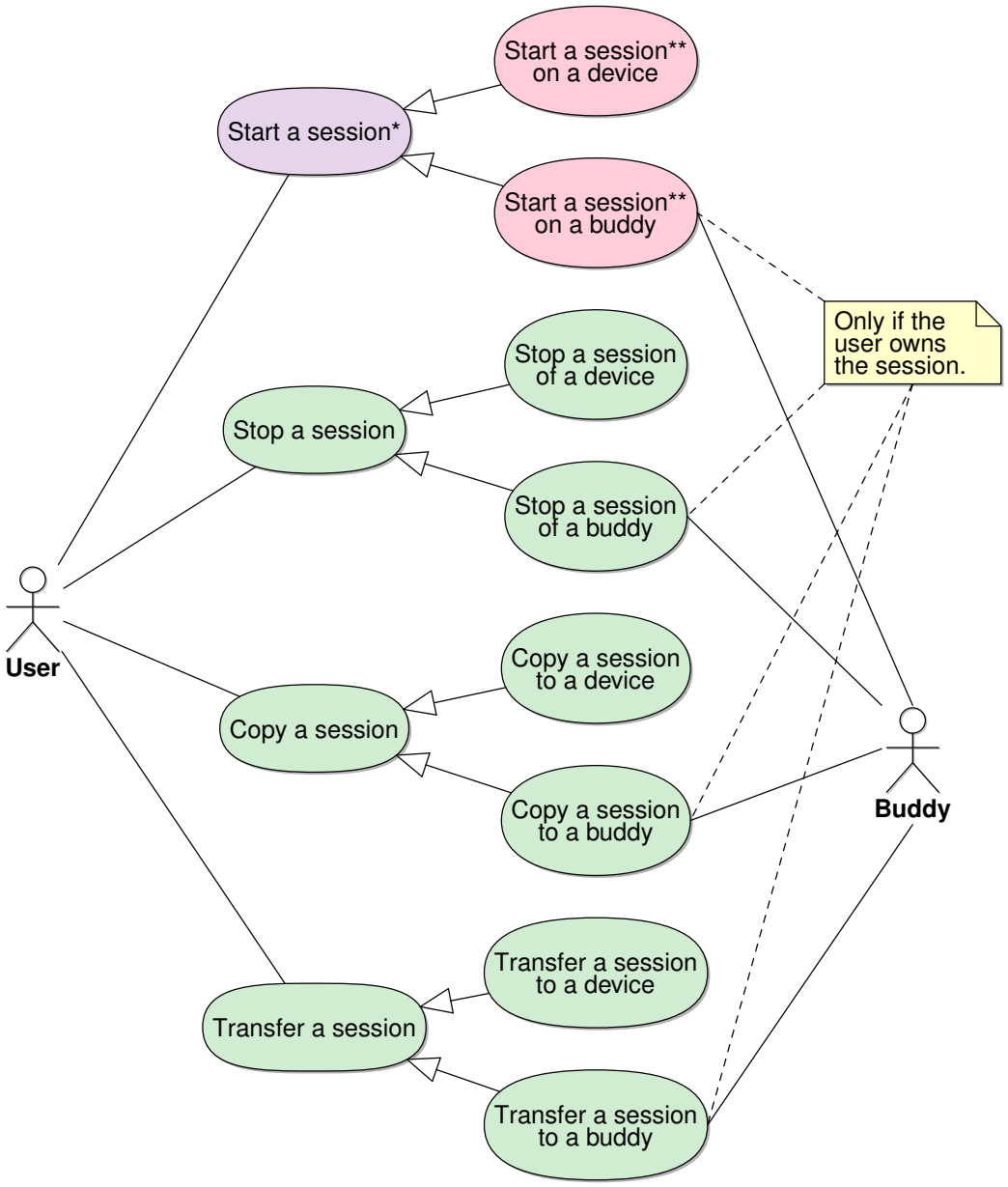


Figure 2.5: Use cases for the IPTV application

Green Available already in the main PNAI page.

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 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

Figure 2.6: Old PNAI page

another container to copy or transfer that session. That is a very visual and fast way to manage sessions.

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
	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**
Precondition	A session is already running on a device, and it is showing in the PNAI interface inside of that 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.

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**
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: Deleting a session in the old PNAI

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

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**
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, 2.4, 2.5 and 2.6 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.

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

USE CASE 3	**Copy a session to a device**
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.
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 2.4: Use case 4 – Copy a session to a buddy (continued)

USE CASE 4	**Copy 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/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.
Alternate Path (A2)	**5c.** The user clicks on the cancel option. **6c.** Popup menu disappears and action is cancelled.

continued on next page

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

USE CASE 4	**Copy a session to a buddy**
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, the software is mainly executed in three machines: two servers and a client. Figure 2.9 shows all the relevant components

running inside those machines and how they interact with each other.

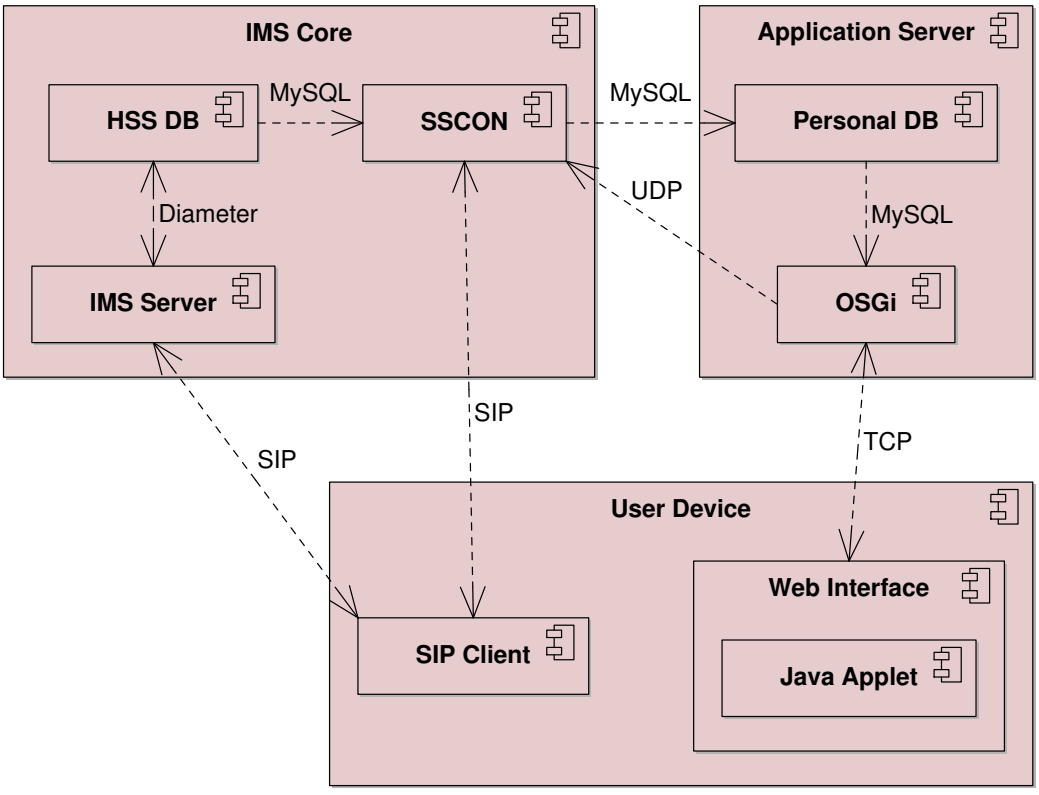


Figure 2.9: Component diagram for the old PNAI

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 shows the flow of the application in the scenario where the user wants to transfer a session from his device to one of those 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

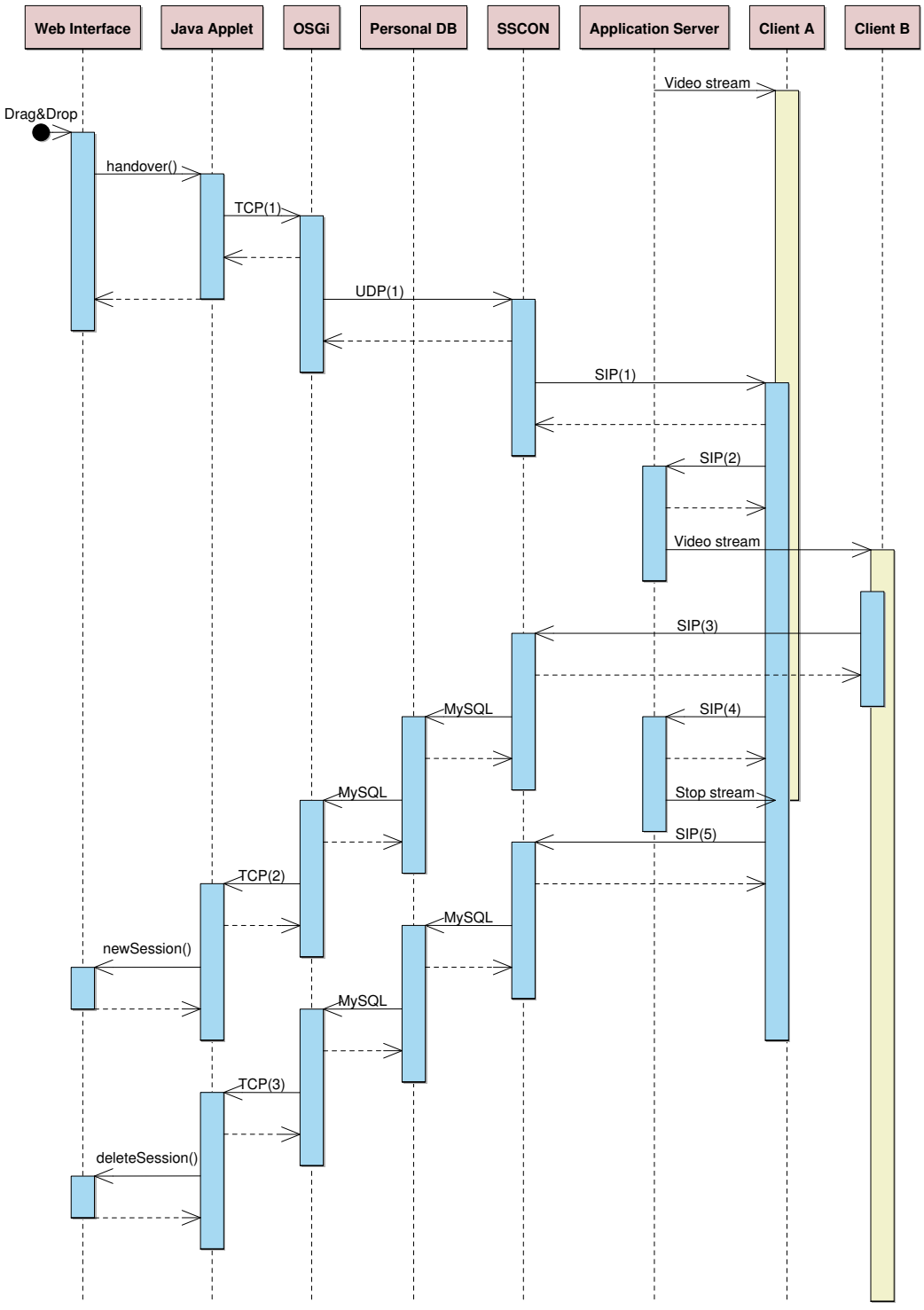


Figure 2.10: Sequence diagram for the old PNAI

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 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 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
impu_id	Unique id to identify impu

continued on next page

Table 2.9: User status table architecture (continued)

Field name	Description
`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 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)
`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, 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`

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`

continued on next page

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

Notification	**Format & Example**
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, 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 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 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.

Figure 2.11 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 and 2.14. 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, 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, *class* is not the right term to refer to a JavaScript object, but it will be used through this

*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.

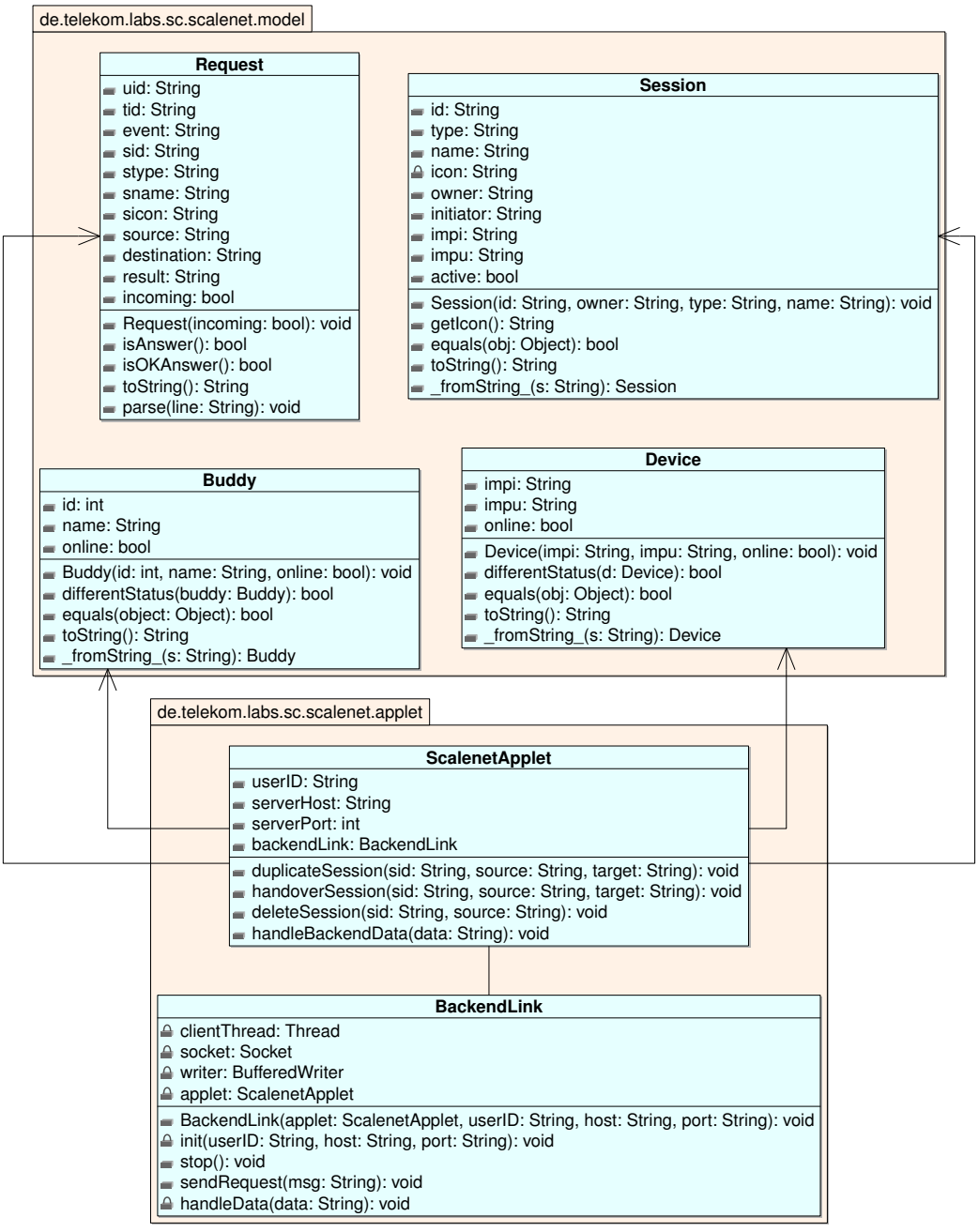


Figure 2.11: Class diagram for the Java applet

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 shows a diagram with all the classes involved and the relationships between them. Figure 2.13 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 `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 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,

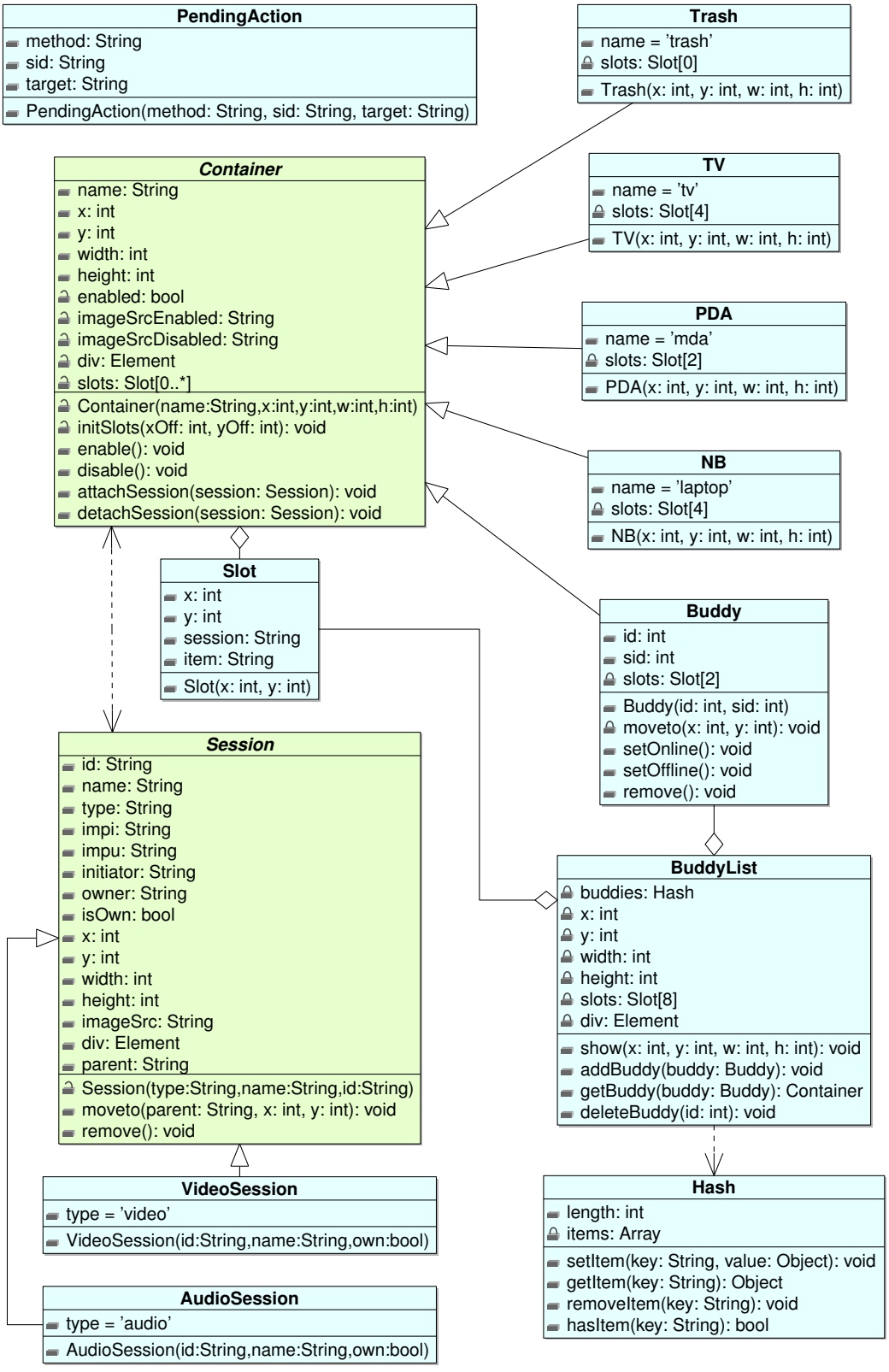


Figure 2.12: Class diagram for the old PNAI

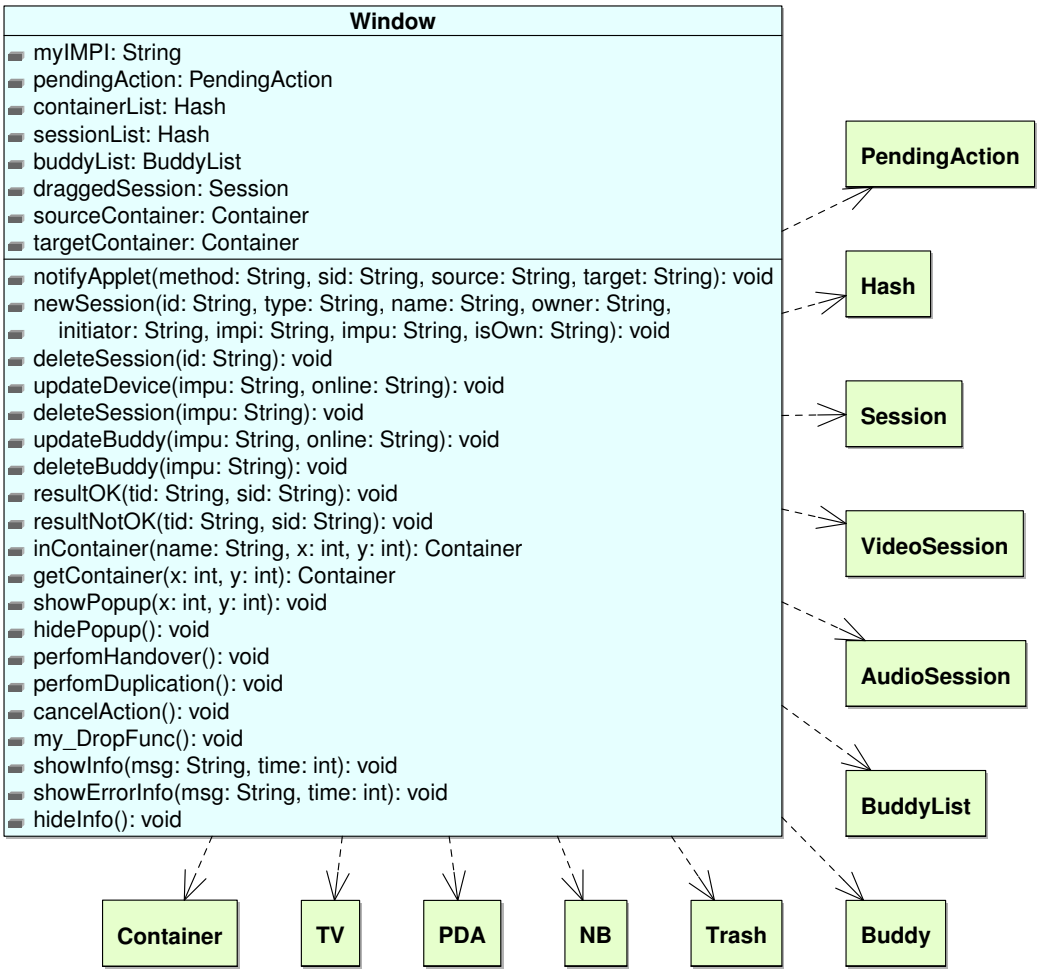


Figure 2.13: The global JavaScript object in the old PNAI

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

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

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 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 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.

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.

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);
```

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 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 these pages are written in PHP.

Figure 2.15 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

Figure 2.14: Old IPTVplus page

```
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

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.

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, 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.

*http://www.php.net/

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 a HTML file with the `.php` extension that it is usually stored in the public folder of the server (Apache, Internet Information 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 shows the resulting HTML output.

Listing 2.2: PHP code embedded within HTML code

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

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>
```

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 `$_SERVER` offer access to information sent from the browser, so it is commonly used to pass parameters to a PHP script (through the URL or forms in the page). On the other hand `$_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 `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 with it 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 HyperText Transfer Protocol (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.

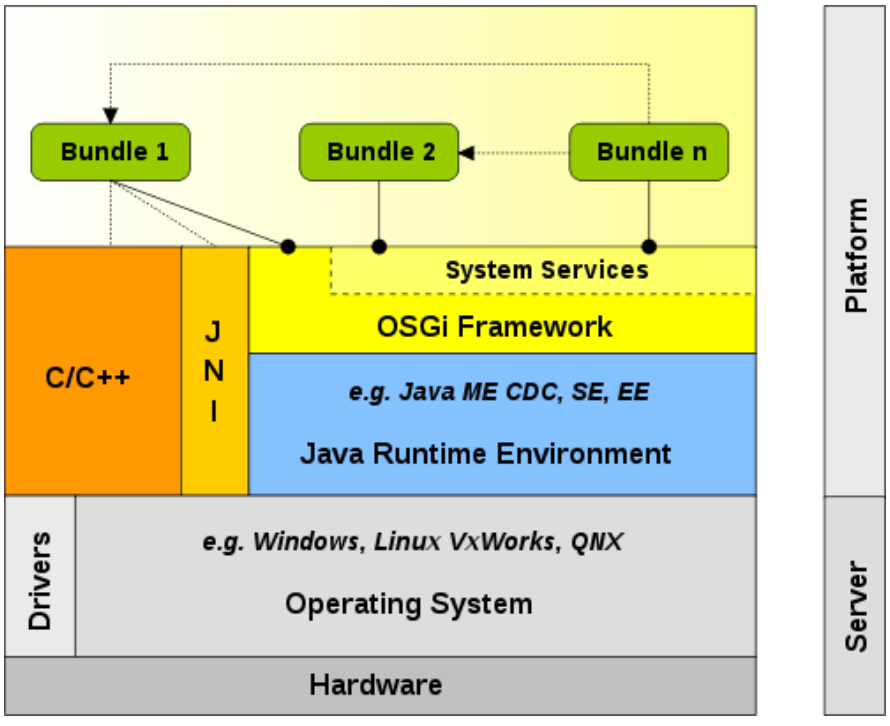


Figure 2.18: OSGi layering

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 through a Web Browser. There are several vendors that offer competing products, but the main ones have been evolved almost in a monthly basis following the growth of the web. These browsers interpret the code and generate a visual (or audible) representation suitable for human consumption.

The language 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.

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 51 shows how a 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 a International Organization for Standardization (ISO) standard. 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.

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.

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 51 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 bad formed pages, in many occasions it sadly backfires to a quirks mode. 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. Internet Explorer (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))*, this has been the primary frustration for every front end developer, and a tremendous drawback for web applications. For example, for 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)). 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.

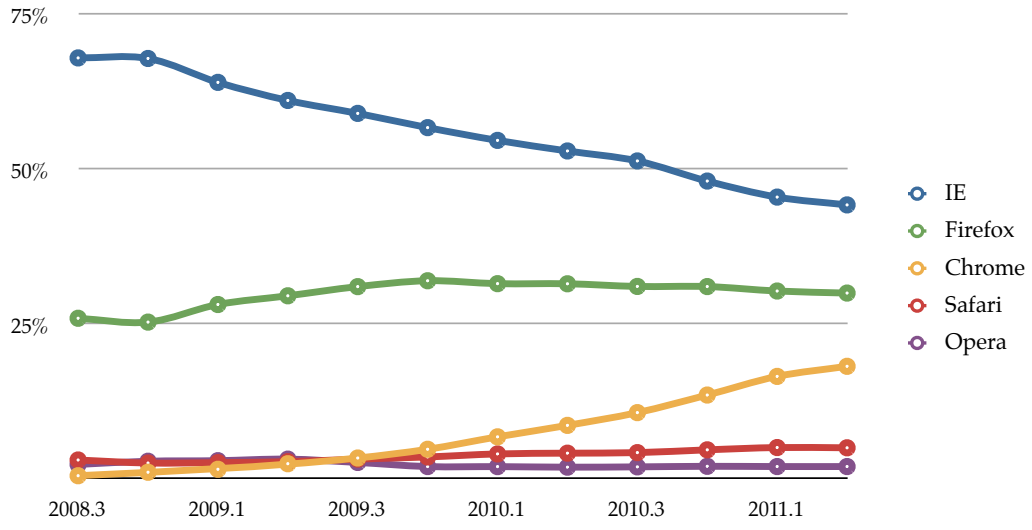
2.5.2 CSS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

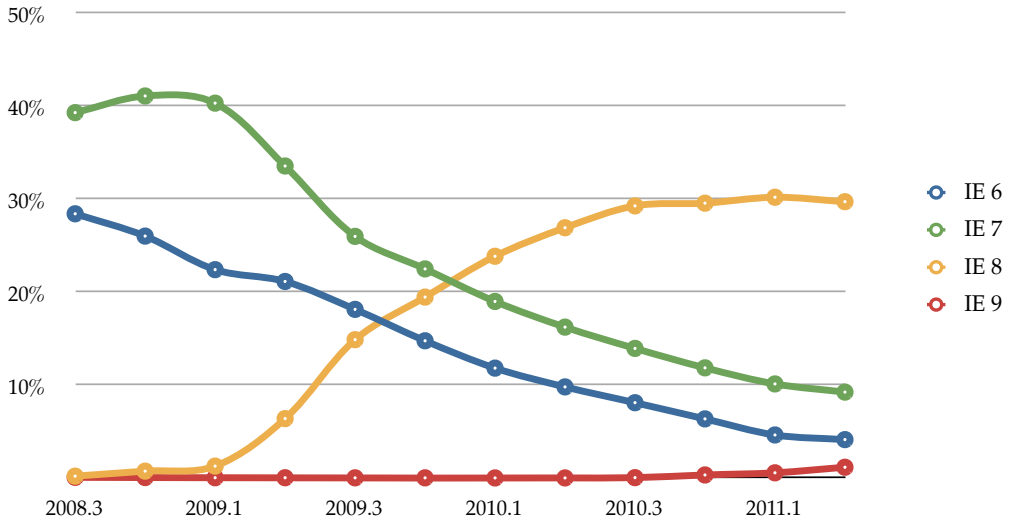
2.5.3 HTML5 and CSS3

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

*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 for the top five browsers



(b) Market share for the current versions of IE

Figure 2.19: Current browser trends

2.6 Client Programming Language: JavaScript

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.6.1 Quick Overview of the Language

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.6.2 Compatibility

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.7 JavaScript Framework: MooTools

Before explaining why MooTools has been chosen for this application, another important question needs to be resolved.

2.7.1 Why Use a JavaScript Framework?

There are several reasons that lead to this conclusion, the most important are:

- 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 Internet Explorer and the rest of the browsers.
- Because we want to facilitate the development, since usually these frameworks cover several holes in the JavaScript specification that allows us fixing common issues with less code.
- 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.

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. However, for this particular project, and after some consideration, MooTools[**] was considered the best option. The reasons for this decision are:

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 it is 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.

[*] 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

Compatible It has been tested with most browsers: Internet Explorer 6+, Firefox 2+, Opera 9+, Safari 2+ (and other Webkit-based browsers, like Chrome).

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

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

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.

2.8 Push Server: the APE Server

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.9 Mobile Web Development

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.9.1 Touchscreens

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.9.2 Webkit

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 3

Development

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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.1 How the Devices Are Placed

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 are different for each user, an algorithm must be used to placed the devices in the available space.

3.1.1 Simplified Algorithm

The restrictions that we have to comply are:

- We have 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.
- We have to place them in the most comfortable way possible, ideally using all the space we have.

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.

In the following figures we can appreciate what all of this means with real examples, using different number of elements.

(insert images with the disposition of different number of elements in the grid)

If we look at the examples below, we can guess an improvement without making the calculation severely complicate. We can 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 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.

We can prove why not 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).

$$(x-1) \cdot (x-1) < x \cdot (x-1) \tag{3.3}$$

$$x^2 - 2x + 1 < x^2 - 2x \tag{3.4}$$

$$1 < 0 \tag{3.5}$$

Then, using this final algorithm, the previous examples will change:

(insert images with the disposition of different number of elements in the grid using the final algorithm)

Using this algorithm we can calculate the height, width, vertical and horizontal offset for every element. 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.

3.1.2 Storage Positioning

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 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` [4]. 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` that is stored in every `Cookie` is composed by the four values needed for positioning the element: `offsetX`, `offsetY`, `width` and `height`. These values are percentages respect the container (the devices list) and a `Hash` example is presented in Listing 3.1.

offsetX: 15,
offsetY: 50,
height: 10,
width: 20

Listing 3.1: 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).

3.2 APE Server Installation and Configuration

In this section the installation and configuration of the APE server are defined step by step.

3.2.1 Install the Server

The APE download page [5] 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 3.2 from the same directory as the package is stored.

Listing 3.2: 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 [6]. It can be checked by visiting the url webportal.imusu.mobile.dtrd.de:6969.

3.2.2 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 [7].

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 3.3.

Listing 3.3: 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 3.4.

Listing 3.4: 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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.2 Outlook

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Appendix B

One More Thing

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea

commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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] MooTools. *MooTools Docs: Class Hash.Cookie*, Version 1.2.5. Last checked on March 2011. http://mootools.net/docs/more/Utilities/Hash.Cookie. 3.1.2

[5] Weelya. *APE Server Download*, Version 1.0. Last checked on March 2011. http://www.ape-project.org/download/APE_Server.html. 3.2.1

[6] Weelya. *APE Setup Manual*, Community Wiki. Last checked on March 2011. http://www.ape-project.org/wiki/index.php/Setup. 3.2.1

[7] Weelya. *Advanced APE Configuration*, Community Wiki. Last checked on March 2011. http://www.ape-project.org/wiki/index.php/Advanced_APE_configuration. 3.2.2

Acronyms

AJAX	Asynchronous JavaScript and XML
APE	Ajax Push Engine (see §2.8)
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
FMC	Fixed and Mobile Convergence
GPL	General Public License
GUI	Graphical User Interface
JAR	Java Archive
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
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
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
UML	Unified Modeling Language
URL	Uniform Resource Locator

VOD Video on Demand

VoIP Voice over IP

W3C WWW Consortium

WHATWG Web Hypertext Application Technology Working Group

WWW World Wide Web

XHTML Extensible HyperText Markup Language

XML Extended Markup Language

Index